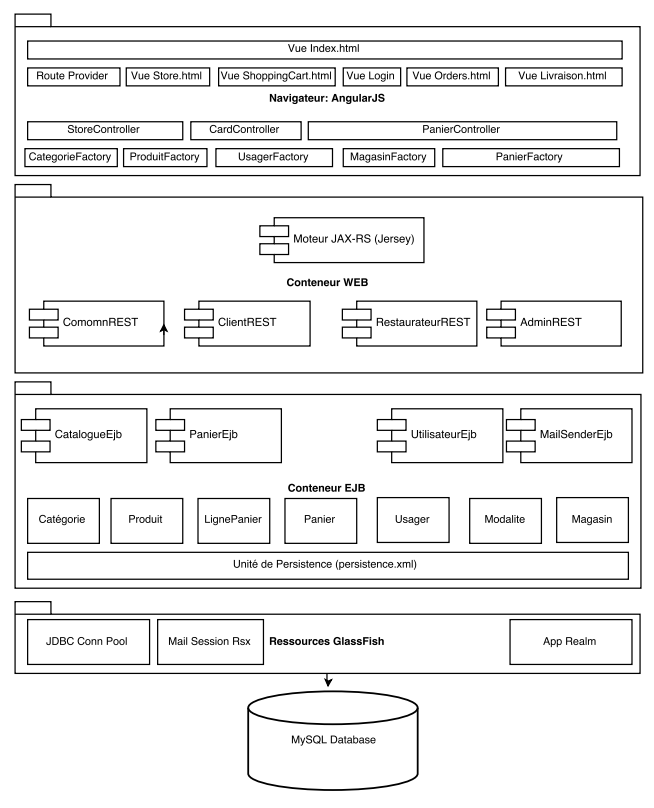
Vue Synoptique de l’architecture du système

La figure ci-dessous présente une vue synoptique des composants de l’application SushiParty. 

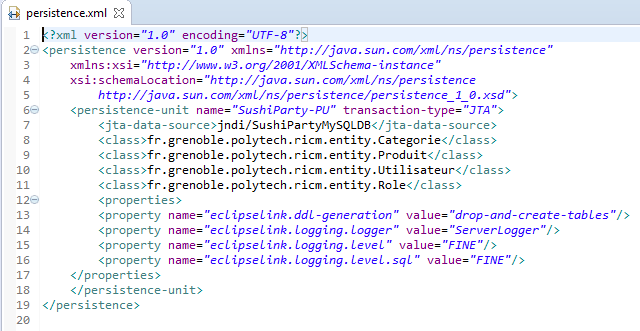
## Ressources GlassFish

Pour faire fonctionner le projet nous avons besoin d’un coté d’une base de donnée et d’un fournisseur de persistence.

Pour abriter la base de données du projet SushiParty nous avons choisi le SGBD ***mysql*** pour sa gratuité, sa simplicité et sa relative robustesse. Pour ce qui est du fournisseur de persistence, nous avons utilisé le fournisseur qui vient par défaut avec ***Glassfish 4.1*** c'est-à-dire ***EclipseLink.***

Du coté du SGBD ***mysql*** nous avons juste besoin de créer la base de données sans penser aux tables, aux séquences ou aux vues qu’elle abritera.

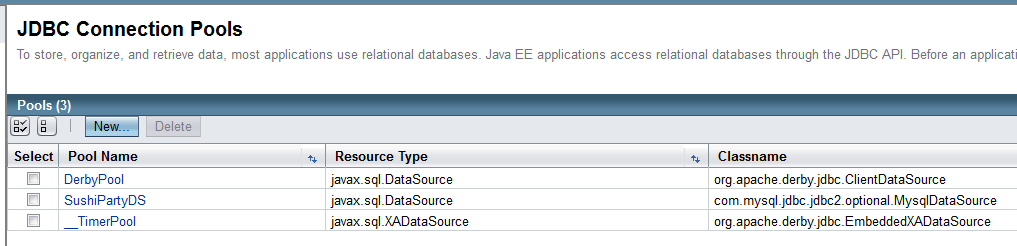
La création des tables et des séquences de l’application SushiParty se feront par le founisseur de persistence donc voici un extrait du fichier de configuration



