Yvert Faustine
BTS SIO 2B

DOCUMENTATION TECHNIQUE
Galaxy Swiss Bourdin

2021 - 2022

SOMMAIRE

CADRE DU PROJET

1) Contexte de l'entreprise

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 siège social de la multinationale est situé à Philadelphie, Pennsylvanie, aux États-Unis.

Le projet

Le but était de faire une application dans un langage orienté objet. L'architecture applicative sera en MVC (Modèle Vue Contrôleur). L'application utilisera la base de données PostgreSQL fournie. Elle est garnie et mise à jour par une entreprise spécialisée qui nous vend ces informations. Un système SCRUD (Search, Create, Update, Delete) a été mis en place.

2) Enjeux et objectifs

La connexion

L'utilisateur possède un identifiant et un mot de passe afin de pouvoir accéder à l'entièreté de l'application.

Les pays

Une liste de pays doit être visible sur l'application. L'utilisateur doit avoir la possibilité de pouvoir rechercher un pays via son nom. L'utilisateur peut aussi voir la fiche complète du pays. L'ajout, la modification, et la suppression d'un pays choisi est possible.

Les départements

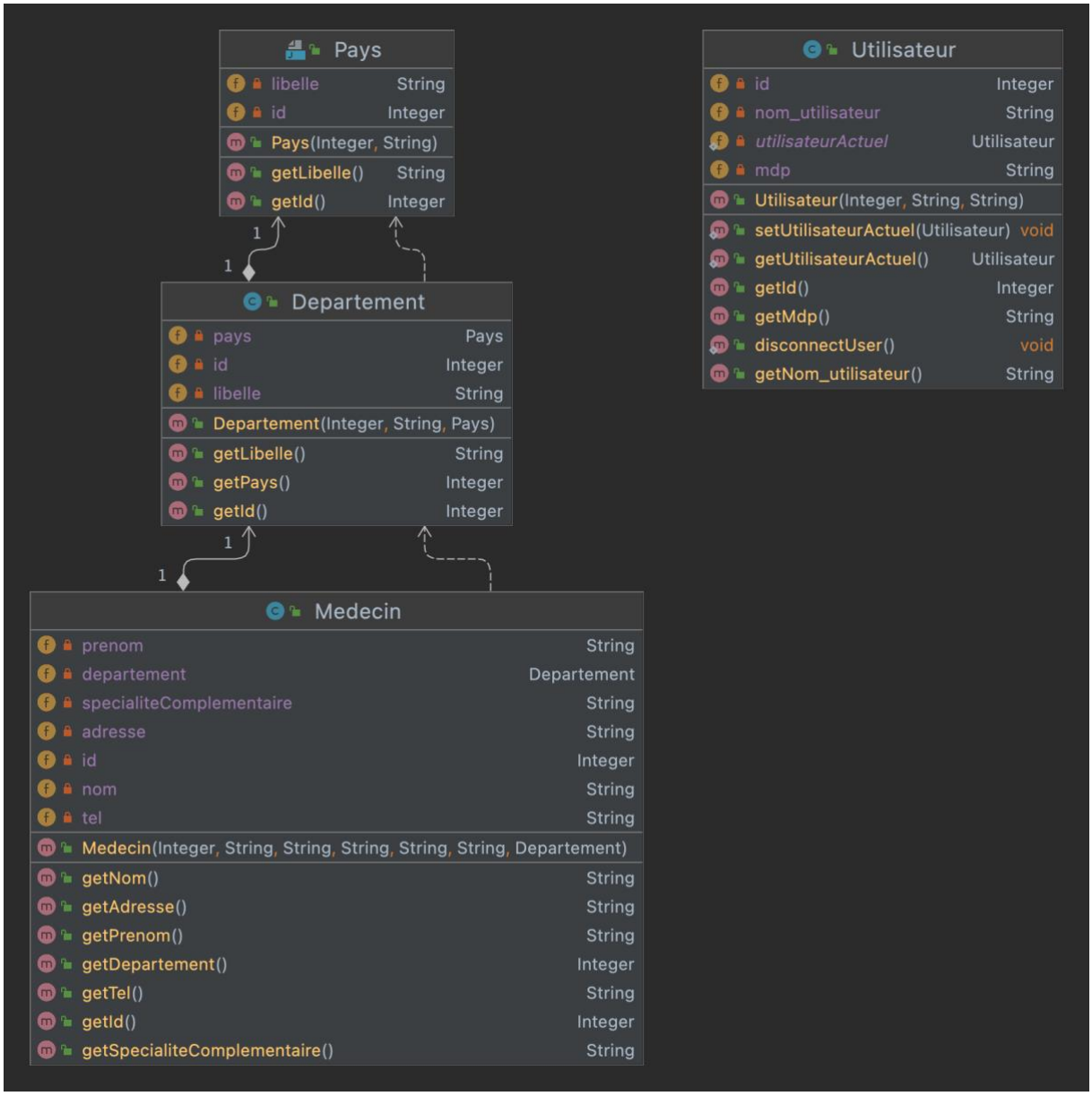
Une liste de département doit être visible sur l'application. L'utilisateur doit avoir la possibilité de pouvoir rechercher un département via son nom de département ou bien le pays qu'il lui est attribué. L'utilisateur peut aussi voir la fiche complète du département. L'ajout, la modification, et la suppression d'un département choisi est possible.

