ARQUITECTURAS DE SOFTWARE
LABORATORIO NRO 1: Introducción a JEE

Se creará una simple aplicación de tipo CRUD utilizando JSP, Servlets y EJB3 por medio de la plataforma JEE. Esta contendrá 4 operaciones dentro del JSP.

- Insert: Inserta datos en la tabla ACCOUNT.
- List: Lista los datos de los contactos generados.
- Delete: Elimina datos en la tabla ACCOUNT.
- Login: Realiza acceso por medio de usuario y contraseña.

OBJETIVOS

I.Objetivo General

Crear una aplicación CRUD para la gestión de cuentas de usuarios utilizando un pool de conexiones, recursos JDBC, anotaciones JPA, EJB, unidades de persistencia, Servlets y JSP.

II.Objetivos específicos

- Comprender el uso de los recursos JDBC y pool de conexiones para el manejo de conexiones y concurrencia en una base de datos.
- Conocer y comprender las anotaciones JPA para simplificar la persistencia y el mapeo de una base de datos objeto-relacional.
- Comprender el uso de EJB-Session Beans: Stateful, Stateless y Singleton para la seguridad de una aplicación transaccional.
- Utilizar Servlets para la lógica del negocio y JSP para la vista del cliente.
- Conocer la utilidad de los objetos request y response que nos proporcionan los Servlets.
- Comprender el manejo de JSP para la creación de una aplicación web.
- Aplicar una arquitectura básica de 3 o N capas para el desarrollo de una aplicación web.
- Aplicar y entender el uso de los patrones de diseño DTO y DAO.

III.Herramientas Empleadas

- NetBeans 8.1 o Superior.
- Web Browser (Firefox, Chrome).
- Java EE 7 Web, plataforma de desarrollo.
- Java Servlets.
- Java Server Pages.
- Enterprise Javabeans.
- Bootstrap
- GlassFish Server 4 (servidor de aplicaciones).
- Mysql 5.1 o superior.

IV. Arquitectura Propuesta

La aplicación está divida en 6 capas bien definidas, las cuales están representadas en la figura 1.

Backend:

El Backend está compuesta de 3 capas las cuales son:

- Capa Datos: Corresponde al motor de base de datos MySQL, que tendrá la instancia ACCOUNT, previamente creada. Acá se almacenaran los respectivos datos que lleguen por medio del patrón DAO (Data Access Object), generado en la capa de la lógica del negocio.
- Capa del Modelo: Contiene un Entity Bean que en esencia es un simple POJO (Plain Old Java Object) con anotaciones JPA para facilitar el acceso al modelo del dominio y poder enlazar los datos serializados usando el patrón DTO (Data Transfer Object). Hay que tener en cuenta que al momento de generar el Entiy Bean se realiza un mapeo objeto relacional (ORM), de forma transparente al desarrollador, lo que facilita el acceso a los datos. El otro componente es la llamada Unidad de Persistencia (P.U), que permite a través de la clase Entity Manager, realizar la conexión y transaccionalidad hacia la base de datos usando los API JPA y JTA respectivamente. Esto es importante porque facilita al desarrollador acceder al modelo del dominio a través de métodos predefinidos sin preocuparse de la complejidad que implica emplear llamadas por medio de JDBC.

- Capa de Lógica del Negocio: En esta capa se usará el patrón DAO, por medio de un Stateless Session Bean, el cual es una clase que contendrá la implementación de los principales métodos CRUD que serán expuestos a través de un Facade. La implementación se realizara en una clase genérica llamada Abstract Facade.

FrontEnd:

Está compuesta de 3 capas las cuales son:

- Capa Cliente: Corresponde a donde estará el web browser elegido por el usuario para interactuar con el sistema.
- Capa Presentación: Corresponde a las diferentes paginas generadas en JSP y usando etiquetas de HTML5 y Twitter Bootstrap que contienen los formularios y las vistas principales del sistema.
- Capa Lógica de la Aplicación: Acá se tendrá la clase Servlet que actuará como un controlador de las vistas y genera una comunicación con los métodos del DAO por medio de la inyección de dependencias usando la anotación @EJB.

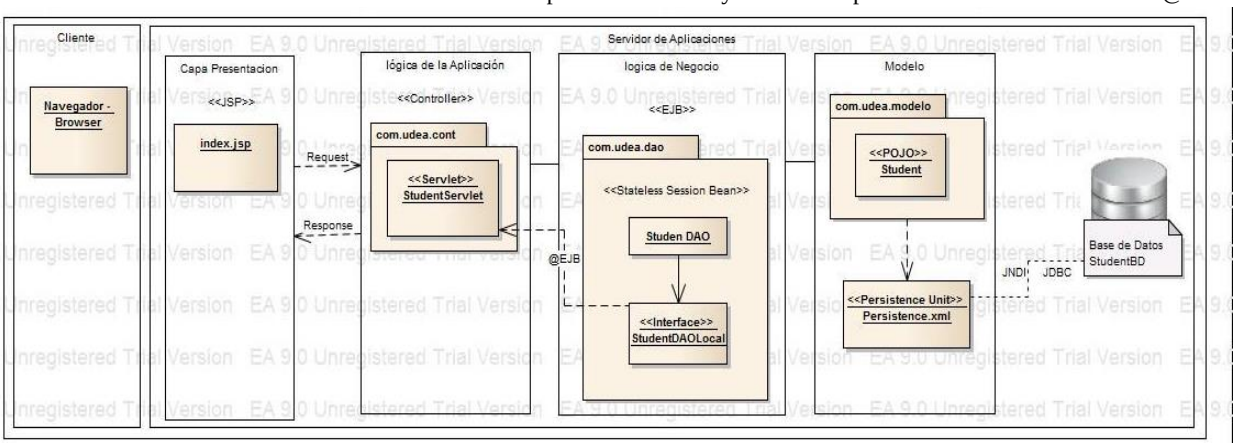


Figura 1: arquitectura de la aplicación.

V. Procedimiento

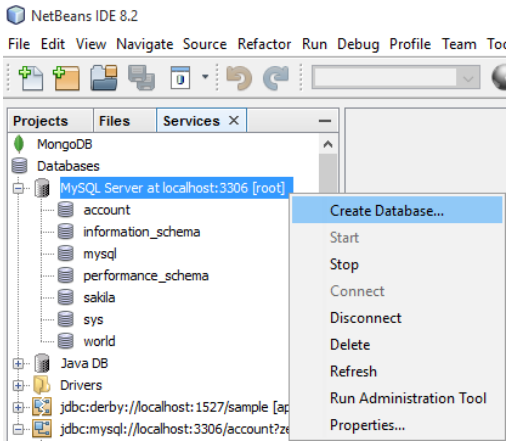
PASOS GLOBALES

1. Configurar la Base de Datos

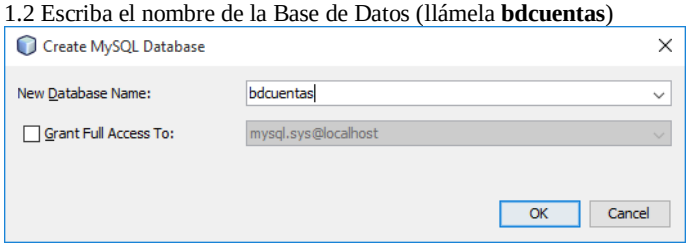
2. Conectar la Base de Datos al Servidor de Aplicaciones Glassfish.
   2.1 Crear Connection Pool
   2.2 Crear Recurso JDBC
3. Crear Proyecto Web
   3.1 Crear el modelo de clases con anotaciones JPA
   3.2 Crear componentes DAO (EJB).
4. Crear la Unidad de Persistencia
5. Crear el componente Controller (Servlet)
6. Crear la página web (JSP)
7. Desplegar la aplicación en el servidor de aplicaciones.

Configurar la Base de Datos.

