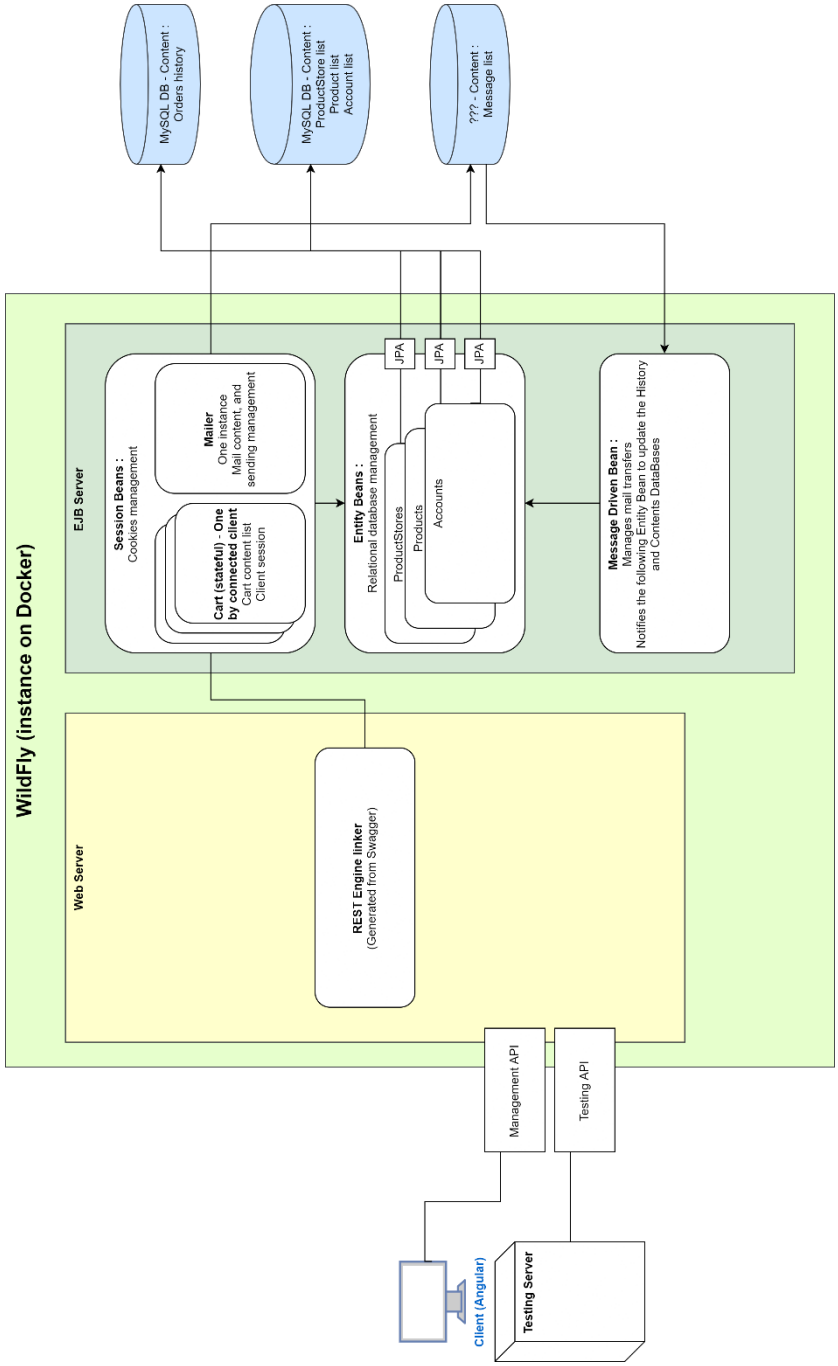
DOSSIER DE CONCEPTION SYSTEME

Application de location de véhicules : MesTransports

Le projet MesTransport requiert une architecture spécifique afin d'assurer le bon fonctionnement de toutes les applications affiliées ainsi que l'accès permanent aux services. Les données sensibles seront chiffrées, et les bases de données seront fréquemment répliquées afin d'assurer la pérennité des informations qu'elles contiennent en cas de pannes.



