DOCUMENTATION TECHNIQUE

Projet « GSB Médecin »

TABLE DES MATIÈRES

PRÉSENTATION DU PROJET

1.1. Contexte du projet

Le laboratoire Galaxy Swiss Bourdin (GSB) est amené dans ses différentes activités à contacter les médecins installés, par exemple les visiteurs médicaux du laboratoire se déplacent afin de présenter les nouveaux produits pharmaceutiques.
Une base de données de médecins est utilisée dans ces différents processus. Cette base de données évolue fréquemment, ceci à cause de départ à la retraite de médecins ou d'installation de nouveaux praticiens. Les applications doivent intégrer ces évolutions des données.

Il a été décidé de confier à une société de services informatiques la responsabilité de développer deux applications donnant accès aux informations de la base de données.

1.2. L'entreprise GSB

Le laboratoire Galaxy Swiss Bourdin (GSB) est issu de la fusion entre le géant américain Galaxy (spécialisé dans le secteur des maladies virales dont le SIDA et les hépatites) et le conglomérat européen Swiss Bourdin (travaillant sur des médicaments plus conventionnels), lui même déjà union de trois petits laboratoires .En 2009, les deux géants pharmaceutiques ont uni leurs forces pour créer un leader de ce secteur industriel. L'entité Galaxy Swiss Bourdin Europe a établi son siège administratif à Paris.Le laboratoire Galaxy Swiss Bourdin (GSB) est issu de la fusion entre le géant américain Galaxy (spécialisé dans le secteur des maladies virales dont le SIDA et les hépatites) et le conglomérat européen Swiss Bourdin (travaillant sur des médicaments plus conventionnels), lui même déjà union de trois petits laboratoires .En 2009, les deux géants pharmaceutiques ont uni leurs forces pour créer un leader de ce secteur industriel. L'entité Galaxy Swiss Bourdin Europe a établi son siège administratif à Paris.

1.3. L'activité à réaliser

Un utilisateur *(connecté ou non)* doit pouvoir voir tous les médecins GSB *(avec leurs informations type : nom, prénom, adresse, numéro de téléphone, etc.)* ainsi que les pays et départements qui sont disponibles et créés dans la base de données.

L'activité à réaliser doit permettre de réaliser des actions **CRUD (C**reate **R**ead **U**pdate **D**elete**)** à l'utilisateur qui aurait les accès administrateurs, ce qui inclue donc également un système de connexion. C'est-à-dire que depuis l'application, ils auront comme possibilité d'ajouter, modifier ou bien supprimer un médecin, un pays ou bien un département.

Le développement complet de l'application doit être réalisé en utilisant une architecture **MVC (M**odèle-**V**ue-**C**ontrôleur**)** ainsi qu'un langage de programmation orienté objet ***(POO)***, dans le cas présent, le langage utilisé sera du **Java**.

PRÉSENTATION DU PROJET

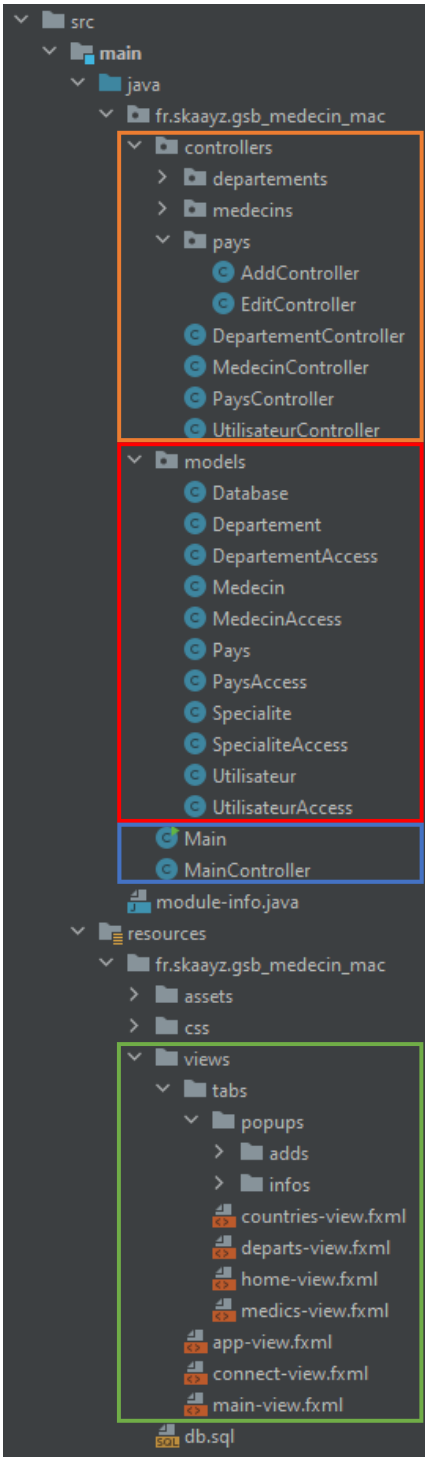
1.4. Mode restreint/complet

L'application possède un **mode restreint**, mais également un **mode complet**. Le mode complet n'est accessible seulement, et seulement si l'utilisateur du logiciel a pu s'authentifier en tant que privilège administrateur. Pour cela certaines sécurités ont été mise en place, avec un chiffrage notamment du nom d'utilisateur et du mot de passe en format **SHA-512**. Une clé de déchiffrage de 512 caractères doit être utilisée pour pouvoir déchiffrer le mot de passe, à noter qu'une clé est pour le nom d'utilisateur et une autre a été créée pour le mot de passe afin de renforcer au maximum la sécurité. Une fois connecté et le **mode complet** activé, l'utilisateur pourra alors modifier, ajouter ou supprimer ce qu'il souhaite, que cela soit un Pays, un Département ou bien un Médecin. C'est ce qui permet donc de réaliser les actions **CRUD** demandé dans le projet.