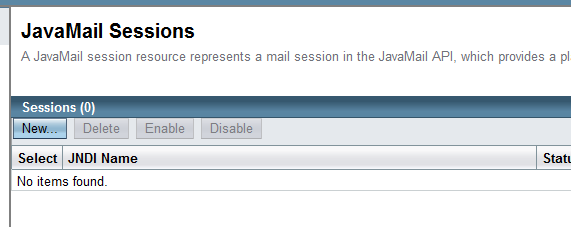
La balise <property name=*"eclipselink.ddl-generation"* value=*"drop-and-create-tables"*/> permet de détruire et de recréer les objets de la base de données

Le lien entre Glassfish et mysql est implémenté par le piote jdbc mysql ***mysql-connector-java-5.1.36-bin.jar***

Par la suite un pool JDBC est créé pour me permettre à l’application de se connecter à la base de données. Ce pool JDBC est par la suite exporté dans l’annuaire JDNI de Glassfish pour être utilisable par notre unité de persistence (cf fig1)



Pour gérer l’envoi des mails nous avons configuré le client mail disponible dans glassfish . Nous obtenons ainsi une ressource glassifsh injectable dynamique dans un bean comment l’illustre la figure suivante.



Pour la gestion des utilisateurs nous avons choisi d’utiliser les fonctions natives disponibles dans glassfish. C'est-à-dire que nous avons sous traité les fonctions d’authentification et d’autorisation à glassfish.

Il a suffit pour cela de créer un realm (domaine de sécurité comportant tous les utilisateurs de notre application SushiParty puis de le définir comme dans les descriptuers de déploiement comme realm par défaut

## Projet EJB

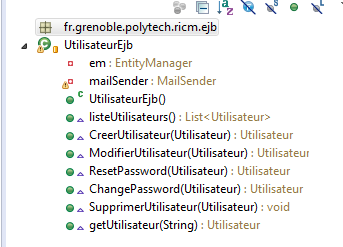
Le projet EJB de notre application comporte trois catégories d’objets : Les EJB comportant à la fois le code et les interfaces locales des EJB et les Entités JPA.

Nous avons architecturé la couche EJB de notre application autour de quatre EJB :

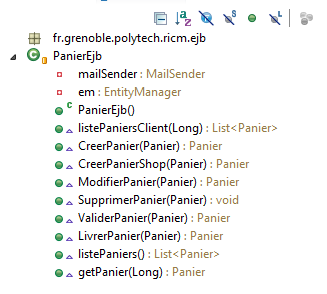
* ***CatalogueEJB*** : Cette EJB offre les fonctions permettant de manipuler le catalogue de l’application SushiParty. Pour cela elle permet de créer, consulter, modifier et supprimer les catégories d’article s et les articles vendus sur le site.



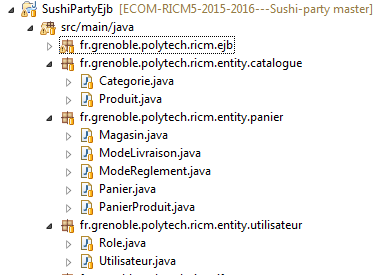
* ***UtilisateurEJB*** : Cette EJB offre les fonctions permettant de manipuler le les comptes utilisateurs et les mots des passes des utilisateurs du système. Pour cela elle permet de créer, consulter, modifier et supprimer les comptes utilisateur sur le site. Et de les affecter à un des trois rôles systèmes : Client, Manager et Administrateur. Cette EJB offre aussi les fonctions de gestion de mots de passes telque la réinitialisation, le déverrouillage etc.



* ***PanierEJB*** : Cette EJB offre les fonctions permettant de manipuler le panier de l’application SushiParty. Pour cela elle permet de créer, consulter, modifier et supprimer un panier. Les fonctions telques la validation et la livraison sont également disponibles



La couche des entités JPA contient uniquement les différentes entités qui sont manipulés par les EJB présentés plus haut. La liste complète des entités qu’il nous a fallut implémenté est donné dans le schéma de la figure ci dessous.



## Projet WEB

Le projet WEB de notre application comporte deux catégories de composants REST et les composants AngularJS.

Les composants REST s’appuient sur le moteur REST Jersey qui implémente la norme JAX-RS. Le moteur REST Jersey s’enregistre auprès du conteneur web via une servlet. Les classes REST portent à leur tour des annotations @Path, @POST, @GET, @PUT et @DELETE pour faire correspondre les requette HTTP aux méthodes des classes REST. Dans cette partie nous avons implémenté une classe REST par roles pour faciliter le contrôle d’accès.



Les composants AngularJS sont reparties dans les trois catégries suivantes : Vues, controlleurs et Modèles. La liste complète des composants de cette partie sont décris dans la figure suivante