BASES DE DONNEES

Le service disposera de trois bases de données distinctes, permettant de stocker chacune une information précise. Ainsi tout le contenu du site ne sera pas présent sur une seule machine, ce qui permettra de délayer les différentes requêtes. De plus, les données seront fréquemment répliquées afin de conserver une version à jour en cas de panne.

BDD Contenu

Le contenu de cette base permet le bon fonctionnement du cœur de l'application. Elle contiendra au minimum les informations concernant les agences de locations, les magasins, la liste des véhicules, ainsi que les informations sur les comptes clients. Le tout sera réuni dans une base MySQL. L'architecture relationnelle à disposition permettra une séparation claire des données. De plus, l'équipe est formée à ce genre de pratique, ce qui assurera une meilleure qualité de produit.

Dans les détails, les magasins seront représentés par un identifiant unique, une adresse, un numéro de téléphone et un nom de gérant. Les comptes seront divisés en plusieurs types (utilisateur, gestionnaire, administrateur) définissants les droits d'accès, de visualisation et d'édition. Les véhicules pourront être renseignés, en précisant les informations relatives à leur prix, type, carburant, consommation, et tout autre **tag** nécessaire aux choix des clients.

Directement reliée aux Entity Beans, cette base de données ne pourra être modifiée que par ce biais. Chaque entrée sera donc un objet de ces classes.

BDD Transaction

Afin de permettre de fournir des données complémentaires nous devons sauvegarder l'ensemble des transactions effectuées sur notre plateforme. Cela permettra d'obtenir des informations plus globales, tel que "Quel trajet est le plus souvent effectué au mois d'avril ?", ou "Quelle entreprise a loué le plus de véhicules ce mois-ci ?". Ces données pourront être utilisées à des fins marketing ou encore de gestion pour permettre un meilleur contrôle de l'application. Aussi il est obligatoire de conserver pendant un certain temps les transactions bancaires effectuée, afin de procéder en cas de problème ou de réclamations.

D'un point de vue plus technique, cette base de données utilisera une architecture MySQL car cette dernière est efficace pour stocker des données, elle nous permettra donc de stocker l'ensemble des données avec un coût minimal en calcul lors de la recherche. MySQL est facile à mettre en place, simple d'utilisation et dispose d'une License totalement gratuite.

ENTERPRISE JAVABEANS SERVER

Depuis java EE 6 les entity beans ont été étiqueté comme obsolète. Des POJOs seront donc utilisés pour représenter nos Beans (cf https://developer.jboss.org/thread/273800)

SESSION BEANS

La Session Bean est la classe chargée de la gestion des sessions utilisateur (Cart Bean) et de la gestion des messages (Mailer Bean).

CART BEAN

Le Cart Bean représentera la session d'un client. Composé d'un token, et directement lié à l'Entity Bean relatif à cet utilisateur, il conservera une connexion avec le client de façon stateful pour assurer la complétion de toutes les requêtes du client.

Ce Bean contiendra entre autres une liste des produits sélectionnés par le client, qui seront donc "réservés" et inaccessible par les autres clients pour une durée pendant laquelle le client devra compléter sa commande, sans quoi l'assurance de sa commande ne pourra être certifiée.

Différentes fonctions seront implémentées afin d'assurer le bon fonctionnement de ce Bean, dont voici les principales :

- addProductToCart(Product p, Date start, Date end) : permet d'ajouter un nouveau produit au panier. Dans notre cas, la date de début et de fin est importante, car c'est une location.
- removeProductFromCart(Product p) : supprime un produit du panier
- validateCart() : "valide" le contenu du panier, et enclenche le début du processus de réservation
- proceedPayment() : réalise le paiement du panier. Si le paiement n'a pas fonctionné, il faut annuler tout le processus.

Le lien entre l'utilisateur et le Cart Bean sera assuré par l'API, via Servlet.

ENTITY BEANS

Les « Entity Beans » permettent de modifier, ajouter et « supprimer » des éléments dans les bases de données de transaction et d'entité. Ainsi, chaque table de la base de données sera représentée par une classe Java indépendante. Ces classes contiendront les attributs des objets ainsi que les méthodes nécessaires à la gestion de ceux-ci (getter et setter). Il y aura donc autant de classes que de tables : les magasins, les comptes clients, les agences de location, les véhicules, les commandes, les transactions.