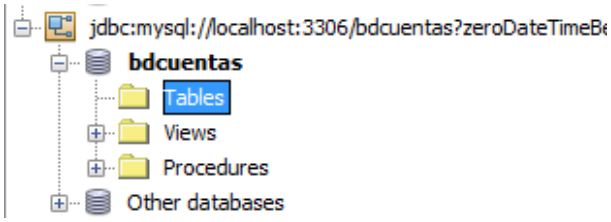
1.- Cargar Netbeans y crear la base de datos y luego realizar la conexión a la BD tal como se muestra a continuación:
1.1 En el separador **Services** de click con el botón derecho del mouse sobre el nodo MySQL Server (Asegúrese de tener previamente instalado el motor). Nota: En caso de no tener MySQL puede usar otro motor de BD. Haga click en **Create Database**. (Asegúrese que el motor está iniciado, si no es así dele click sobre Start).



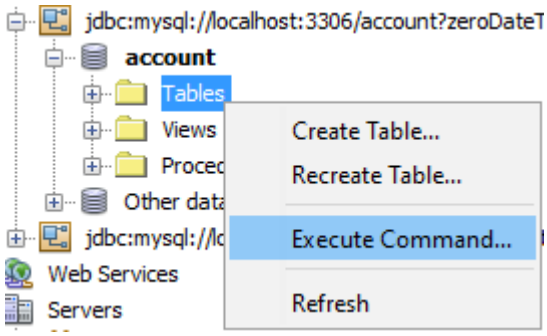
1.2 Escriba el nombre de la Base de Datos (llámela **bdcuentas**)



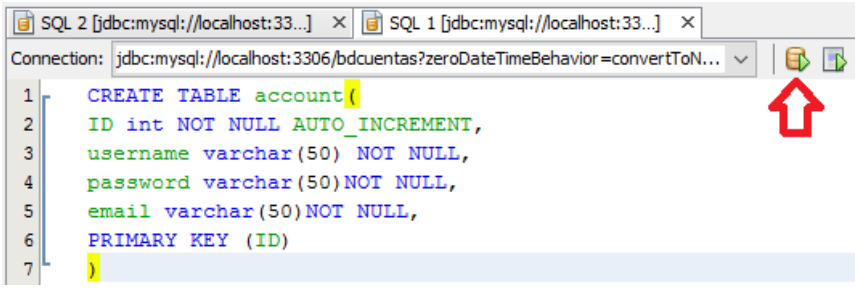
1.3 Sobre el nodo de conexión de Bases de Datos, haga clic con el botón derecho del ratón y elija la opción **Connect.** Sobre el nodo bdcuentas, seleccione a **Tables.**



1.4 Sobre **Table**s haga click con el botón derecho del ratón sobre la opción **Execute Command** (Otra opción usando un asistente sencillo si no cuenta con el DDL es Create Table.). Se crea la estructura de la tabla Account.



1.5 En el editor SQL coloque el siguiente código para crear la tabla. Por el momento no nos preocuparemos de los tipos de datos ya que esta base de datos es solo ilustrativa para nuestro ejemplo. Una vez tenga el código ejecútelo en la opción mostrada en la flecha de la siguiente figura.



En resumen la estructura de la Tabla será la siguiente:

COLUMN NAME

COLUMN NAME	TYPE	DETAIL
ID	INTEGER	PRIMARY KEY
username	VARCHAR(50)	NOT NULL
password	VARCHAR(50)	NOT NULL
email	VARCHAR(50)	NOT NULL

NOTA 1: En el momento no haremos cifrado de la constraseña por facilidad de este laboratorio. En una práctica posterior se aplicará el proceso de encriptación y desencriptación.

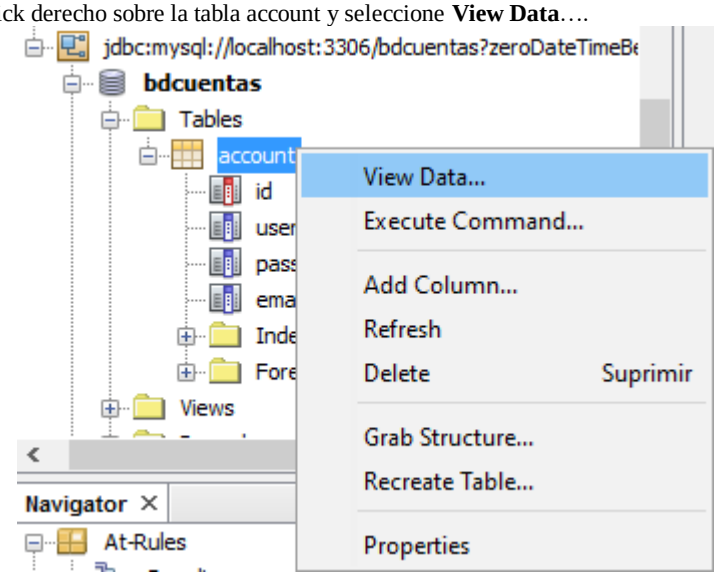
NOTA 2: Si usted lo desea puede crear la BD con otras herramientas como MySQL Workbench o PhpMyAdmin.

NOTA 2: Si usted lo desea puede agregar más campos a la BD para pruebas.

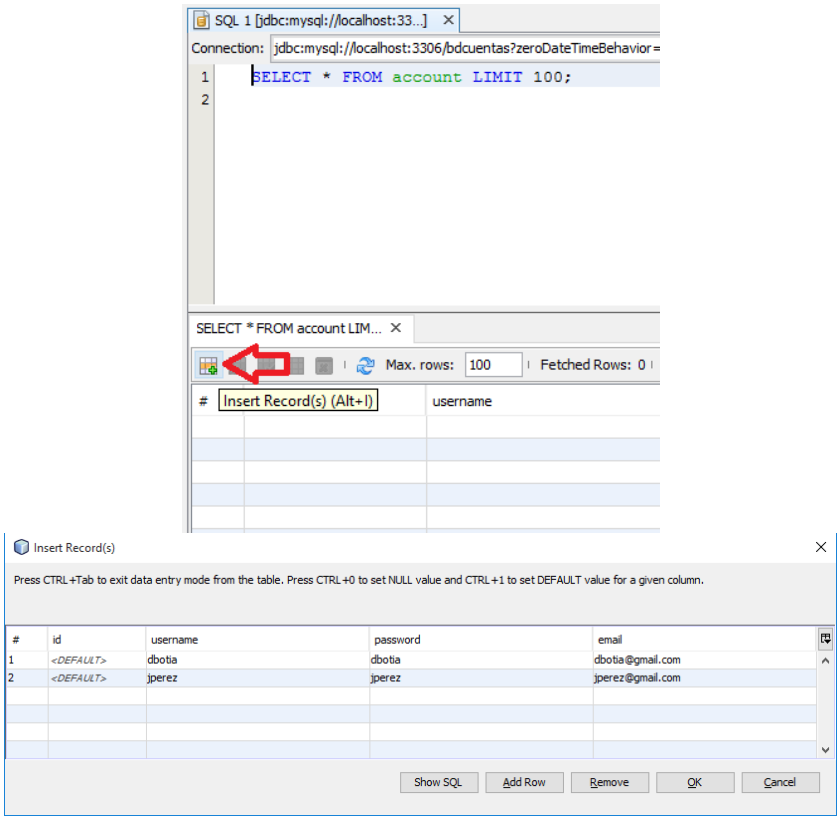
NOTA 3: En caso de usar la BD Derby (JavaDB) puede usar el siguiente script SQL para crear la tabla:

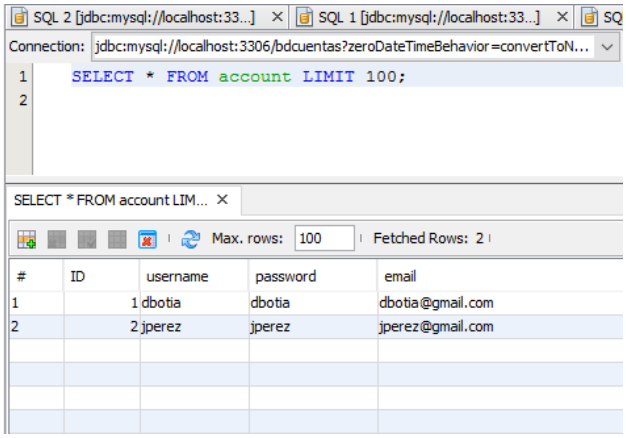
```
CREATE TABLE account(
id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
username VARCHAR(50) NOT NULL,
password VARCHAR(50) NOT NULL,
email VARCHAR(50) NOT NULL,
CONSTRAINT primary_key PRIMARY KEY (id)
);
```

Una vez creada la BD, haga click derecho sobre la tabla account y seleccione **View Data**….