Les médecins

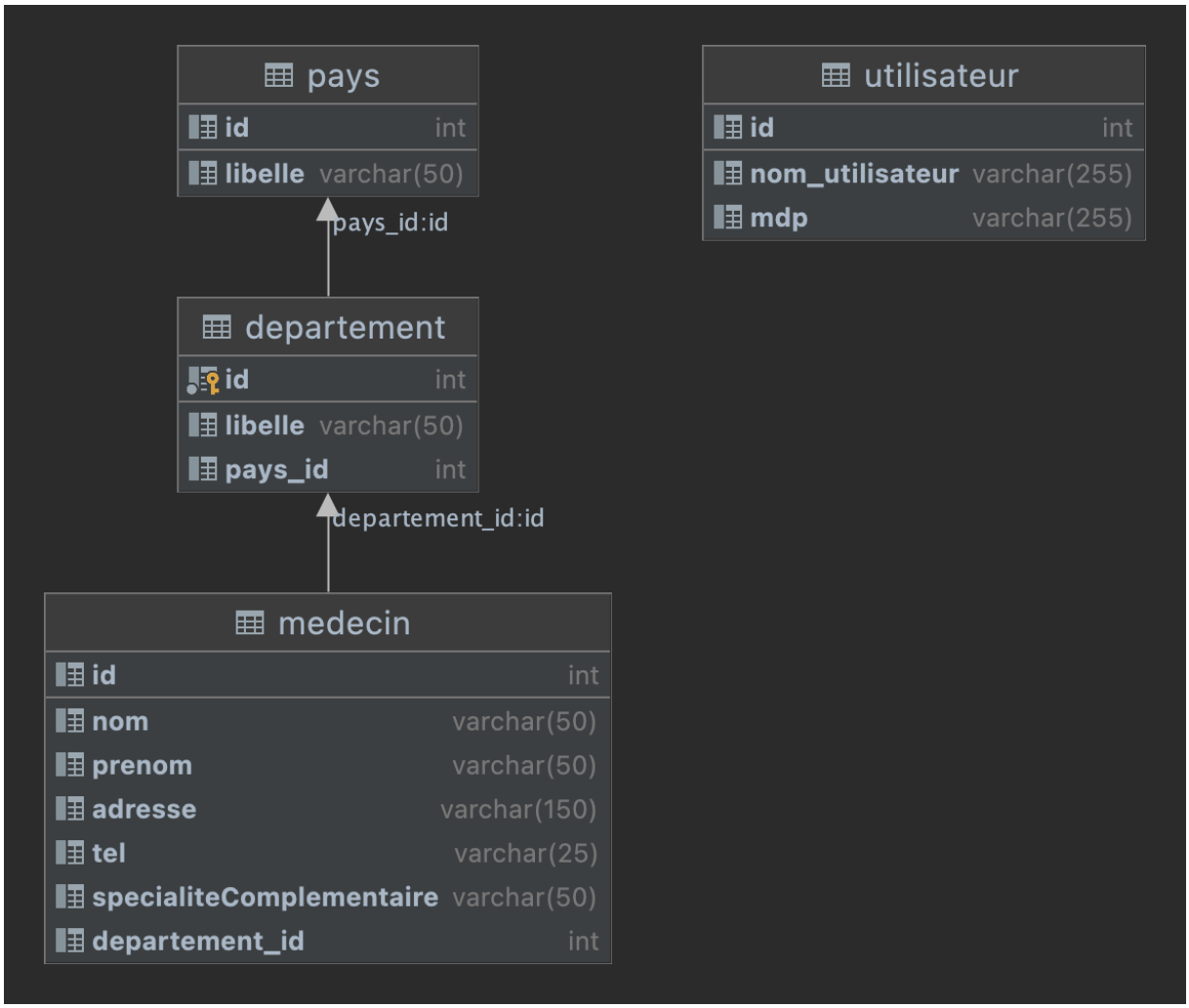
Une liste de médecins doit être visible sur l'application. L'utilisateur doit avoir la possibilité de pouvoir rechercher un médecin via son nom, son prénom, ou bien l'id du département qu'il lui est attribué. L'utilisateur peut aussi voir la fiche complète du médecin. L'ajout, la modification, et la suppression d'un médecin choisi est possible.

