Adam Christophe

Aissanou Sarah

Fotsing Eric Michel

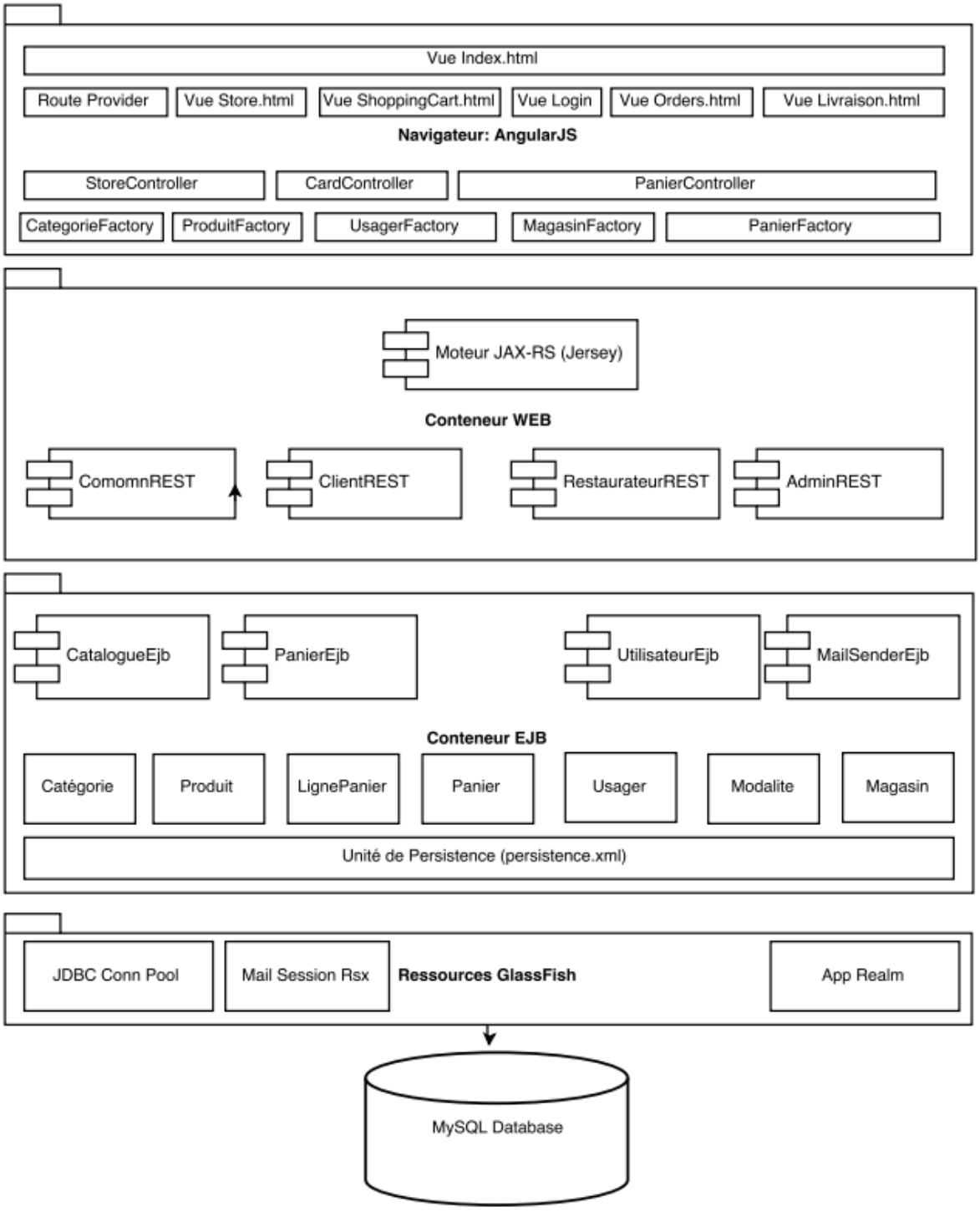
Barthelemy Romain

Rapport système du projet

Sushi Party

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'une part d'une base de donnée et d'une autre, d'un fournisseur de persistance.

