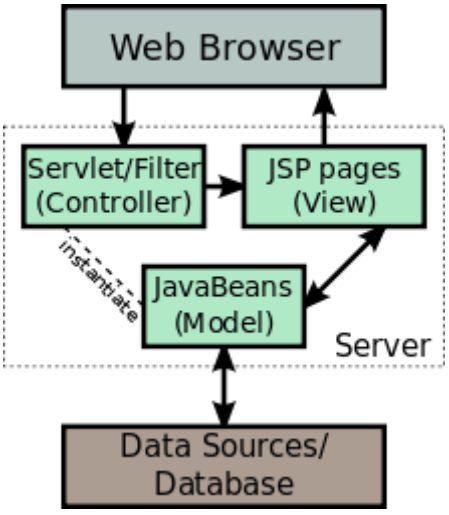
Document de conception système

Architecture

Nous allons bâtir notre application sur le modèle classique Modèle-Vue-Contrôleur car c'est celui qui offre le plus de flexibilité pour développer une grosse application comme la nôtre.



Nous allons développer le backend de l'application grâce à Java EE et le front-end utilisera le framework Bootstrap. L'application sera déployé au sein d'un serveur Wildfly. La persistance des données sera assuré grâce à une base de données MySQL.

Déploiement et cycle de vie de l'application

Le code de l'application sera hébergé par Github et l'intégration continue réalisé par Openshift. À chaque push, le code sera compilé puis testé par Maven. Si la modification réussi la compilation puis vérifie tous les tests Maven générera l'archive .war contenant notre application et sera déployé au sein d'un container docker wildfly hébergé sur Openshift.

Packages développés

Package entities

Les objets métiers que l'on va manipuler dans l'application :

- Utilisateur
- Réservation
- Spectacle

- Représentation
- Salle
- Panier

Package filters

Tous les filtres nécessaires au fonctionnement de l'application. Les filtres servent à restreindre les URLs visibles depuis l'extérieur et à faire des redirections. On s'en sert notamment que les utilisateurs se soient connectés avant d'accéder au fonction de paiement.

Package forms

Tous les formulaires de l'application sont gérés dans ce package.

- SignUpForm
- LoginForm
- SearchForm
- BuyingForm
- UserForm

Un formulaire prend en paramètre une requête avec donc des champs (GET ou POST) qui seront les données fournies par l'utilisateur. Chaque donnée saisis par l'utilisateur devra être vérifié et en cas d'invalidité, l'utilisateur devra être notifié. Les erreurs à détecter sont de natures diverses : détection de champ vide, d'adresse mail non valide, de mot de passe ne respectant notre politique de sécurité, etc.

Les formulaires ayant un comportement relativement similaire les uns des autres à la différence des données en entrée nous pourrons créer une couche d'abstraction en créant une classe Form.java dont hériteront les formulaires afin de généraliser et centraliser le code.

Package mails

Gestion des mails pour notifier l'utilisateur en fin de commande ou lors de la gestion de son compte (inscription, récupération de mot de passe).

Package bank

Contiendra la connection vers une API bancaire. Dans notre cas on transformera l'API bancaire par une fausse banque crée pour l'occasion. Nous aurons donc d'un côté le code pour se connecter à une API bancaire puis d'un autre côté une fausse banque pour simuler les achats. Dans cette fausse banque nous devrons quelques carte bancaires ainsi que quelques comptes afin d'avoir un comportement réalistes.

Package servlets

Les servlets sont la partie Contrôleur du modèle MVC. Ils permettent de faire le lien entre le modèle, les vues et ainsi d'organiser le cheminement du requète à travers les différentes couches de l'applications (formulaire, code métier, accès aux données, etc).

Globalement chaque servlet est associé à une seule et unique vue. Chaque servlet possède une méthode doGet pour fournir la page à l'utilisateur ainsi qu'une méthode doPost pour traiter les données envoyés par l'utilisateur.

Lorsque nous recevrons des données de l'utilisateur nous allons préalablement les examinés grâce à nos formulaires. Si les données sont incorrectes nous retournerons la page utilisée à l'utilisateur en lui indiquant le problème et en pré-remplissant les champs avec les données précédemment envoyé afin de garantir une expérience utilisateur correcte.

Le comportement des servlets étant souvent proche nous pourrons créer une classe Servlet dont hériteront les servlets afin de généraliser et centraliser les comportements.

- SignUpServlet
- LoginServlet
- SignOutServlet
- HomeServlet -> page où on fera les recherches de spectable
- ShowServlet -> page ou on pourra consulter en détail un spectacle
- BuyingServlet
- USerServlet

Si nous étendons notre projet avec la création d'une véritable API, le code de l'API serait stocké avec les autres sockets.

Package DAO

Contiendra les classes pour accéder aux données -> pattern DAO.

Pour chaque objet métier *objet* dont on veut assurer la persistance nous allons créer une interface *objetDAO* et une classe *objetDAOImpl* qui l'implémente pour notre solution de stockage (MySQL pour nous). Les méthodes de l'interface seront typiquement des méthodes CRUD avec plus ou moins de variante du Read.

Nous allons créer une DAOFactory afin d'initier et gérer les communications avec notre solution de stockage. Le rôle principal de DAOFactory sera de fournir des *Connection* vers la base de données aux classes *objetDAOImpl* afin qu'elles puissent réaliser leurs requêtes. Établir une *Connection* est coûteux, nous devrons donc gérer un pool de connexion afin d'améliorer les performances (utilisation de BoneCP). La DAOFactory fournira également des instances d' *objetDAOs.*

La classe DAOTools fournira un ensemble de méthodes utilitaires afin de simplifier la manipulation de requête. Typiquement nous aurons des méthodes de création de requête préparé simplifié ainsi que de fermetures des connections, des statements et des resultSets.

Package WEB-INF

Contiendra les vues et le front end de l'application (code html et JSP). Nous utiliserons les tags de la JSTL, ainsi que Bootstrap et JQuery afin de générer rapidement un front end responsive et design.

Package tools

Ensemble de code utilitaire pouvant être réutilisé dans l'application. Citons par exemple le code de cryptage de mot de passe.