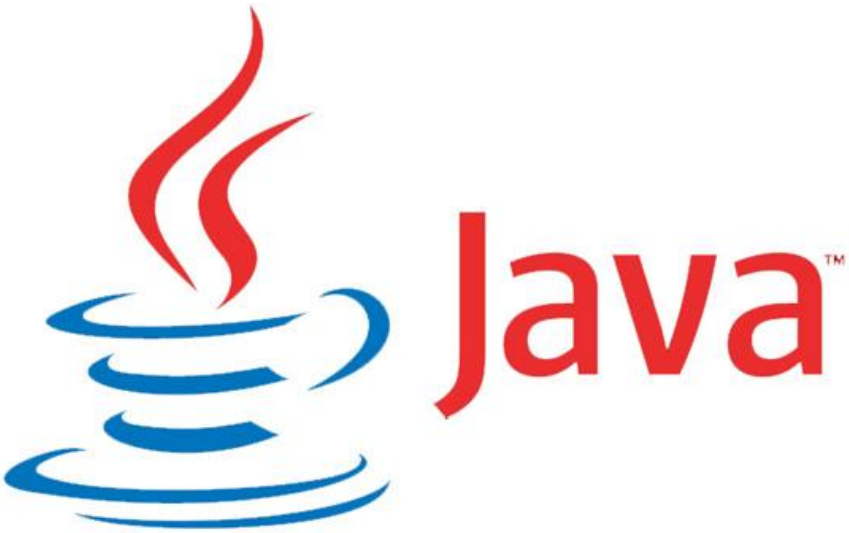
DOM-EZI



09/01/2017

RAPPORT

Rapport sur le projet DOM-EZI (domotique easy), projet d'algorithmique et programmation avancée

DOM-EZI

RAPPORT

SYSTEME D'UTILISATEURS

Le système d'utilisateurs est une partie fondamentale du projet. Toute personne possédant des objets connectés à ajouter au gestionnaire doivent préalablement créer un compte.

Mise en œuvre

Classiquement, un utilisateur devait pouvoir s'inscrire sur l'application en fournissant un nom, un login, une adresse e-mail et un mot de passe. Les trois composantes du système d'utilisateurs sont :

- La servlet ServletUser
- La facade UserFacade
- Les pages login.jsp et signin.jsp

Inscription

La vue utilise les fonctionnalités de HTML obligeant la personne souhaitant s'inscrire à entrer une adresse e-mail valide et à rentrer deux fois son mot de passe afin de confirmer la saisie.

Toutes les informations saisies dans le formulaire d'inscription sont transmises à la servlet ServletUser dédiée au contrôle du système d'utilisateurs.

Cette servlet vérifie alors les informations entrées. Il s'agit de vérifications attrayant à la sécurité (tentative d'injection de code malveillant), mais aussi à la validité des informations saisies (exemple :login entré ne doit bien évidemment pas encore exister dans la base de données).

Une fois ces vérifications effectuées, on enregistre l'utilisateur grâce à la façade UserFacade. Cette dernière créée un objet User et le rend persistant dans la base de données. Les informations sensibles telles que le mot de passe sont hachées pour plus de sécurité.

Connexion

Lors de l'identification d'un utilisateur sur le site, son login et son mot de passe lui sont demandé. Là encore, les informations sont envoyées à ServletUser qui vérifie que l'utilisateur est bien inscrit sur le site et que les informations d'identification sont correctes.

A partir du moment où l'utilisateur s'est identifié, l'objet User qui lui est associé est stocké dans une variable de type session. Ainsi chaque page .JSP peut savoir si l'utilisateur est identifié ou non en testant la nullité de la variable correspondante.

Pour aller plus loin

- Idéalement, nous aurions souhaité que la vérification du login lors de l'inscription se fasse dynamiquement (sans avoir à recharger la page)
- Le mot de passe est actuellement stocké dans l'objet User, or celui-ci est transmis à toutes les pages. Il aurait fallu créer une table Password liant un utilisateur à son mot de passe.
- Pour plus de facilité, la clé primaire de la table User est le login de l'utilisateur. En effet, il suffit alors de faire un find() pour obtenir l'objet User correspondant. Néanmoins, le temps de recherche serait optimisé si l'on utilisait un identifiant utilisateur entier.
- Vu nos connaissances en ce qui concerne la sécurité d'application web, il aurait mieux valu utiliser le système d'authentification intégré à JBoss.

SYSTEME OBJETS

Le système de gestion des objets est le but même du projet. Celui-ci s'intègre dans le projet notamment au travers des composantes suivantes :

- La façade ObjetFaçade.java
- La servlet ServletGestion.java
- La page JSP gestion.jsp

Le stockage des objets au niveau de l'utilisateur est modélisé par une simple Collection d'objets. Cette configuration nous a permis de répondre à certaines exigences requises lors de l'utilisation de beans. De plus, les zones conservent elles aussi une List d'objets afin de pouvoir facilement déterminer ceux présent dans celle-ci.

La création d'un objet (i.e. l'ajout dans la base de données) se fait en amont de la gestion de celui-ci, en effet, la page de gestion ne permet pas encore de pouvoir créer des objets "à la volée".