FONCTIONNEMENT DE L'APPLICATION

2.1. Fonctionnement technique

2.1.1. Structure de l'application



La structure de l'application est telle qu'elle répertorie les différents contrôleurs *(dans le dossier **controllers**)*, les différents modèles *(dans le dossier **models**)* ainsi que les différentes vues *(dans le dossier **views** situé en bas de la structure, dans le dossier **resources**)*.

Chaque fichier travaille sur un ou plusieurs éléments liés à eux.
Par exemple, le **MedecinController** *(situé dans le dossier **controllers**)*, le **Medecin** ainsi que le **MedecinAccess** *(situés dans le dossier **models**)* fonctionnent ensemble pour tout ce qui est lié de près ou de loin aux différents médecins.
Dans ce même dossier, il est possible voir d'autre dossiers nommés **departements**, **medecins** et **pays**. Ils ont été créés afin de pouvoir stocker les fichiers **AddController** et **EditController** qui seront alors utilisés pour permettre de modifier ou ajouter du contenu dans la table y étant référée. *(Exemple : le **AddController** du dossier **pays** va permettre de travailler sur l'ajout d'autres pays).*

Toutes les différentes méthodes liées au sujet du fichier seront alors inscrites dedans et seulement dedans.

Le fichier **Database** sera la classe principale permettant la connexion à la base de données créée pour le projet, ainsi certaines méthodes à l'intérieur telle que « **execute()** » permettent alors d'effectuer des requêtes directement dans la BDD.

Dans la partie « **views** », quelques dossiers/fichiers sont alors visibles également. Premièrement, le fichier **main-view.fxml** est lié à la page principale qui est affichée lorsque le logiciel se lance. Ainsi, cela est pareil pour le **connect-view.fxml** qui affiche la page de connexion et **app-view.fxml** qui lui se charge d'afficher l'interface principale du logiciel.

Enfin, pour les fichiers **Main** et **MainController** ceux-ci sont nécessaires à l'exécution du projet, car c'est comme cela qu'il est configuré. Il va aller chercher un fichier **Main** et **MainController** qui eux vont permettre de gérer la partie principale de l'application. C'est-à-dire que par exemple, le **Main** et **MainController** vont contrôler le **main-view.fxml** et renverront vers d'autres vues qui utiliseront alors les contrôleurs prévus à cet effet.

FONCTIONNEMENT DE L'APPLICATION

2.1.2. L'architecture d'application « MVC »

Pour la création de ce logiciel, une architecture dites « MVC » *(**Modèle-V**ue-**C**ontrôleur)* était demandée.

Dans cette architecture, le « **Modèle** » lui, va gérer toute la partie base de données de l'application. Son rôle principal est de récupérer les informations « brutes » venant de la BDD *(base de données)*, de les organiser et de les renvoyer directement au **contrôleur** qui les aura au préalable demandé. Donc on y retrouve entre autres les différente requêtes SQL, qui peuvent permettre de récupérer des informations comme cité plus haut, ou bien alors d'en envoyer via l'exécution de certaines requêtes comme des « **INSERT INTO** » ou bien « **UPDATE** ». L'entièreté des fichiers du dossier **models** sont développé en **Java**.

La « **Vue »** sera l'affichage du logiciel et des différentes pages. Elle ne fera aucun calcul et elle se contentera juste de faire afficher des informations qui pourront être modifiée depuis le contrôleur. *(Exemple, un texte placeholder pour le nom de famille d'un médecin qui pourra alors être modifié directement depuis le contrôleur des médecins)*. L'entièreté des fichiers du dossier **views** sont développé en **FXML**. Cependant, un logiciel *(Scene Builder)* existe afin de faire seulement du **drag-n-drop**. Cela simplifie donc la mise en place d'un design pour les pages.

Le « **Contrôleur** » va être la passerelle entre la **vue** et le **modèle**, c'est par ici que la vue pourra alors être modifiée *(comme par exemple, comme cité plus haut, modifier un texte pour qu'il affiche la bonne valeur demandée)*, le contrôleur va alors traiter les différentes demandes et les vérifier pour ensuite aller questionner le modèle et récupérer les informations initialement demandées. C'est un peu le « chef d'orchestrte » de cette architecture. Il gère la partie logique du code et les décisions. L'entièreté des fichiers du dossier **controllers** sont developpé en **Java**.

FONCTIONNEMENT DE L'APPLICATION

2.1.3. Le pattern des classes et leurs fonctionnements

Partie contrôleur :

Durant cette partie, je vais décrire le fonctionnement « **basique** » d'une classe contrôleur dans mon logiciel, elle aura pour fonctionnalité principale d'exécuter des méthodes, avec notamment des « **getter** » mais aussi des « **setter** » qi prendront la forme de requêtes **SQL**, envoyée directement à la couche **modèle** de mon architecture **MVC**. Cette classe est nommée « **MedecinController** », j'ai décidé de la prendre en exemple car c'est celle qui a le plus de méthodes concrète et différente permettant de mieux comprendre le fonctionnement global de mon code et de ma manière de concevoir l'application. Enfin, nous verrons aussi très rapidement le « **UtilisateurController** » pour que puissiez savoir comment j'ai organisé mon système de connexion à l'application.

Fichier « **MedecinController** » :

```
package fr.skaayz.gsb_medecin_mac.controllers;

import fr.skaayz.gsb_medecin_mac.MainController;
import fr.skaayz.gsb_medecin_mac.controllers.medecins.EditController;
import fr.skaayz.gsb_medecin_mac.models.*;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Cursor;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;
import javafx.util.Callback;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
```