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 persistance, nous avons utilisé le fournisseur qui vient par défaut avec ***Glassfish 4.1*** c'est-à-dire ***EclipseLink.***

Du côté 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 fournisseur de persistance dont voici un extrait du fichier de configuration

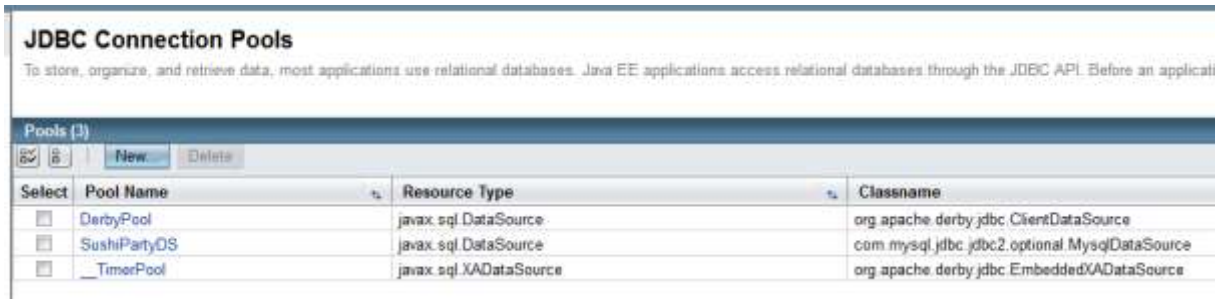
persistence.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
5          http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
6      <persistence-unit name="SushiParty-PU" transaction-type="JTA">
7          <jta-data-source>jndi/SushiPartyMySQLDB</jta-data-source>
8          <class>fr.grenoble.polytech.ricm.entity.Categorie</class>
9          <class>fr.grenoble.polytech.ricm.entity.Produit</class>
10         <class>fr.grenoble.polytech.ricm.entity.Utilisateur</class>
11         <class>fr.grenoble.polytech.ricm.entity.Role</class>
12         <properties>
13         <property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>
14         <property name="eclipselink.logging.logger" value="ServerLogger"/>
15         <property name="eclipselink.logging.level" value="FINE"/>
16         <property name="eclipselink.logging.level.sql" value="FINE"/>
17     </properties>
18     </persistence-unit>
19 </persistence>
20
```

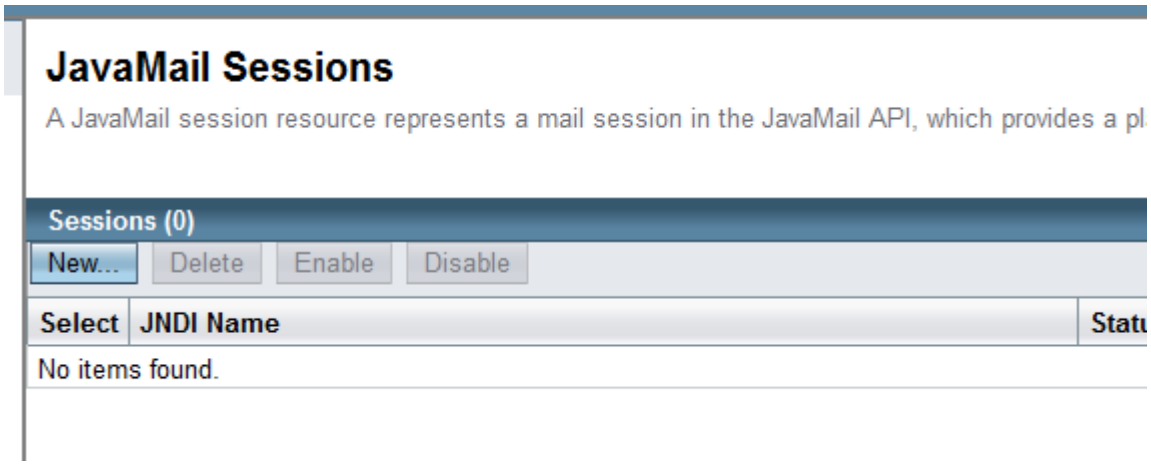
La balise `<property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>` permet de détruire et de recréer les objets de la base de données

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

Par la suite un pool JDBC est créé pour 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 persistance (cf fig1)



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



En ce qui concerne 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 suffi pour cela de créer un realm (domaine de sécurité comportant tous les utilisateurs de notre application SushiParty) puis de le définir comme dans les descripteurs de déploiement un realm par défaut

Projet EJB

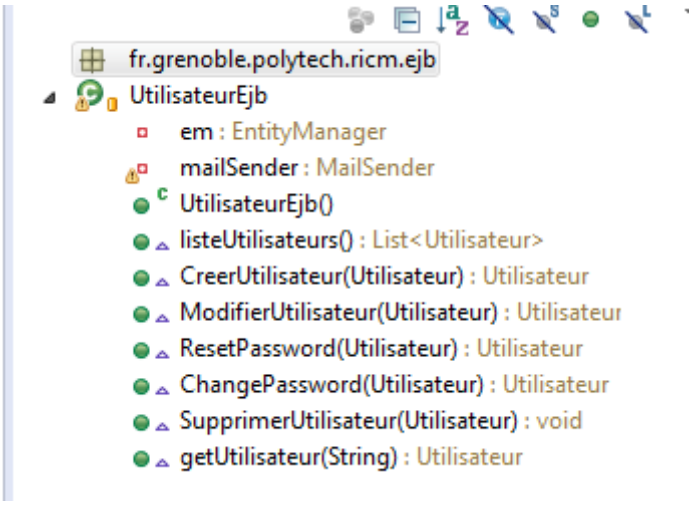
Le projet EJB de notre application comporte trois catégories d'objets : Les EJB comportant à la fois le code et les interfaces locales des EJB et les Entités JPA.

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

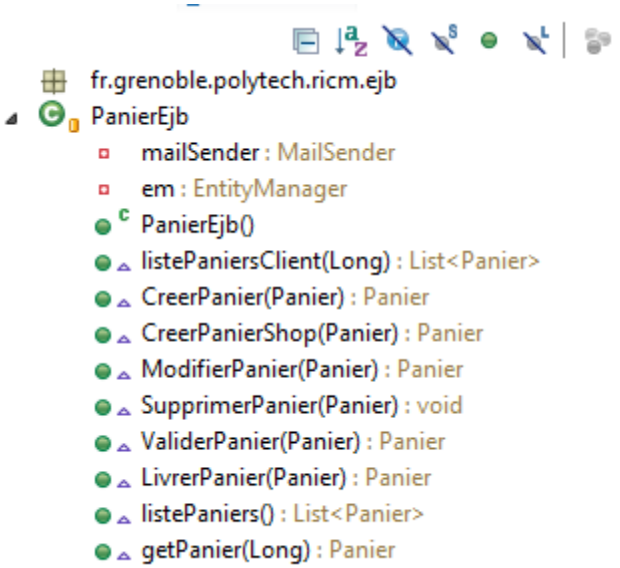
- ***CatalogueEJB*** : Cette EJB offre les fonctions permettant de manipuler le catalogue de l'application SushiParty. Pour cela elle permet de créer, consulter, modifier et supprimer les catégories d'articles et les articles vendus sur le site.