En el editor SQL haga click en **Insert Record** y agregue algunos datos de prueba.

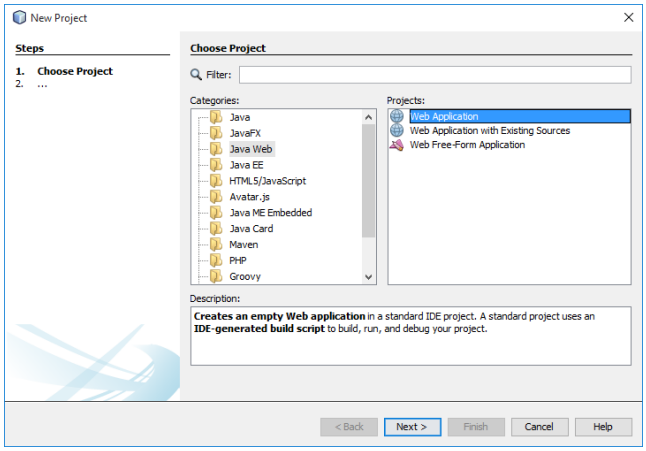




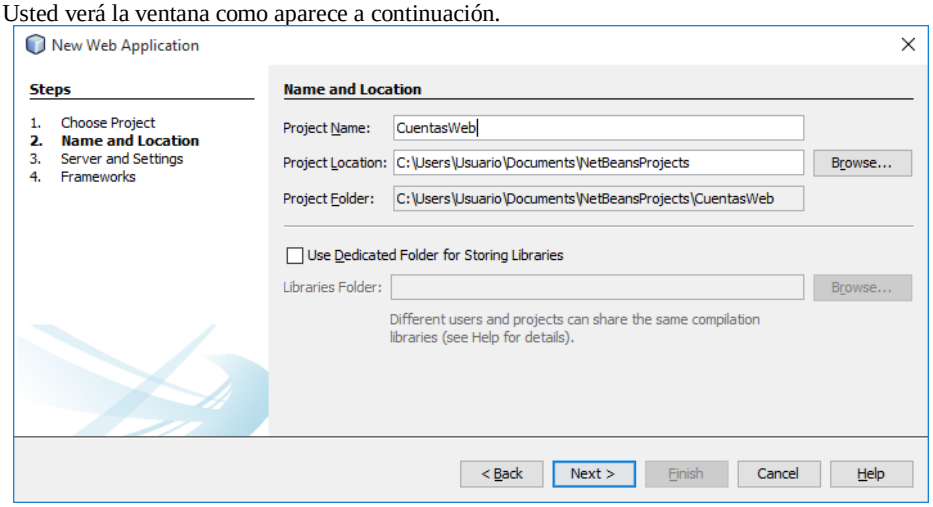
NOTA: Tenga en cuenta que el campo ID es auto incremental por lo tanto no necesita ingresarlo.

2. Creando la aplicación:

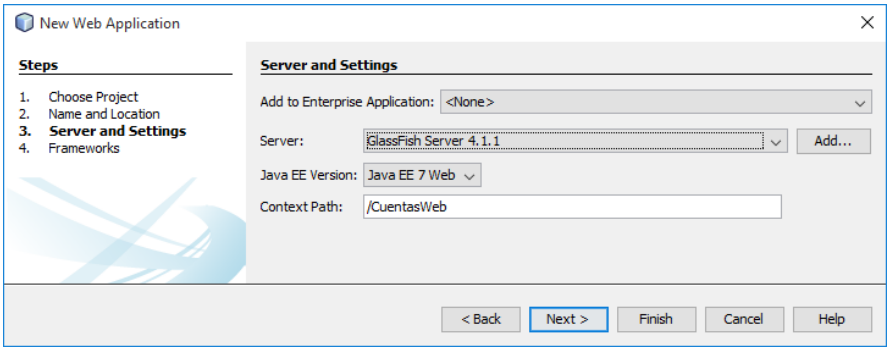
- Inicie NetBeans.
- Seleccione el menú **File -> New project**.
- Seleccione **categories** y luego **Java Web**
- Ahora seleccione **Web Application** y haga clic en el botón **next.**



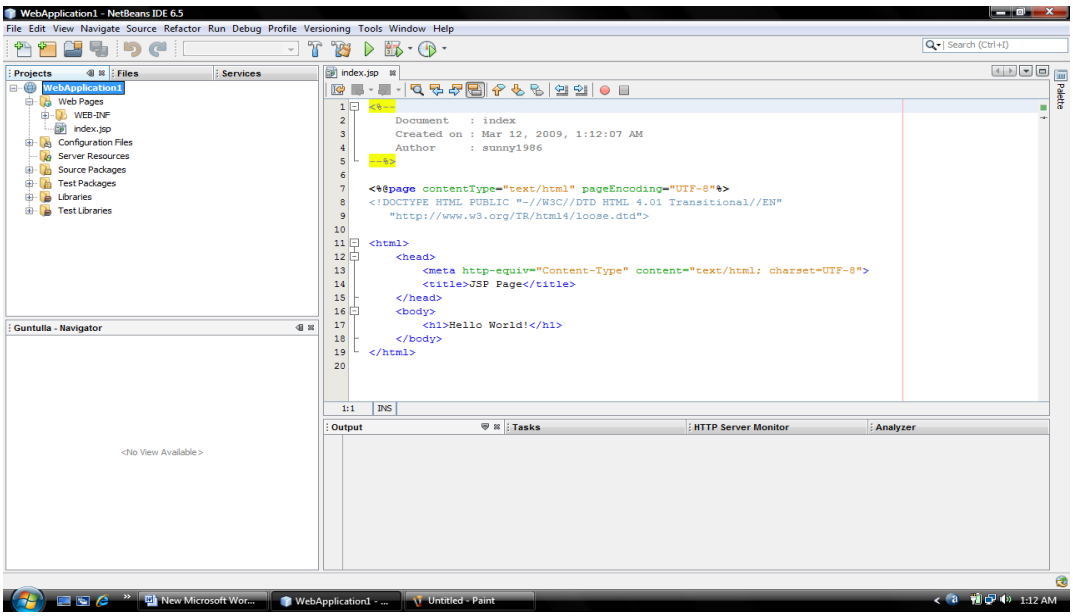
- Usted verá la ventana como aparece a continuación.



- Aquí puede especificar el nombre del proyecto (En este caso la llamaremos **CuentasWeb**) y la ruta donde almacenar los proyectos.
- Cuando usted da en el botón siguiente se mostrará la configuración del servidor de aplicaciones.
- Seleccione como servidor de aplicaciones a GlassFish.