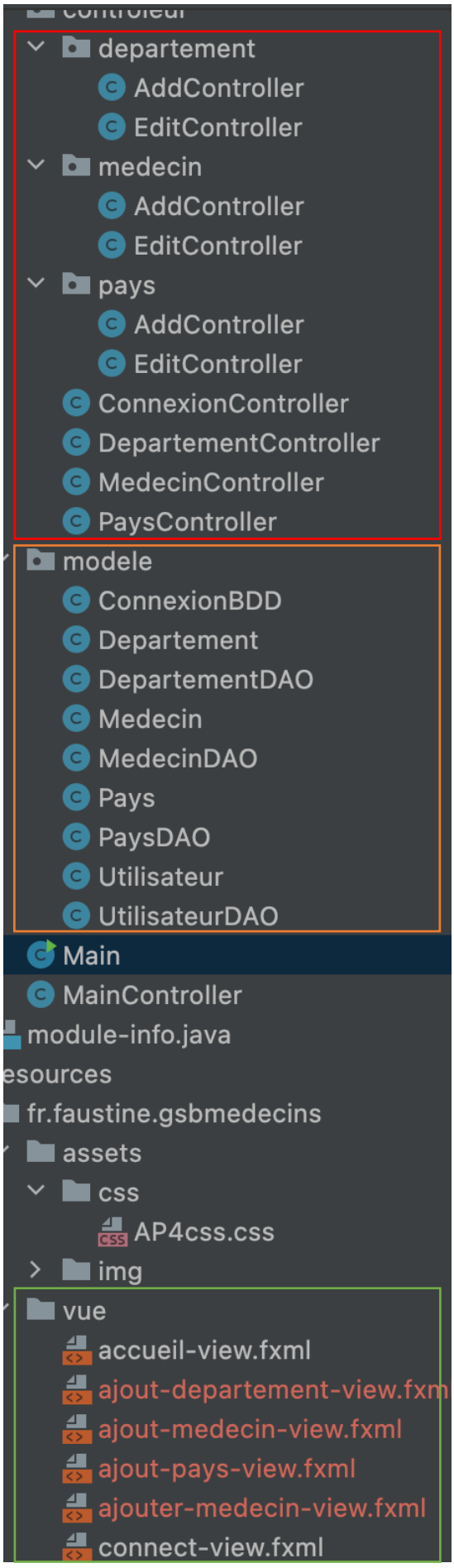
3. Modélisation

Unified Modeling Language (UML)



Modèle Logique Relationnel (MLR)





LE PROJET

1) L'architecture

Le motif d'architecture est en MVC (Modèle, Vue, Contrôleur).

Le pattern MVC permet de bien organiser son code source. Il va vous aider à savoir quels fichiers créer, mais surtout à définir leur rôle. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- **Modèle** : cette partie gère les données du site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.
- **Vue** : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.
- **Contrôleur** : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

➔ Connexion à la base de données

1ère étape : la connexion à la base de données est très importante, afin que le modèle puisse récupérer toutes les informations de celle-ci.

```
package fr.faustine.gsbmedecins.modele;
import java.sql.*;

public class ConnexionBDD {
    // Variables
    private static Connection conn_db = null;

    private static final String host_db = "127.0.0.1";
    private static final String name_db = "gsb_medecins";
    private static final String user_db = "root";
    private static final String pass_db = "";

    private static final String link_db = "jdbc:mysql://" + host_db + "/" +
name_db;

    // Functions
    private static Connection getConnection() {
        try {
            conn_db = DriverManager.getConnection(link_db, user_db,
pass_db);
        } catch(Exception e) {
            e.printStackTrace();
        }

        return conn_db;
    }

    protected static ResultSet query(String request) throws SQLException {
        Connection connect_db = getConnection();
        ResultSet queryOutput;

        Statement statement = connect_db.createStatement();
        queryOutput = statement.executeQuery(request);

        return queryOutput;
    }

    protected static void execute(String request) throws SQLException {
        Connection connect_db = getConnection();
        Statement statement = connect_db.createStatement();

        statement.execute(request);
    }
}
```

Exemple pour les Médecins

➔ Contrôleur

(MedecinController)

```
package fr.faustine.gsbmedecins.controleur;

import fr.faustine.gsbmedecins.MainController;
import fr.faustine.gsbmedecins.controleur.medecin.EditController;
import fr.faustine.gsbmedecins.modele.Medecin;
import fr.faustine.gsbmedecins.modele.MedecinDAO;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
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;
```

Dans le contrôleur il est important d'importer les « package » car dans le code de base, tous ces packages ne sont pas importés automatiquement.
Par exemple, « import javafx.fxml.FXML; » va permettre d'importer toutes les méthodes afin de pouvoir utiliser les variables et fonctions relative au FXML.
Pour que le code interprète du FXML, il faut impérativement la déclarer avec un « @FXML » juste au-dessus.