```
22
23  @SuppressWarnings("unchecked")
24  @Stateless(name = "PanierEjb", mappedName = "ejb/PanierEjb")
25  @TransactionManagement(TransactionManagementType.CONTAINER)
26  public class PanierEjb implements IPanierEjbRemote {
27
28
29      @Inject
30      private MailSender mailSender;
31
32      @PersistenceContext
33      private EntityManager em;
34
36      * Default constructor.
38      public PanierEjb() {
41
44      public List<Panier> listePaniersClient(Long idClient) throws Exception {
47
48      //@RolesAllowed({"Admin","Manager","Client"})
51      public Panier CreerPanier(Panier panier) throws Exception {
59
60      //@RolesAllowed({"Admin","Manager","Client"})
63      public Panier CreerPanierShop(Panier panier) throws Exception {
71
74      public Panier ModifierPanier(Panier panier) throws Exception {
78
81      public void SupprimerPanier(Panier panier) throws Exception {
85
88      public Panier ValiderPanier(Panier panier) throws Exception {
93
96      public Panier LivrerPanier(Panier panier) throws Exception {
101
104     public List<Panier> listePaniers() throws Exception {
107
```

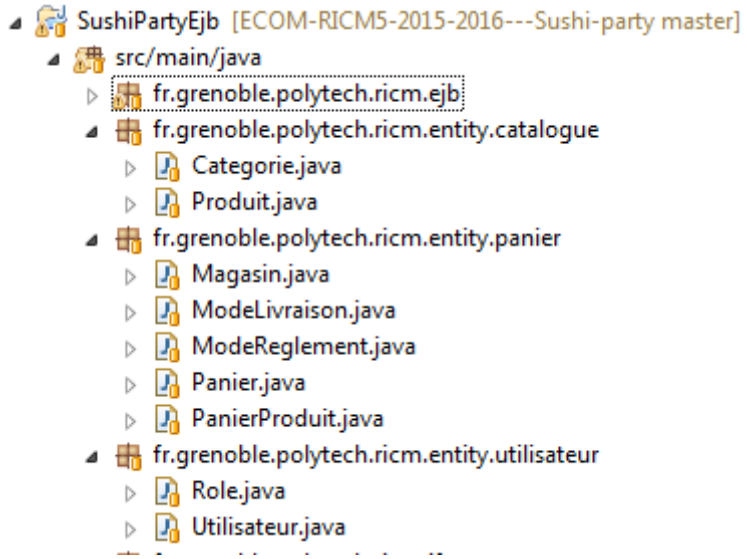
- ***UtilisateurEJB*** : Cette EJB offre les fonctions permettant de manipuler les comptes utilisateurs et les mots des passes des utilisateurs du système. Pour cela elle permet de créer, consulter, modifier et supprimer les comptes utilisateur sur le site, et de les affecter à un des trois rôles systèmes : Client, Manager et Administrateur. Cette EJB offre aussi les fonctions de gestion de mots de passes telles que 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 telles que 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 fallu implémenter est donnée 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 requêtes HTTP aux méthodes des classes REST. Dans cette partie nous avons implémenté une classe REST par rôles pour faciliter le contrôle d'accès.