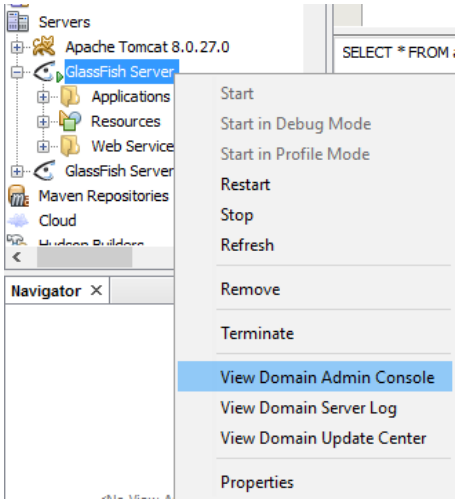
- Después de dar al botón siguiente se mostrará la ventana para seleccionar un framework.
- No se utilizará frameworks aquí, así que no seleccione y haga click en Finalizar. Se creará nuestro proyecto.



- La página index.jsp se creará automáticamente por defecto.

4. Creando un pool de conexiones a la BD:

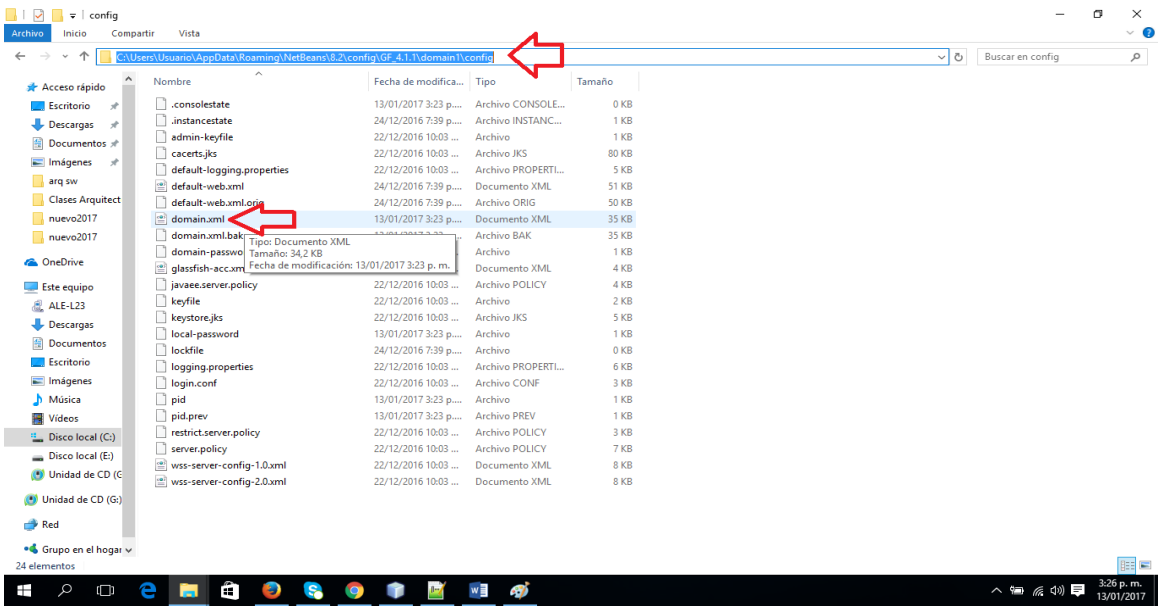
Necesitamos crear un Pool de Conexiones para crear un recurso JDBC desde el servidor de aplicaciones a MySQL. Para esto vaya al separador **Services** del proyecto. Haga click en el nodo **Servers → Glassfish Server** luego haga click derecho y seleccione **Start**. Una vez iniciado el servidor seleccione ahora la opción **View Domain admin Console**. Ver la siguiente figura:



Se iniciará la consola de administración principal de Glassfish en el navegador que tenga por defecto. Dentro de todas las opciones que aparecen nos interesan los nodos JDBC Resources y JDBC Connection Pools.

Nota Especial: Debido a un bug reportado por los usuarios en la versión de Glassfish 4.1.1, se recomienda hacer el siguiente procedimiento para agregar la configuración del Pool.

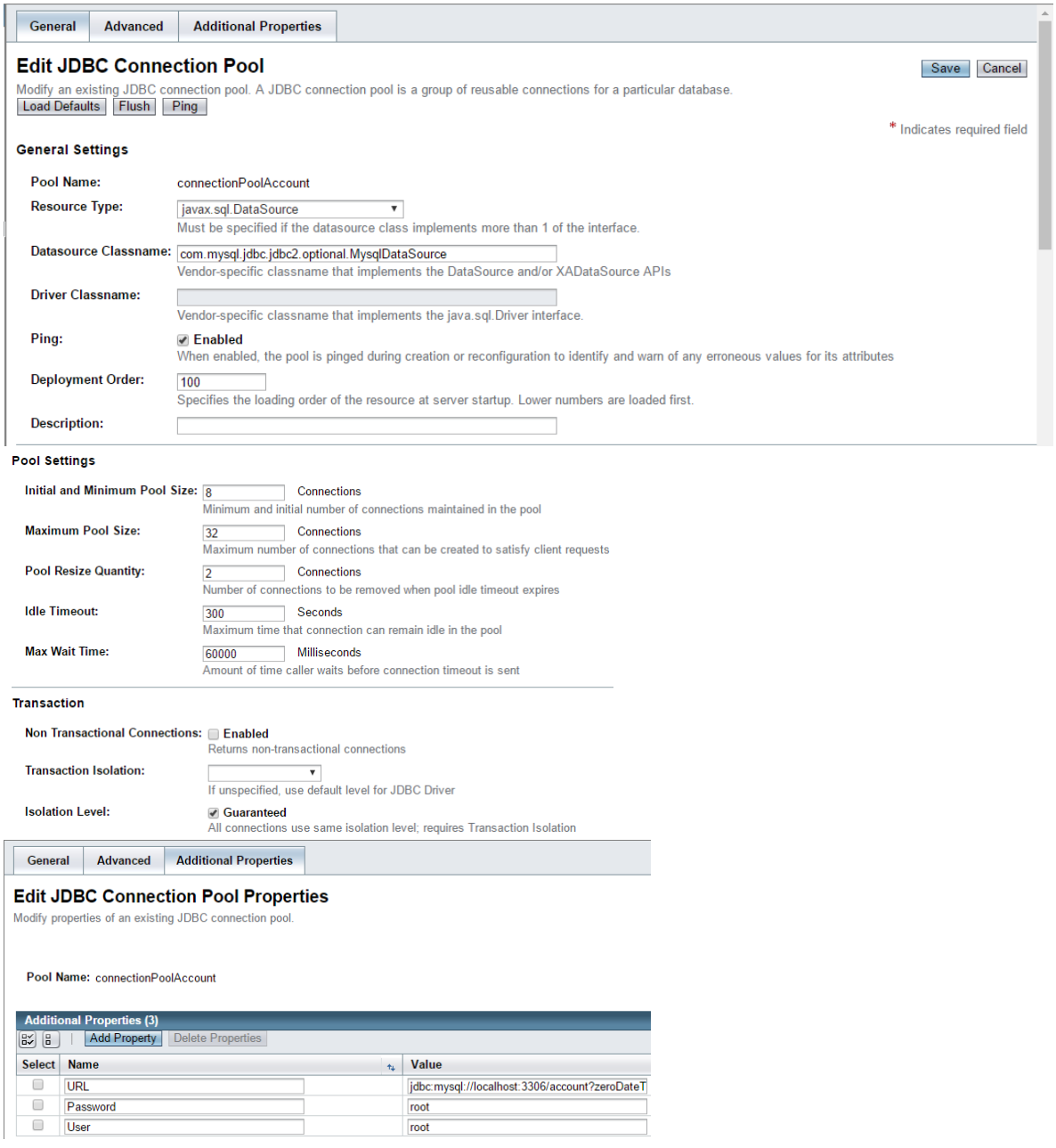
Ubique el archivo **domain.xml** que se encuentra en la ruta de configuración de glassfish. (Puede verlo mirando las propiedades de glassfish en Netbeans).