Exemple de variables FXML dans le fichier « MedecinController » :

```
public class MedecinController implements Initializable {
    // FXML Variables
    @FXML
    public TableView<Medecin> medecins_table;

    @FXML
    public TableColumn<Medecin, Integer> medecin_id;

    @FXML
    public TableColumn<Medecin, String> medecin_lastname,
medecin_firstname, medecin_action;

    @FXML
    public TextField medecins_searchbar;
```

➔ Vue

Maintenant, prenons l'exemple pour supprimer un médecin.

La première partie de passe dans le fichier « voirplus-medecin-view.fxml » situé dans la vue. L'ajout d'un bouton est nécessaire afin de pouvoir lui attribuer une action afin de pouvoir lui faire effectuer une fonction précise dans un autre fichier.
Ici, l'action a été nommée 'supprimerButtonClicked'

```
<Button fx:id="supprimer_button" layoutX="87.0" layoutY="330.0"
mnemonicParsing="false" onAction="#supprimerButtonClicked"
prefHeight="20.0" prefWidth="70.0" style="-fx-background-color: #e06666; -
fx-background-radius: 5;" text="Supprimer" textFill="WHITE">
   <font>
      <Font size="10.0" />
   </font>
   <cursor>
      <Cursor fx:constant="HAND" />
   </cursor>
</Button>
```

Après avoir donné un nom à mon action, tout le reste se passe dans le « EditController » du médecin.

➔ Contrôleur

```
// Exécution du bouton supprimer
public void supprimerButtonClicked() {
    Alert alert = new Alert(Alert.AlertType.NONE, "Êtes-vous sûr de
supprimer le médecin " + getMedecinActuel().getNom() + " " +
getMedecinActuel().getPrenom(), ButtonType.YES, ButtonType.NO);
    alert.showAndWait();

    if (alert.getResult() == ButtonType.YES) {
        MedecinDAO.deleteMedecinByID(getMedecinActuel().getId());
        retourButtonClicked();
    }
}
```

Ici, la fonction envoie une nouvelle fenêtre de type alerte avec une phrase inscriste : « Êtes-vous sur de supprimer le médecin … » (Ici le get.nom ainsi que le get.prenom auront été récupérés grâce à la base de donnée : table médecin. Le nom et prénom qui s'affichera sera celui dont l'utilisateur aura sélectionné). Si l'utilisateur clique sur la réponse « non » alors aucune suppression ne sera effectuée, mais l'alerte sera toujours présente tant que l'utilisateur n'aura pas dit oui, ou alors quitté l'alerte. En revanche, s'il clique sur « oui » alors le médecin selectionné sera supprimé (grâce au modèle avec la requête attribuée à 'deleteMedecinByID'. Ainsi l'utilisateur pourra retourner à la liste des médecins grâce au 'retourButtonClicked'.

➔ Modèle

Voici la classe médecin avec ses variables, son constructeur et ses fonctions.

```
package fr.faustine.gsbmedecins.modele;

public class Medecin {
    //Variables
    private Integer id;
    private String nom;
    private String prenom;
    private String adresse;
    private String tel;
    private String specialiteComplementaire;
    private Integer departement_id;

    // Constructeur
    public Medecin(Integer id, String nom, String prenom, String adresse,
String tel, String specialiteComplementaire, Integer departement_id) {
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.adresse = adresse;
        this.tel = tel;
        this.specialiteComplementaire = specialiteComplementaire;
        this.departement_id = departement_id;
    }

    // Getter
    public Integer getId() {
        return id;
    }
    public String getNom() {
        return nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public String getAdresse() {
        return adresse;
    }
    public String getTel() {
        return tel;
    }
    public String getSpecialiteComplementaire() {
        return specialiteComplementaire;
    }
    public Integer getDepartement_id() {
        return departement_id;
    }
}
```

➔ Modèle DAO

Comme vu précédemment, voici la requête attribuée à ‘deleteMedecinByID’ avec un simple delete par l’id du médecin. Pas besoin de mettre le nom, prénom, adresse, tel,… Car chaque médecin à un id différent donc en supprimant juste avec l’id, cela supprimera le médecin avec toutes ses informations.

```
// suppression d'un médecin par son ID
public static void deleteMedecinByID(int id) {
    try {
        ConnexionBDD.execute("DELETE FROM medecin WHERE id = " + id + ";");
    } catch(SQLException e) {
        e.printStackTrace();
    }
}
```