```
@Path("common")
public class CommonRest {

    @Context
    private UriInfo uriInfo;

    private Properties env;
    private javax.naming.InitialContext ctx;

    public CommonRest() {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/categorie/")
    public List<Categorie> getCategories() {
        //logger.info("In getProjects");
        List<Categorie> list = null;
        try {
            ICatalogueEjbRemote catalogue = ( ICatalogueEjbRemote ) ctx.lookup( ICatalogueEjbRemote.JNDI_NAME );
            list = catalogue.listeCategories();
            //list = entityManager.createNamedQuery(Project.PROJECT_FIND_ALL_PROJECTS).getResultList();
            //logger.info("Nombre de projet : " + (list != null ? list.size() : "0"));
        } catch (Exception e) {
            //logger.error("Exception catchee " + ExceptionUtil.displayException(e));
        }

        return list;
    }

    public List<Produit> getProduits(@PathParam("id") Long id) {

    public List<Produit> getProduits() {

    public List<Magasin> getMagasins() {

}
```

Les composants AngularJS sont repartis dans les trois catégories suivantes : Vues, contrôleurs et Modèles. La liste complète des composants de cette partie sont décrits dans la figure suivante :

```
storeApp.config(['$routeProvider', function($routeProvider)
  $routeProvider.
      when('/store', {
        templateUrl: 'partials/store.htm',
        controller: storeController
      }).
      when('/cart', {
        templateUrl: 'partials/shoppingCart.htm',
        controller: storeController
      }).
      when('/livraison', {
        templateUrl: 'partials/livraison.htm',
        controller: storeController
      }).
      when('/validation', {
        templateUrl: 'partials/validation.htm'
      }).
      when('/adminPage', {
        templateUrl: 'partials/adminPage.htm',
        controller: storeController
      }).
      when('/modifierArticle', {
        templateUrl: 'partials/modifierArticle.htm'
      }).
      otherwise({
        redirectTo: '/store'
      });
}]);

storeApp.factory('Categorie', function($resource) {
    return $resource('resources/common/categorie/:id');
});

storeApp.factory('Produit', function($resource) {
    return $resource('resources/common/produit/:id');
})

storeApp.factory('Magasin', function($resource) {
    return $resource('resources/common/magasin/:id');
});

storeApp.factory('User', function($resource) {
    return $resource('resources/admin/utilisateur/:id');

//Contrôleur utilisé par toutes les vues de l'application
function storeController($scope, $routeParams, DataService, Categorie, Magasin, Produit, User, Panier) {

 // récupération du store et du cart avec le service
    $scope.store = DataService.store;
    $scope.cart = DataService.cart;

    $scope.loading = true;
    $scope.states = {};
    $scope.panier = {produits:[],modeLivraison:true,dateLivraison:new Date()};
    $scope.panier.id=420484336;
    $scope.panier.dateLivraison = new Date();
    $scope.panier.dateValidation = new Date();
    $scope.panier.dateCreation = new Date();
    $scope.panier.modeLivraison = true;
    $scope.panier.modeReglement = '';



    $scope.loading = true;
    var listeCategories = Categorie.query(function() {
        $scope.items = listeCategories;
        $scope.states.activeItem = $scope.items[0].designation;
        $scope.loading = false;
    });
```