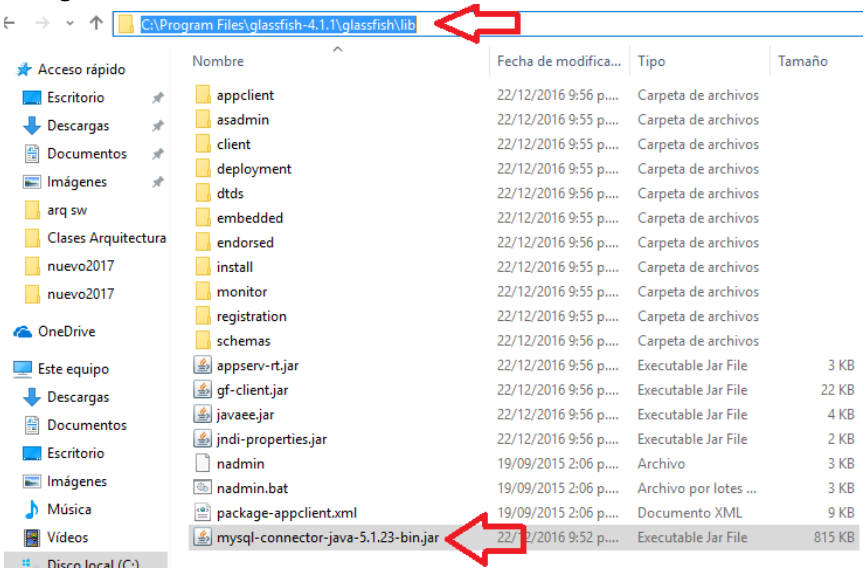
Agregue en la sección de Pool de Conexiones JDBC del archivo la siguiente sentencia:

```
<jdbc-connection-pool datasource-classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" name="connection-poolAccount"
wrap-jdbc-objects="false" connection-validation-method="auto-commit" res-type="javax.sql.DataSource">
    <property name="URL" value="jdbc:mysql://localhost:3306/bdcuentas?zeroDateTimeBehavior=convertToNull"></property>
    <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
    <property name="Password" value="root"></property>
    <property name="portNumber" value="3306"></property>
    <property name="databaseName" value="bdcuentas"></property>
    <property name="User" value="root"></property>
    <property name="serverName" value="localhost"></property>
  </jdbc-connection-pool>

<jdbc-resource pool-name="connection-poolAccount" jndi-name="jdbc/account"></jdbc-resource>
```

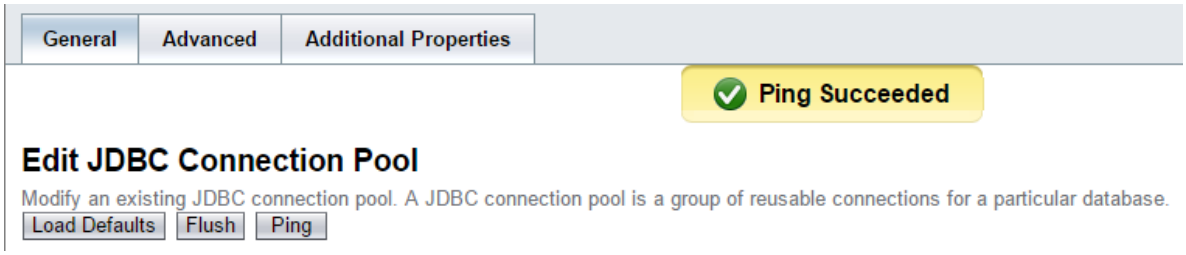
Una vez guardado los cambios en el archivo proceda a reiniciar el servidor de aplicaciones desde Netbeans. Una vez reiniciado el servidor abra de nuevo la consola de administración y verifique que se hayan guardado los respectivos cambios en el Pool de Conexiones JDBC. (Explore y estudie cada una de las opciones que se presentan).


General
Advanced
Additional Properties
Edit JDBC Connection Pool
Save
Cancel
Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.
Load Defaults
Flush
Ping
* Indicates required field
General Settings
Pool Name:
connectionPoolAccount
Resource Type:
javax.sql.DataSource
Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:
com.mysql.jdbc.jdbc2.optional.MysqlDataSource
Vendor-specific classname that implements the DataSource and/or XADataSource APIs
Driver Classname:
Vendor-specific classname that implements the java.sql.Driver interface.
Ping:
Enabled
When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
Deployment Order:
100
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.
Description:
Pool Settings
Initial and Minimum Pool Size:
8
Connections
Minimum and initial number of connections maintained in the pool
Maximum Pool Size:
32
Connections
Maximum number of connections that can be created to satisfy client requests
Pool Resize Quantity:
2
Connections
Number of connections to be removed when pool idle timeout expires
Idle Timeout:
300
Seconds
Maximum time that connection can remain idle in the pool
Max Wait Time:
60000
Milliseconds
Amount of time caller waits before connection timeout is sent
Transaction
Non Transactional Connections:
Enabled
Returns non-transactional connections
Transaction Isolation:
If unspecified, use default level for JDBC Driver
Isolation Level:
Guaranteed
All connections use same isolation level; requires Transaction Isolation
General
Advanced
Additional Properties
Edit JDBC Connection Pool Properties
Modify properties of an existing JDBC connection pool.
Pool Name: connectionPoolAccount
Additional Properties (3)
Add Property
Delete Properties
Select
Name
Value
URL
jdbc:mysql://localhost:3306/account?zeroDateT
Password
root
User
root

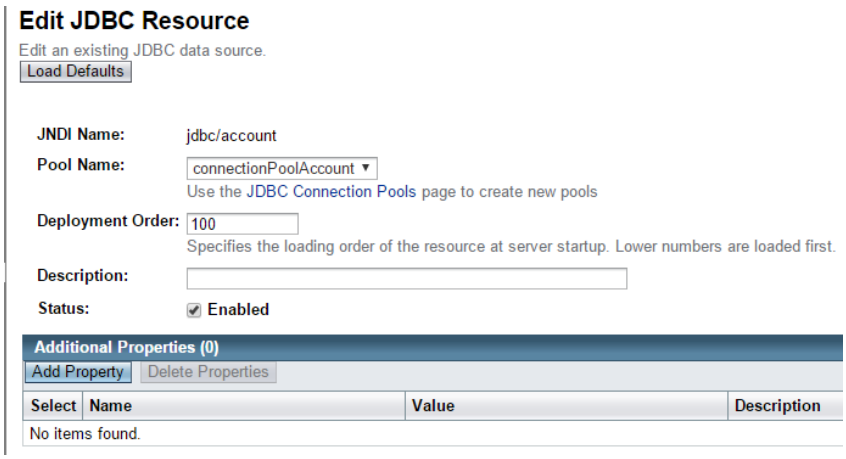
Para poder comprobar la correcta configuración del pool, debemos hacer una prueba de conexión, para hacerlo debemos copiar el JAR de MySQL en la capeta de librerías de Glassfish en donde se encuentre instalado el servidor. Por ejemplo **C:\Program Files\glassfish-4.1.1\glassfish\lib**



Ahora dentro de la consola web de Glassfish en las propiedades generales del **Connection Pool** haga click en el botón **Ping** para probar la conexión a MySQL. Si es exitoso aparece el mensaje **Ping Succeeded.**



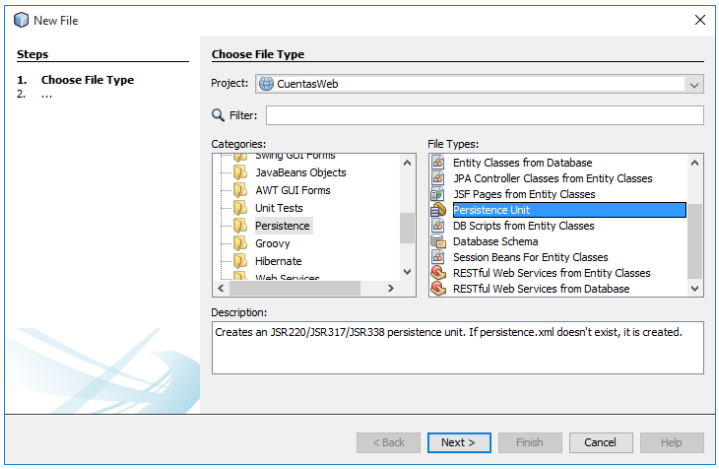
El siguiente paso es crear un recurso JDBC, en este caso se debe crear una referencia al respectivo recurso por medio del **API JNDI** (Java Naming And Directory Interface), el cual fue configurado en el archivo domain.xml como **jdbc/account**.