La gestion en elle-même des objets se fait sur la vue gestion.jsp, l'utilisateur y est invité à choisir/gérer une ou plusieurs zones. De là il peut alors au choix :

- Ajouter un objet existant à une zone existante
- Créer une zone
- (Utiliser un objet présent dans une zone)

Toutes ces fonctions sont réalisées par passage par la servlet ServletGestion. De même l'accès à la page gestion.jsp ne se fait que par un passage préalable par cette même Servlet afin de pouvoir passer l'objet du User dans la session. Si cela n'était pas fait il nous serait impossible de pouvoir récupérer la liste des objets disponibles pour l'utilisateur sans avoir recours à diverses API.

Amélioration

- Une réelle interface de gestion des objets, permettant aussi l'implémentation du système d'automatisation de tâches courantes, qui existe mais est pour le moment non utilisable.
- Un stockage moins redondant où l'utilisation d'API pour pourvoir éviter d'avoir une collection java pour chaque zone et chaque utilisateur contenant des objets, et non pas par exemple leur ID.

INTROSPECTION ET CREATION DU FICHIER XML

Motivations

L'objectif est de connaître les fonctions proposées par l'objet et ensuite de rassembler dans un fichier XML les valeurs des données correspondantes. À un fichier XML correspond donc un objet unique.
L'utilisation d'un fichier XML sert à :

- Communiquer à l'application web les fonctions que peut réaliser l'objet : ainsi, à chaque fois que l'utilisateur veut gérer l'objet, il suffit à l'application de lire le fichier XML pour savoir ce qu'elle doit afficher (opérations que l'utilisateur est autorisé à réaliser sur l'objet)
- Connaître les informations de l'objet sans avoir à rappeler la méthode.

Introspection

L'utilisateur ajoute le fichier .class de l'objet à son espace personnel. Ce fichier est créé par le constructeur et contient les fonctions permettant à l'utilisateur de commander l'objet et d'en recevoir des informations depuis l'application web. L'introspection est réalisée une seule fois, lors de l'ajout du fichier, car l'utilisateur ne peut pas modifier le fichier .Class. Durant cette phase, le fichier .class est analysé afin de trouver des méthodes que l'application sait traiter.

Création du fichier XML

Les méthodes reconnues servent à gérer un état (state), une valeur (value), une couleur (color) et une information (info). Lorsque certaines d'entre elles sont trouvées, les paramètres correspondants (state, value, etc.) sont entrés dans le fichier XML sous forme de balises, les fonctions d'accès à leurs valeurs sont appelées et les valeurs ainsi recueillies sont inscrites dans les balises.

GESTIONS DES ZONES

Les zones sont des lieux virtuels dans lesquels on peut ajouter des objets ou bien d'autres zones contenant elle-même des objets (analogie avec le système de répertoires dans lesquels on a des fichiers et d'autres répertoires). Les zones sont toutes sauvegardées en base de données et les zones sont présentes sous la forme d'Entity et gérés par la façade.

Lors de l'inscription et de la création d'un compte, un utilisateur possède par défaut une zone principale dans laquelle il peut ajouter zones ou objets. L'ajout de zone se fait par un formulaire HTML+CSS dans la page de gestion de zones.

Nous avons rencontré des difficultés avec JPA et les FetchTypes, c'est pourquoi il a été décidé d'utiliser le package « Hibernate » avec les annotations associées.

SECURITE

L'aspect sécurité WEB a aussi été abordée lors de la conception de notre application, la domotique inclut des aspects de la vie privée des utilisateurs c'est pourquoi nous avons décidé de contrôler la saisie d'information tant côté client que côté serveur. Tout ceci est bien sur transparent pour l'utilisateur qui navigue sur le site sans réel changement.

Côté client :

- Vérification des saisies grâce aux fonctions de HTML (on pourrait et devrait utiliser si l'on voulait aller plus loin Javascript)
- Utilisation du protocole HTTPS (aussi côté serveur)

Côté serveur :

- Vérification des saisies à l'aide d'expression régulières dans les Servlets
- Hash du mot de passe en utilisant le hash SHA-256
- Chiffrement de la base de données en utilisant le chiffrement symétrique AES avec une clef de chiffrement de 256 bits (un package est dédié à la crypto). Cette clef est générée grâce à un mot de passe utilisé en fonction des données de l'utilisateur enregistrées

PLANIFICATION DE TACHES

Il est possible pour un utilisateur de planifier des tâches, c'est-à-dire qu'il peut programmer une action à effectuer sur un objet à une date et une heure donnée.

Pour faire cela nous avons utilisé un EJB de type Entity « Task » afin de les sauvegarder dans la base de données. Une tâche possède un objet, le nom de la méthode à appliquer sur l'objet avec les arguments et une date d'application.

Un EJB Session de type Stateless « TaskManager » est utilisé avec à l'intérieur l'annotation *@Schedule* qui permet d'exécuter une procédure/fonction à un intervalle de temps régulier en JEE, l'EJB est instancié dès le déploiement du .WAR effectué.

Les tâches sont ordonnées par ordre croissante de date d'exécution dans une classe « Single » qui est l'utilisation du patron de conception « Singleton » qui permet d'utiliser une seule instance dans tout le site WEB. Dès qu'un utilisateur ajoute une tâche, une fonction *addTask(Task task)* est appelée pour ajouter la nouvelle tâche dans la liste de tâches ; elle est ajoutée en respectant l'ordre croissant de date d'exécution. Une fois la date d'exécution atteinte, on appelle la méthode *executeTasks()* qui permet d'exécuter les tâches qui doivent l'être.