Cahier de conception

Analyse fonctionnelle

Avant de commencer le développement de notre application nous avons procédé à une analyse fonctionnelle en se basant sur le cahier des charges. Ainsi un utilisateur qui se connecte peut :

- Enregistrer une clé avec son type et la lier à un canon
- Enregistrer une porte et la lier à une salle lui donner un canon
- Enregistrer un canon en précisant un fournisseur
- Enregistrer une salle en lui donnant un nom
- Voir une liste des clés avec leurs salles correspondantes
- Récupérer la clé correspondant à une salle donnée
- Créer un trousseau basé sur une collection de clés
- Prêter ce trousseau à un utilisateur
- Rendre une clé
- Déclarer une clé comme perdue
- Prolonger la date de rendu d'un emprunt
- Récupérer les clés possédés par une personne donnée

Le détail du déroulement de ces fonctionnalités est précisé dans le cahier d'utilisation.

Modèle de données

Lors de la phase de conception nous avons mené une étude du modèle de données fourni. Après étude nous avons apporté un certain de nombre de modifications. En effet, les différents VO sont liées les unes aux autres.

Ainsi une clé est liée à un canon, un canon à une porte, une porte à une salle.

Pour réaliser cela nous avons effectué des reports d'identifiants en ajoutant des données membres aux objets VO.

De plus afin d'assurer le maintien d'un historique des clés présentes dans les trousseaux nous avons créé une nouvelle structure de stockage mettant en relation l'identifiant de la clé et l'identifiant du trousseau.

Architecture logicielle

L'architecture logicielle de notre projet reprend celle proposée en début de projet. Nous avons cependant défini des rôles particuliers à chaque type de classes dans le but de développer un projet le plus maintenable et évolutif possible.

Modèles

Les **VO** sont la représentation logicielle des objets réels que l'application manipule. Elles sont le reflet des données stockées.

Les **DAO** sont des interfaces permettant d'accéder à la base de données de manière transparente pour le développeur. A l'heure actuelle l'application utilise les sessions du serveur web pour persister les données. Cependant, il est probable qu'en cas de mise en

production de l'application le passage vers l'utilisation d'une base de données réellement persistante soit demandée.
L'utilisation des DAO permet de faciliter cette transition, en effet les DAO exposent différentes méthodes dont le but est d'avoir le même comportement en fonction des implémentations.

Ainsi l'appel à la méthode *getKeys()* de la classe *implementationKeyDAO_Dummy* retourne un tableau d'instances de la classe *KeyVO* depuis la tableau de session. De la même manière, dans la cas de l'utilisation d'une base de données SQL, la fonction *getKeys()* de la classe *implementationKeyDAO_MYSQL* retournerai les même données en utilisant une requête SQL.
Ainsi, le reste du code de l'application est totalement isolé de la manière dont on stock les données. Pour passer d'une méthode de persistance à une autre il suffit alors de changer la classe utilisée pour récupérer les données.

Les **services** regroupent toute la logique métier de l'application. Le but d'un service est d'être à la fois décorrélé de la vue et du stockage des données. Ainsi on pourrait réutiliser les services dans une autre projet n'ayant pas les même vues et n'utilisant pas les même solutions de persistance des données.

Dans le cadre du cours de monsieur Soufflet nous avons mis en place un système de *Factory*. Le principe est de remplacer les différents appels à la méthode getInstance des DAO par l'appel d'une méthode d'une factory. Il existe une factory par méthode de persistance.
On récupère la factory correspondante via une classe provider qui prend en paramètre le type de moyen de persistance à utiliser. Ainsi, lors du passage d'un moyen de persistance à l'autre il suffit de remplacer la valeur de ce paramètre.

La logique métier est alors totalement isolée du reste de l'application.

Contrôleurs

Les contrôleurs font le lien entre modèle et vues. Ils sont instanciés par le routeur à l'appel d'une route HTTP précise.

Les contrôleurs associés directement à des vues prennent en paramètre le nom de la page est éventuellement des paramètres passés dans l'URL. Ces contrôleurs récupèrent les données via les DAO ou les services. On utilise directement les DAO quand on a pas besoin de croiser les données avec une autre table. Dans le cas contraire on utilise les services.
Une fois les données récupérées il est parfois nécessaire de retraiter ces données avant de les afficher. Les données prêtes à être affichées sont alors stockées dans des variables publiques du contrôleur afin d'être récupérable par la vue.

Dans le cas d'un contrôleur utilisé uniquement pour le traitement de données il utilise en grande partie les méthodes des services. Le code du contrôleur se résumant alors à gérer les contrôles sur les données fournies par l'utilisateur, notamment via les formulaires. De cette manière la partie métier du code reste bien isolée de la partie liée à l'affichage.

Dans le cas où la vue venait à radicalement changer le code des services resterait valable et il suffirait de redévelopper le contrôleur.

Vues

Le rôle de la vue est de générer du code HTML avec les données préparées par le contrôleur. Nous avons fait le choix de ne pas utiliser de moteur de template autre que celui fourni par PHP afin de nous concentrer sur le développement de la partie métier.

Nous utilisons donc PHP pour récupérer les données stockées dans le contrôleur et le afficher.

Routeur

Afin de coordonner ces différentes parties nous avons fait le choix d'utiliser un routeur. Le principe du routeur est de récupérer toutes les requêtes sur une seule page PHP qui se charge d'appeler les différents contrôleurs.

Chaque adresse est appelée route. On distingue deux types de routes en fonction du mode de passage des paramètres : GET ou POST.

Dans le cadre de ce projet nous avons choisi d'utiliser un framework nommé limonade.php dont la principale fonctionnalité est d'offrir un routeur simple d'utilisation.

La déclaration d'une route se fait via l'appel de la fonction dispatch qui prend en paramètres le nom de la route et le nom de la fonction à appeler.

```
dispatch('/home', 'home');
function home(){
   // Appel du contrôleur puis de la vue
}
```

Par défaut la création d'une route se fait en mode GET, pour créer une route accessible via une requête POST on utilise la fonction dispatch_post.

De plus, il est possible de récupérer les données passées en paramètre d'une requête GET. On place pour cela le nom du paramètre dans la déclaration de la route, précédée du caractère ":". La valeur de ce paramètre est alors récupérable via une variable du même nom en paramètre de la fonction associée.

```
dispatch('/returnKey/:id', returnKey);
function returnKey($id){

}
```

Ainsi la fonction returnKey récupèrera la variable id avec la valeur 5 à l'appel de la route /returnKey/5.