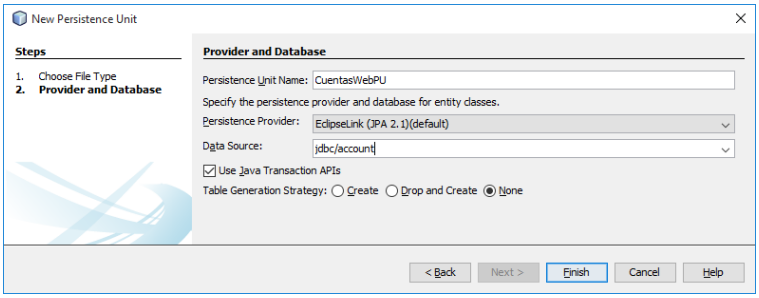
Creación de la Unidad de Persistencia

Agregue la Unidad de Persistencia (P:U) como se presenta a continuación:

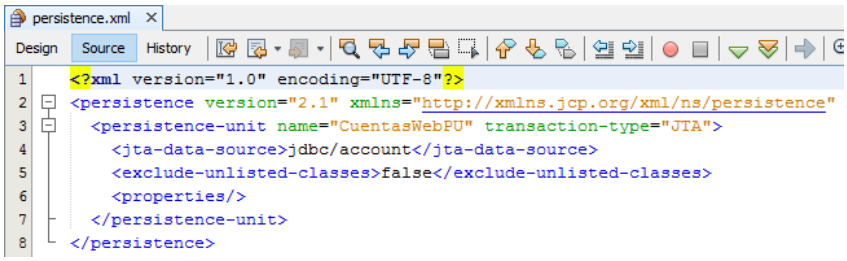
1 Haga click derecho sobre el proyecto y luego seleccione **New → Other → Persistence → Persistence Unit.**



2. En la ventana que aparece deje el nombre que aparece por defecto a la P.U, el proveedor de persistencia será **Eclipse Link (JPA)** aunque puede agregar otro si lo desea. En el **Data Source** seleccione el respectivo **JNDI** que se creó previamente para glassfish, deje activado el API de **JTA (Java Transaction API)** y en la estrategia de generación de tablas en **None**, debido a que ya se tiene previamente creado la tabla en la Base de datos.



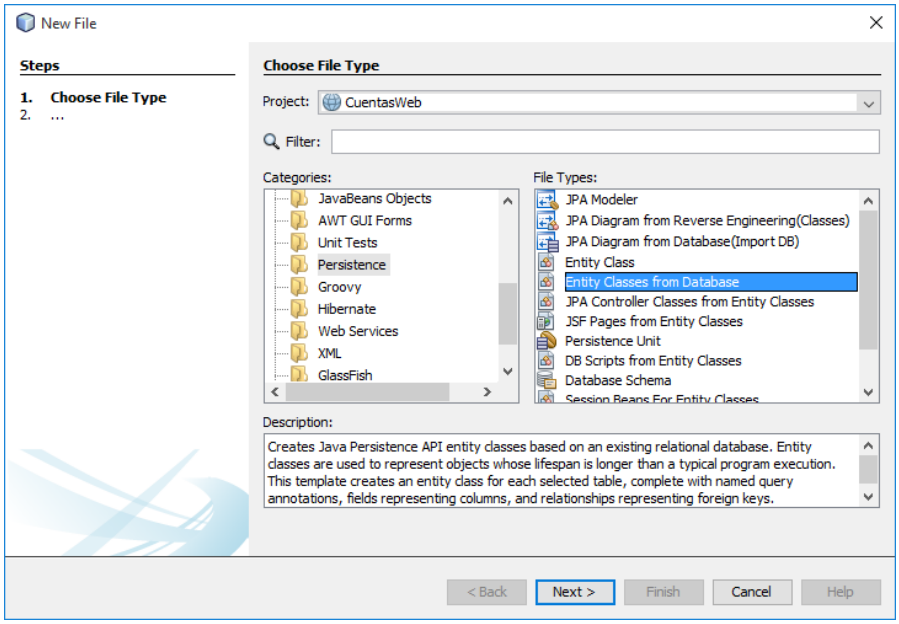
Una vez creado podrá observar el contenido del archivo XML.



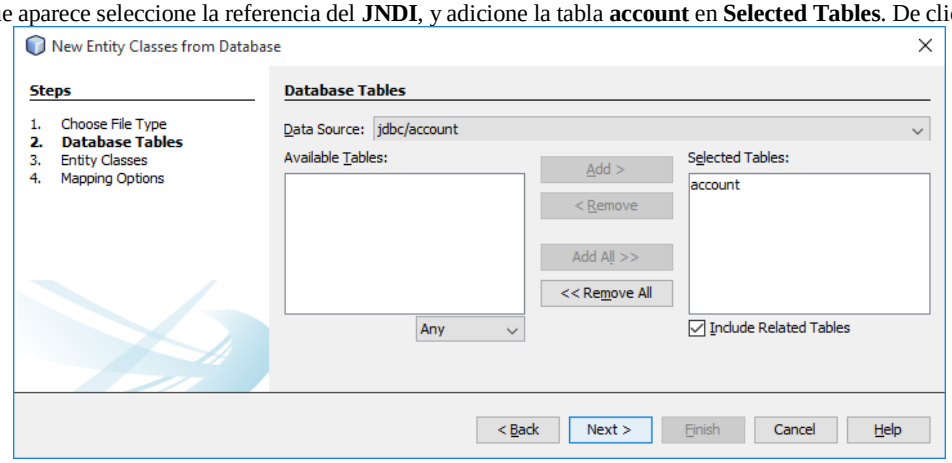
Creación del Entity Bean

A continuación se creará una clase Java el cual permitirá crear el POJO Account. Esta clase representará el modelo del dominio de persistencia.

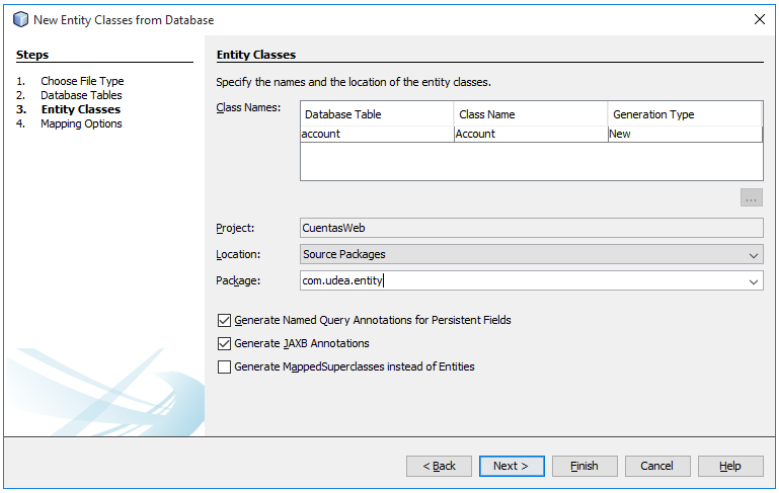
1. Haga click derecho sobre el proyecto y seleccione **New → Other→ Persistence → Entity Classess from Database**.