Chaque classe aura une persistance et sera stockée en base de données. Ce processus sera effectué en utilisant la **Java Persistence API**. Cette API permet pour un objet Java de perdurer dans le temps en stockant son état en base de données. Ce système fonctionne avec des annotations et sera utilisé comme ORM (Objet Relationnel Mapping). Ainsi les objets java seront cohérents avec leur stockage en base de données.

JMS MOM

Tout d'abord un MOM ou "intergiciel à messages" est un composant utilisé au sein d'un système informatique pour permettre la communication par messages de plusieurs applications au sein de celui-ci. JMS est un API permettant la mise en place d'architectures MOM entre applications Java.

Une application voulant faire appel à JMS utilise un objet spécial appelé ConnectionFactory qui permet d'établir la connexion avec le Provider, le service mettant à disposition le module JMS. Ce ConnectionFactory permet ensuite de générer une Session qui prendra en charge la création du Message, et des interfaces source et destination. Les messages sont constitués d'une zone de données formatées librement précédées d'un en-tête parmi les différents possibles.

JMS est implantable en utilisant une implémentation compatible avec la structure considérée. Ainsi ayant fait le choix d'utiliser WildFly, JMS sera pris en charge par HornetQ au sein de WildFly. Pour se faire, il suffira d'éditer les fichier de configuration WildFly afin d'y faire figurer les informations relatives à HornetQ et de l'ajouter au sein de la structure.

INTERFACES ET UTILITAIRES DIVERS

HAProxy

HAProxy est un outil qui permet d'équilibrer les charges entre les serveurs d'une application web. Il s'agit d'un logiciel gratuit permettant de gérer un serveur Proxy pour les protocoles TCP et HTTP entre l'utilisateur et les serveurs de connexion. Il permet de gérer toutes les connexions entrantes, ainsi que de répartir les requêtes et connexions utilisateurs entre tous les serveurs actifs.
En cas de besoin, il sera donc possible d'ajouter ou supprimer un serveur à la liste du proxy. Cette scalabilité va assurer la pérennité de l'application en cas d'augmentation de la charge, ou d'économiser des ressources en période de faible demande.

API (Swagger)

La communication entre les interfaces client et le serveur se feront à partir d'une API. La technologie Swagger permettra de générer les URI à partir desquelles l'API accédera aux fonctions de l'application JEE. Les échanges de données se feront via des requêtes REST basées sur les méthodes GET et POST.

Une première application sera dédiée à l'application en elle-même; Il sera donc possible moyennant les droits d'accès qui seront accordés à l'utilisateur de consulter, éditer ou "supprimer" (masquer, en réalité) les données relatives à l'utilisateur. L'api permettra entre autres d'obtenir la liste des véhicules disponibles, le contenu des paniers clients ou la position des taxis dans une zones prédéfinie.

Une seconde API sera également disponible pour les tests. Le serveur Jenkins possédera donc des droits d'accès à cette API.

ANGULAR

AngularJS est un framework, qui utilise la sémantique Javascript afin de proposer un front-end a une application web. AngularJS possède de nombreux points forts :

- Utilise un couplage faible
- Est capable de supporter un grand nombre de connexions simultanées.
- Propose une évolution des données en pseudo temps réel (AJAX) sans recharger l'ensemble des vues ce qui permet une navigation fluide.

- Permet de créer des applications multi-plateforme

C'est pour toutes ces raisons que l'application client sera basée sur AngularJS. Cette application sera composée de quatre interfaces différentes :

- Interface administrateur
    - Permet de restaurer les données supprimées
    - Permet de gérer les droits d'accès des utilisateurs
- Interface locataire
    - Permet de louer des véhicules, consulter des paniers, consulter l'historique des locations
- Interface agence de location
    - Permet de consulter les statistiques d'une agence de location ainsi que gérer son parc de véhicules
- Interface agence de taxis
    - Permet de visualiser l'état des taxis, ajouter des véhicules, les supprimer ou encore d'affecter des courses