2. En la ventana que aparece seleccione la referencia del **JNDI**, y adicione la tabla **account** en **Selected Tables**. De click en el botón Next.

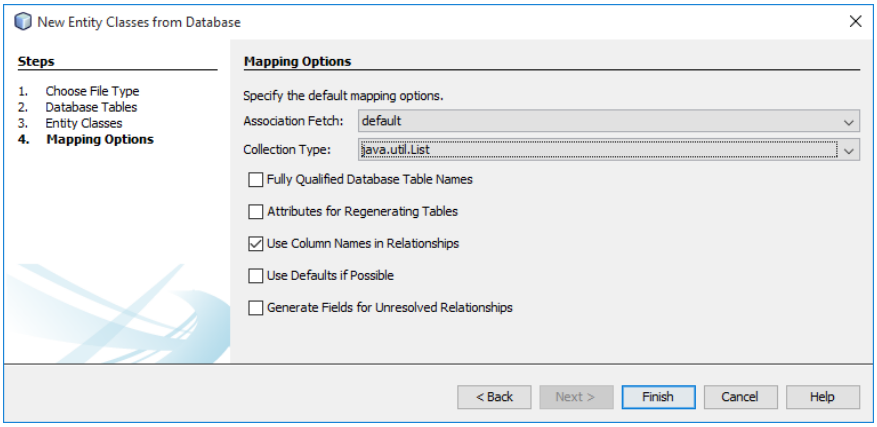


3. En esta ventana aparece como se realizará el Mapeo entre la tabla de la BD y la Clase del Modelo del dominio. Coloque un nombre al paquete, y asegúrese de seleccionar las opciones **Generate Named Query Annotation for Persistence Fields**, que creará unas consultas básicas con **JPQL** para cada atributo del POJO y **Generate JAXB Annotations** para que la estructura de los datos de nuestra clase sea generado en XML.



4. Por ultimo en las opciones de Mapeo puede colocar el respectivo **Association Fetch (Eager o Lazy)** y en **Collection Type** dejaremos la clase **java.util.List**. Las demás opciones puede dejarlas como están por defecto.

Nota: Investigue cual es la diferencia entre los tipos de asociación Eager o Lazy.



En el código de la clase del Entity Bean llamado Account podrá observar varias anotaciones JPA como **@Entity, @Table, @NamedQueries**, etc; que indicaran el comportamiento del Entity Bean.

NOTA: Recuerde buscar el significado de cada anotación empleada en este laboratorio. También puede observar que se generan automáticamente los respectivos setters y getters asi como algunos **idioms** como el método **toString(), Equals() y hashCode()**

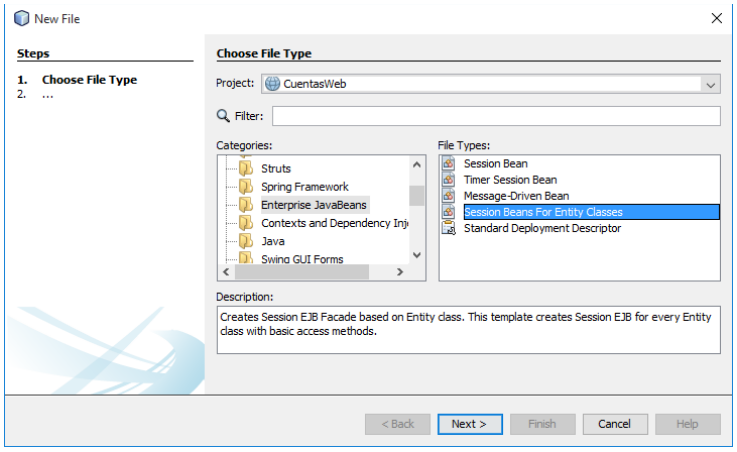
persistence.xml | Account.java

Source | History

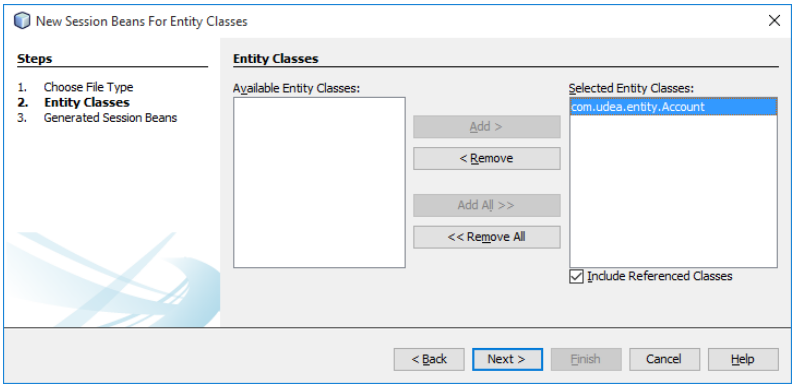
```
26    @Entity
27    @Table(name = "account")
28    @XmlRootElement
29    @NamedQueries({
30        @NamedQuery(name = "Account.findAll", query = "SELECT a FROM Account a")
31        , @NamedQuery(name = "Account.findById", query = "SELECT a FROM Account a WHERE a.id = :id")
32        , @NamedQuery(name = "Account.findByUsername", query = "SELECT a FROM Account a WHERE a.username = :username")
33        , @NamedQuery(name = "Account.findByPassword", query = "SELECT a FROM Account a WHERE a.password = :password")
34        , @NamedQuery(name = "Account.findByEmail", query = "SELECT a FROM Account a WHERE a.email = :email")})
35    public class Account implements Serializable {
36
37        private static final long serialVersionUID = 1L;
38        @Id
39        @GeneratedValue(strategy = GenerationType.IDENTITY)
40        @Basic(optional = false)
41        @Column(name = "ID")
42        private Integer id;
43        @Basic(optional = false)
44        @NotNull
45        @Size(min = 1, max = 50)
46        @Column(name = "username")
47        private String username;
48        @Basic(optional = false)
49        @NotNull
50        @Size(min = 1, max = 50)
51        @Column(name = "password")
52        private String password;
53        // @Pattern(regexp="[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[
54        @Basic(optional = false)
```

A continuación agregue un EJB de tipo **Session Bean** el cual representará el patrón DAO de nuestra capa de Lógica del negocio. Este Bean será de tipo **Stateless** y con interfaz **Local**. Ubíquelo en el paquete **com.udea.ejb**.

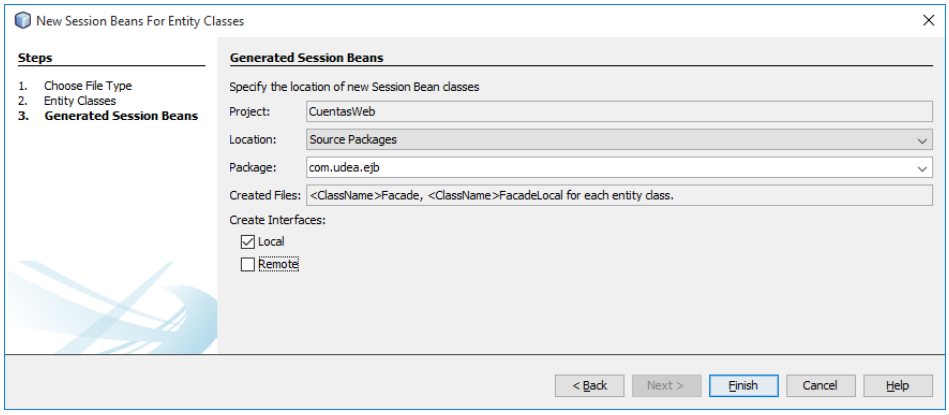
Para crearlo haga click derecho sobre el nodo del proyecto luego **New → Other → Enterprise JavaBeans → Session Bean for entity Clasess.**



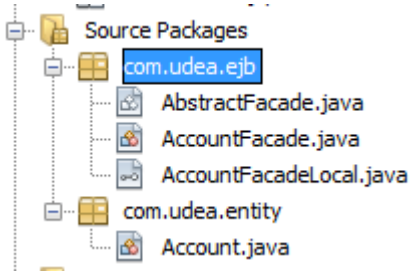
En la ventana que aparece arrastre la clase entidad hacia la derecha y de click en el botón next.



En la siguiente ventana coloque el nombre al paquete en este caso **com.udea.ejb**, y en crear interfaces seleccione de tipo **Local**. De click en el boton Finish



Como puede observar se van a crear 2 clases (Abstract Facade y AccountFacade) y 1 interface Java (AccountFacadeLocal).



Por cada método de negocio se colocará automáticamente la referencia hacia la interfaz que actuará como un **Facade** contra el servlet. Como puede observar a continuación la inteface local tiene expuesto los principales métodos del negocio que serán implementados en la clase **Abstract Facade**.

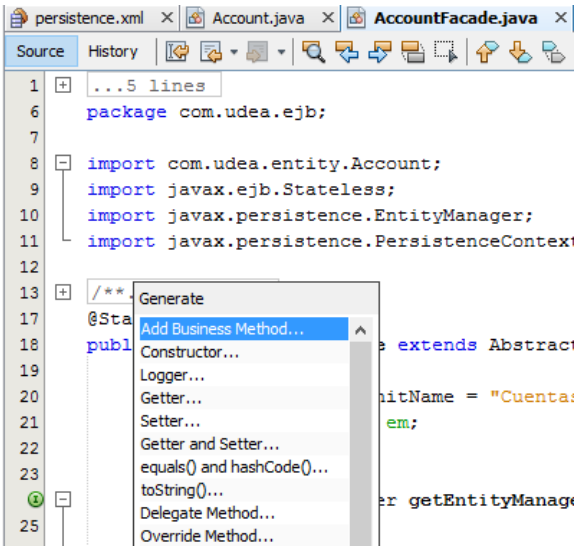
```java
package com.udea.ejb;

import com.udea.entity.Account;
import java.util.List;
import javax.ejb.Local;

/**...4 lines */
@Local
public interface AccountFacadeLocal {

    void create(Account account);

    void edit(Account account);

    void remove(Account account);

    Account find(Object id);

    List<Account> findAll();

    List<Account> findRange(int[] range);

    int count();

}
```

El **Session Bean** AccountFacade, es la clase que usa el patrón DAO y se extiende de la clase **AbstractFacade** , además genera una referencia a la unidad de persistencia usando la anotación **@PersistenceContext**, que permite llamar a la clase **EntityManager** que facilitará realizar las transacciones a la Base de Datos.

```java
package com.udea.ejb;

import com.udea.entity.Account;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**...4 lines */
@Stateless
public class AccountFacade extends AbstractFacade<Account> implements AccountFacadeLocal {

    @PersistenceContext(unitName = "CuentasWebPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public AccountFacade() {
        super(Account.class);
    }

}
```

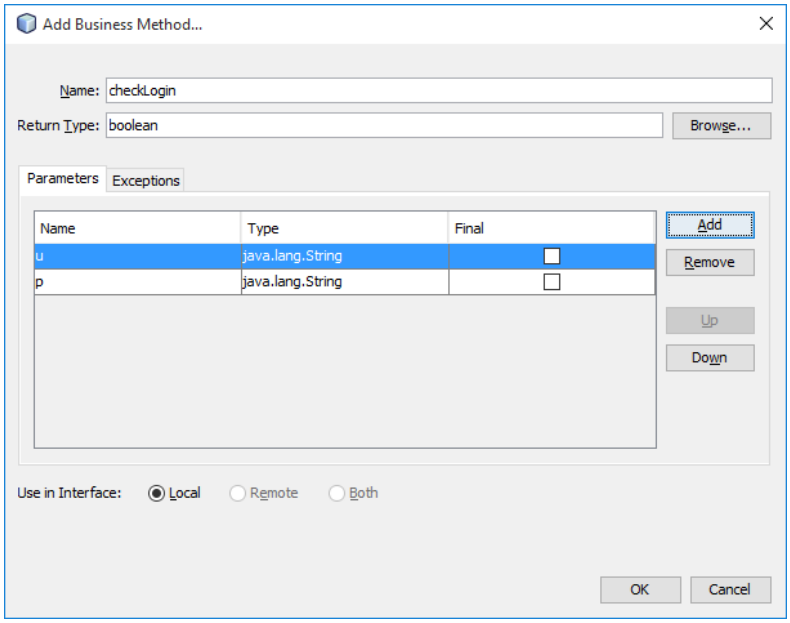
Un método necesario para realizar el chequeo por usuario y contraseña será agregado en el DAO. Para ello agregaremos el método **checkLogin()** como se presenta a continuación:

1 Sobre el editor de código con la clase **AccountFacade** abierta, haga click derecho y en el menú emergente seleccione **Add Business Method**..



En la ventana que aparece coloque el nombre del método (**checkLogin)**, el tipo de retorno será **boolean** y los parámetros de tipo **String** serán el **usuario** y **password**.

Nota: El atributo Final indica que una variable es de tipo constante: no admitirá cambios después de su declaración y asignación de valor. En este caso final determina que un atributo no puede ser sobreescrito o redefinido.

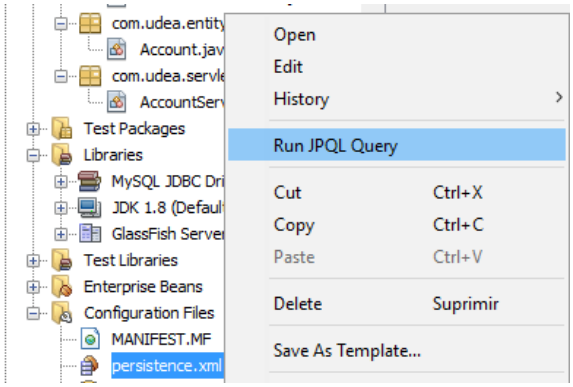


Sobre el esqueleto del método adicione el siguiente código. Note que se usará la Clase **Query** que permite enlazar una consulta **JPQL**, en este caso consular los objetos de la clase **Account** que correspondan a los valores de los atributos Username y Password.

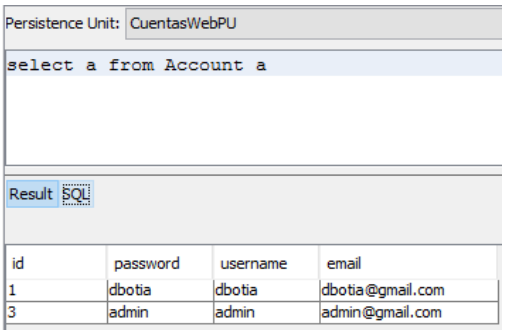
```
@Override
public boolean checkLogin(String u, String p) {
    Query q = em.createQuery("select a from Account a "
            + "where a.username=:u and a.password=:p");
 q.setParameter("u", u);
 q.setParameter("p", p);

    return q.getResultList().size()>0;
}
```

Nota: Si usted desea comprobar una consulta con JPQL se recomienda hacer los siguientes pasos:

1- Haga click derecho sobre el archivo Persistence.xml en el nodo del proyecto.
2- Seleccione la opción Run JPQL Query como se muestra en la siguiente figura:



En el editor que aparece coloque la consulta (Recuerde que la sintaxis es diferente a **ANSI SQL**, que corresponde al modelo relacional). Al ejecutar la consulta aparecerá los resultados obtenidos de la instancia de los objetos de la clase. Por ejemplo si se coloca la sentencia **select a from Account a** , se obtiene la siguiente salida:



Para la consulta JPQL del método **checkLogin(),** la probamos con la siguiente sentencia: **select a from Account a where a.username="Usuario" and a.password="password"** y obtenemos el siguiente resultado: