



DOSSIER DE CONCEPTION

PROJET NFA019

Application de Gestion de rendez-vous médicaux
Dossier Médical Partagé (DMP)



I. Table des matières

I. Table des matières	2
II. Introduction.....	3
1. Objectif du document.....	3
2. Description du projet.....	3
III. Définition des besoins	3
1. Besoins fonctionnels.....	3
2. Besoins non-fonctionnels	4
IV. Contraintes de conception	5
V. Technologies utilisées	6
1. Choix du langage	6
2. Solution logicielle	6
3. Choix du support enregistrement de données.....	6
VI. Liste des fonctionnalités.....	7
1. Présentation	7
2. Fonctionnalité de l'Administrateur	7
3. Fonctionnalité du Patient	8
4. Fonctionnalité du Médecin	9
VII. Maquette des écrans.....	10
VIII. Architecture.....	15
1. Architecture de la Base de données.....	16
2. Architecture du code Java	17
3. Diagramme de classe des Beans	18
4. Utilisation du pattern DAO.....	19
5. Séparation des accès avec l'objet Filtre	22
6. Objets Traitement	23
7. Structuration des Servlets	24
8. Librairies externes utilisées	25
9. Architecture des pages HTML/CSS	25
IX. Procédure d'installation	26
X. Documentation ressources externes.....	31
XI. Conclusion	32



II. Introduction

1. Objectif du document

Le but de ce document est de **présenter l'architecture d'une application développée** dans le cadre du cours NFA019 de la formation Technicien Développeur au CNAM.

Il s'agit donc à la fois d'un **dossier de conception** et d'un **compte rendu du projet**.

Nous commencerons donc par expliquer l'objectif et les besoins attendus avant de détailler le processus de création du projet.

2. Description du projet

Une application offrant un **service de gestion de rendez-vous médicaux** pour les patients et les médecins, ainsi qu'un accès administrateur pour la gestion des utilisateurs.

A la manière du Dossier Médical Partagé, l'application permet aussi aux médecins d'y inscrire et d'y modifier les diagnostics et prescriptions liées aux consultations.

III. Définition des besoins

1. Besoins fonctionnels

L'application sera utilisée principalement par **trois types d'utilisateur**, et chaque utilisateur doit être en mesure d'utiliser des **services** particuliers qui sont listés dans ce tableau :

Utilisateur	Services
Admin	<ul style="list-style-type: none"> - S'identifier - Gérer les médecins - Gérer les patients
Patient	<ul style="list-style-type: none"> - S'identifier - Rechercher et consulter la liste des médecins - Contacter un médecin (demander un rendez-vous)
Médecin	<ul style="list-style-type: none"> - Afficher son dossier - S'identifier - Rechercher ou contacter un autre médecin - Afficher les consultations - Saisir le résultat d'une consultation - Saisir une prescription



2. Besoins non-fonctionnels

Etant donné que ce projet a pour objectif principal d'exercer et **de mettre en pratique les notions acquises** dans les cours d'informatique, le programme doit respecter les critères de réalisation suivants :

[NFA031 : Java : notions de base] [NFA032 : Java : programmation objet]

- **Les classes devront comporter des constructeurs, accesseurs et modificateurs ainsi que des redéfinitions de méthode « ToString() »**
- **Les attributs ne devront être accessibles que par leurs accesseurs**
- **Le code doit comporter la gestion des erreurs/exceptions**
 - ❖ Vérifier que les champs d'un formulaire ne sont pas vides
 - ❖ Vérifier que certains champs possèdent certains caractères (ex : email)
 - ❖ Vérifier si un champ est numérique
 - ❖ Vérifier si une date est valide
- **Le code doit comporter des structures de données (tableau, ArrayList...)**

[NFA008 : Base de données] [NFA011 : applications avec bases de données]

- **Pour la lecture et l'écriture des données, un choix est offert entre plusieurs solutions :**
 - ❖ Vers un fichier *.txt
 - ❖ Vers un fichier *.csv
 - ❖ Vers un fichier *.XML
 - ❖ Base de données

[NFA016 : Développement web côté client] [NFA084 : Graphisme et Web]

- **Ergonomie et convivialité des applications**
 - ❖ Disposition des composants
 - ❖ Navigation
 - ❖ Comportement
 - ❖ Traitement des erreurs
- **Aspects graphiques**
 - ❖ Esthétiques
 - ❖ Choix des couleurs
 - ❖ Choix des icônes



[NFA035 : Java : bibliothèques et patterns]

- **Qualité du codage**
 - ❖ Architecture
 - ❖ Découpage
 - ❖ Réutilisation du code
 - ❖ Structuration
 - ❖ Lisibilité
 - ❖ Désignations
 - ❖ Commentaires

[NFA007 + NFA013 : Méthodes pour Informatisation + compléments] [NFA018 : Gestion projet informatique]

- **Qualité et contenu du rapport rendu**
 - ❖ Description du projet
 - ❖ Commentaires personnels
 - ❖ Un en-tête avec nom de projet, date, formation et nom des étudiants
 - ❖ Description de l'application réalisée avec quelques copies d'écrans représentatives.
 - ❖ Une procédure d'installation de l'application.
 - ❖ La description/ documentation d'éventuelles ressources externes utilisées dans le projet.

IV. Contraintes de conception

En plus des contraintes indiquées dans la partie liée aux besoins non-fonctionnels, le langage imposé pour la création du projet est soit PHP, soit JAVA.

Dans le cas de Java, il y a la liberté d'opter pour un **client lourd** (ex : Swing) ou un **client léger** (ex : Techno Java EE).

A cela s'ajoute la contrainte de temps puisque le projet est à **rendre le 02 juillet 2019**.

Le groupe doit être constitué de **deux auditeurs maximums**.



V. Technologies utilisées

1. Choix du langage

Nous avons opté pour **Java**, car c'est celui que nous avons le plus appris au cours de notre apprentissage. De plus étant donné que nous étions limités par deux langage de programmation dont PHP, il paraissait plus prudent de travailler avec celui que nous connaissons le mieux.

2. Solution logicielle

Nous avons le choix entre client léger et client lourd. Nous avons délaissé Swing pour opter une solution avec **Java EE**. Les raisons sont multiples, mais la principale est qu'il valait mieux travailler sur la solution la plus populaire du langage.

Java est selon nous, plus utilisé pour le Web. Dans l'objectif d'intégrer une entreprise, il nous a paru plus judicieux de réaliser un produit qui nous permettra de mettre en pratique des compétences recherchées pour ce langage.

3. Choix du support enregistrement de données

Pour garder une cohérence avec nos choix précédents (Java EE, client léger, priorité aux technologies étudiées), il paraissait logique d'opter pour la base de données, bien que les autres supports d'enregistrement étaient compatibles.

En effet nous avons eu l'occasion de travailler sur des bases de données en NFA008 et NFA011.

De plus la base de données est réputée fiable, solide et sécurisé, il s'offre comme le choix le plus intéressant de la liste.

Le système de gestion de base de données choisi est **MySQL** car il est gratuit, performant pour des petits projets et c'est celui que nous connaissons le mieux.



VI. Liste des fonctionnalités

1. Présentation

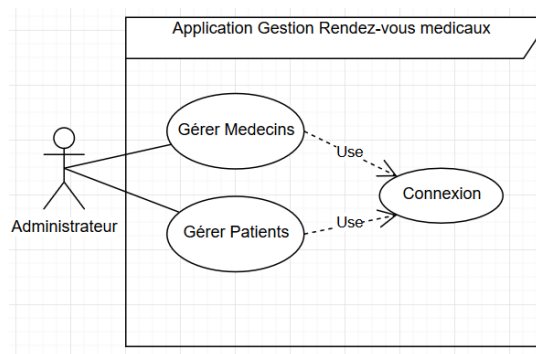
Ce chapitre servira à détailler les besoins fonctionnels abordés brièvement au chapitre « Définition des besoins ».

Nous nous servirons des **diagrammes de cas d'Utilisation (Use Case)** pour illustrer les interactions des différents utilisateurs avec le système.

2. Fonctionnalité de l'Administrateur

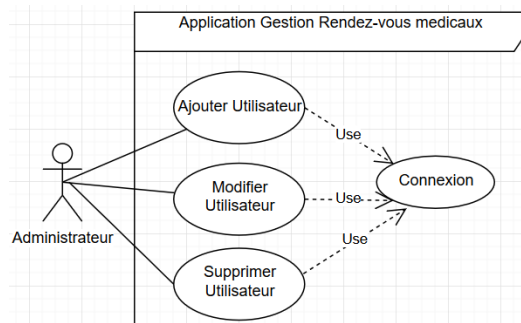
Cas d'utilisation globale

- Un administrateur peut gérer les médecins et les patients
- Il doit d'abord être identifié en tant qu'Administrateur (« include »)



Cas d'utilisation détaillé

- « Utilisateur » représente ici l'entité Médecin ou Patient (mais pas Admin !)
- La gestion d'un utilisateur se résume à **ajouter, modifier et supprimer** un utilisateur.

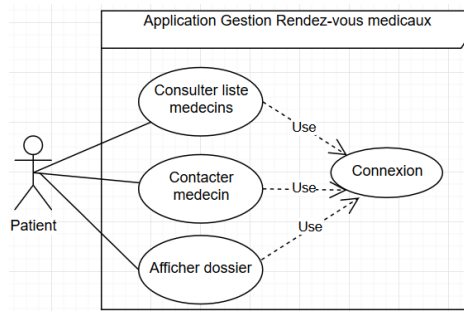




3. Fonctionnalité du Patient

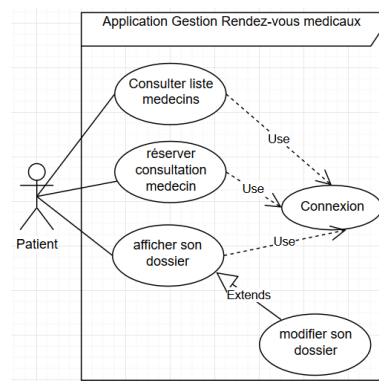
Cas d'utilisation globale

- Un Patient peut **consulter** la liste des médecins
- Un Patient peut **contacter** un médecin
- Un Patient peut **afficher** son dossier



Cas d'utilisation détaillé

- Un Patient peut **consulter** la liste des médecins
- Un Patient peut **réserver une consultation** auprès du médecin de son choix.
(C'est cette réservation qui représentera le contact avec le médecin)
- Un Patient peut **afficher son dossier** et par extension **modifier les informations de son dossier**.
(Pas de lien d'include du service « modifier dossier » avec « connexion » puisque pour modifier son dossier il doit obligatoirement l'afficher).

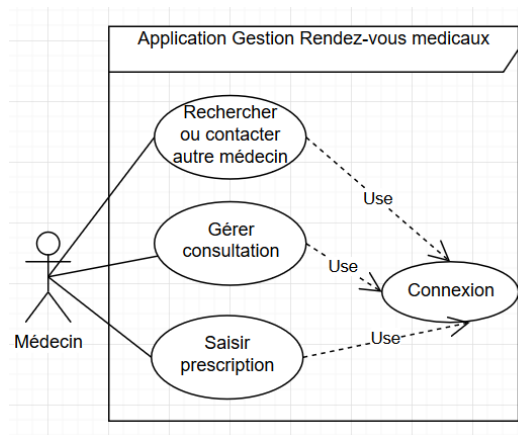




4. Fonctionnalité du Médecin

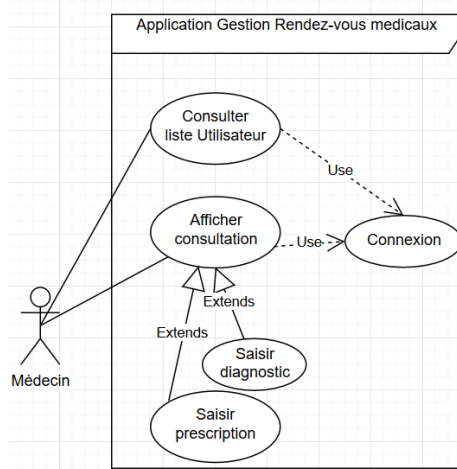
Cas d'utilisation globale

- Un Médecin peut **rechercher** ou **contacter** un autre médecin
- **Gérer** une consultation
- **Saisir** une prescription



Cas d'utilisation détaillé



- Le Médecin peut **consulter la liste des Utilisateurs** [Médecin ou Patient]
- Le Médecin peut **afficher une consultation** liée à un patient en le recherchant dans la liste.
- Il peut **saisir un diagnostic** (et la **modifier** en la ressaisissant une nouvelle fois)
- Il peut **saisir une prescription** (et la **modifier** en la ressaisissant une nouvelle fois)
- La saisie d'un diagnostic et d'une prescription ne sont possible que si la consultation existe.





VII. Maquette des écrans

Page d'Identification

IDENTIFIANT




MOT DE PASSE

☐ ADMIN
 ☐ PATIENT
 ☐ MÉDECIN

[J'ai oublié mon mot de passe.](#)

CONNEXION

Page Administrateur [gestion des patients]

Admin : admin

Gestion des patients | Gestion des médecins

Liste des Patients

numeroSS	nom	prenom	telephone	sexe	email	rue	ville	codePostal
545005858718464	Arsenio	Hyatt	04 92 44 83 50	H	vulputate.eu@etmalesuadafames.co.uk	CP 151, 8530 Cras Rue	Rockhampton	86412
692275191478546	Mackenzie	Peter	02 94 71 39 60	H	adipiscing.Mauris@ante.co.uk	Appartement 687-5610 Egestas Avenue	Pirmasens	01164
498175050577015	Phoebe	Quail	03 76 42 04 51	F	lectus@commodotincidunt nibh.org	6547 Non Impasse	Kidwelly	92843

Fiche Informations Patient

NUMERO SS :
 EMAIL :

NOM :
 ADRESSE :

PRENOM :
 VILLE :

TELEPHONE :
 CP :

SEXE : ☐ Homme ☒ Femme

AJOUTER **MODIFIER** **SUPPRIMER**

rechercher... **rechercher**



Page Administrateur [gestion des médecins]

Admin : admin

Gestion des patients
Gestion des médecins

Liste des Médecins

nom	prenom	Specialite	telephone	email	rue	ville	codePostal
Arthur	Kim	médecin généraliste	02 99 98 32 12	fermentum.metus.Aenean@euismodurnaNullam.net	8865 Molestie Av.	Retiro	42929
Judah	Fatima	médecin généraliste	03 40 51 90 42	ultrices@elitAliquamauctor.edu	CP 633, 2114 Malesuada Ave	Pepingen	14564
Justine	Cathleen	médecin généraliste	07 07 34 72 30	ac.mattis@luctuset.ca	155-1082 Nec Route	Bhagalpur	86265

Fiche Informations Médecin

NOM :
 PRENOM :
 SPÉCIALISTE :
 TELEPHONE :

EMAIL :
 ADRESSE :
 VILLE :
 CP :

AJOUTER
MODIFIER
SUPPRIMER

rechercher

Page Patient [Réservation des consultations]

Patient : login80

Reservation des consultations
Mes consultations
Afficher mon dossier

rechercher

Liste des Médecins

id	login	nom	prenom	Specialite	telephone	email	rue	ville	codePostal
1	medecin1	Arthur	Kim	médecin généraliste	02 99 98 32 12	fermentum.metus.Aenean@euismodurnaNullam.net	8865 Molestie Av.	Retiro	42929
2	medecin2	Judah	Fatima	médecin généraliste	03 40 51 90 42	ultrices@elitAliquamauctor.edu	CP 633, 2114 Malesuada Ave	Pepingen	14564
3	medecin3	Justine	Cathleen	médecin généraliste	07 07 34 72 30	ac.mattis@luctuset.ca	155-1082 Nec Route	Bhagalpur	86265

Fiche Informations médecin

nom prenom	Dr. Justine Cathleen
spécialité	médecin généraliste
téléphone	07 07 34 72 30
adresse	155-1082 Nec Route
CP, Ville	86265 Bhagalpur

Réserver une consultation



VOTRE NUMÉRO SS :
 VOTRE NOM :
 DATE :

VOTRE NR.TELEPHONE :
 VOTRE PRÉNOM :
 HEURE :

AJOUTER




Page Patient [Mes consultations]

[Profil de l'utilisateur](#)
 Profil de l'utilisateur

Reservation des consultations
Mes consultations
Afficher mon dossier


Rendez vous confirmé


 Dr Arthur Kim
 médecin généraliste
dimanche 09 juin 2019 à 08h00

#1

DEPLACER RDV
ANNULER RDV

VALIDER
✕


 Dr Arthur Kim
 médecin généraliste
dimanche 09 juin 2019 à 16h30

DEPLACER RDV
ANNULER RDV

Page Patient [Afficher mes dossiers]




Reservation des consultations
Mes consultations
Afficher mon dossier

Mes informations

VOTRE NUMÉRO SS :	093601097990364	VOTRE NR.TELEPHONE :	07 59 02 63 25
VOTRE NOM :	neron	VOTRE PRÉNOM :	Kalia
VOTRE EMAIL :	neque@non.co.uk	VOTRE RUE :	CP 314, 8216 Ante, Route
VOTRE CODE POSTAL :	62832	VOTRE VILLE :	Martigues
VOTRE SEXE :	F		

[MODIFIER INFORMATIONS](#)

<p>VOTRE NUMÉRO SS : 093601097990364</p> <p>VOTRE NOM : neron</p> <p>VOTRE EMAIL : neque@non.co.uk</p> <p>VOTRE CODE POSTAL : 62832</p> <p>SEXE : <input type="radio"/> Homme <input checked="" type="radio"/> Femme</p>	<p>VOTRE NR.TELEPHONE : 07 59 02 63 25</p> <p>VOTRE PRÉNOM : Kalia</p> <p>VOTRE RUE : CP 314, 8216 Ante, Rou</p> <p>VOTRE VILLE : Martigues</p>
--	---

[VALIDER MODIFICATIONS](#)



Page Médecin [Gestion des consultations]

Gestion des consultations | Les diagnostics (résultats de consultation) | Prescriptions | Chercher un medecin

Liste des Patients

numeroSS	nom	prenom	telephone	sexe	email	rue	ville	codePostal
545005858718464	Arsenio	Hyatt	04 92 44 83 50	H	vulputate.eu@etmalesuadafames.co.uk	CP 151, 8530 Cras Rue	Rockhampton	86412
692275191478546	MacKenzie	Peter	02 94 71 39 60	H	adipiscing.Mauris@ante.co.uk	Appartement 687-5610 Egestas Avenue	Pirmasens	01164
498175050577015	Phoebe	Quail	03 76 42 04 51	F	lectus@commodotinciduntinibh.org	6547 Non Impasse	Kidwelly	92843

VOIR CONSULTATIONS DE YULI

Consultations

jeudi 27 février 2020 13h00

Fiche Informations Patient:

Yuli Elaine
156530359574500
CP 854, 262 Mauris Impasse 62472 Wechelderzande
06 53 68 93 82
sit@SedmolestieSed.co.uk
Date de consultation : jeudi 27 février 2020 13h00

Page Médecin [Les diagnostics]

Gestion des consultations | Les diagnostics (résultats de consultation) | Prescriptions | Chercher un medecin

Liste des Patients

numeroSS	nom	prenom	telephone	sexe	email	rue	ville	codePostal
545005858718464	Arsenio	Hyatt	04 92 44 83 50	H	vulputate.eu@etmalesuadafames.co.uk	CP 151, 8530 Cras Rue	Rockhampton	86412
692275191478546	MacKenzie	Peter	02 94 71 39 60	H	adipiscing.Mauris@ante.co.uk	Appartement 687-5610 Egestas Avenue	Pirmasens	01164
498175050577015	Phoebe	Quail	03 76 42 04 51	F	lectus@commodotinciduntinibh.org	6547 Non Impasse	Kidwelly	92843

VOIR CONSULTATIONS DE YULI

Consultations

jeudi 27 février 2020 13h00

cliquez sur une consultation pour charger les info du diagnostic

Descriptif du diagnostic médical :

Allergies :

Antécédents médicaux :

Date du diagnostic :



Page Médecin [Les prescriptions]

[Gestion des consultations](#)
[Les diagnostics \(résultats de consultation\)](#)
[Prescriptions](#)
[Chercher un medecin](#)

Liste des Patients

numeroSS	nom	prenom	telephone	sexe	email	rue	ville	codePostal
545005858718464	Arsenio	Hyatt	04 92 44 83 50	H	vulputate.eu@etmalesuadafames.co.uk	CP 151, 8530 Cras Rue	Rockhampton	86412
692275191478546	MacKenzie	Peter	02 94 71 39 60	H	adipiscing.Mauris@ante.co.uk	Appartement 687-5610 Egestas Avenue	Pirmasens	01164
498175050577015	Phoebe	Quail	03 76 42 04 51	F	lectus@commodotinciduntlibh.org	6547 Non Impasse	Kidwelly	92843

VOIR CONSULTATIONS DE YULI

Consultations

jeudi 27 février 2020 13h00

cliquez sur une consultation pour charger les info du diagnostic

Prescription :

fff

Page Médecin [Chercher un médecin]

[Gestion des consultations](#)
[Les diagnostics \(résultats de consultation\)](#)
[Prescriptions](#)
[Chercher un medecin](#)

Liste des Medecins

id	login	nom	prenom	Specialite	telephone	email	rue	ville	codePostal
1	medecin1	Arthur	Kim	médecin généraliste	02 99 98 32 12	fermentum.metus.Aenean@euismodurnaNullam.net	8865 Molestie Av.	Retiro	42929
2	medecin2	Judah	Fatima	médecin généraliste	03 40 51 90 42	ultrices@elitAliquamauator.edu	CP 633, 2114 Malesuada Ave	Pepingen	14564
3	medecin3	Justine	Cathleen	médecin généraliste	07 07 34 72 30	ac.mattis@luctuset.ca	155-1082 Nec Route	Bhagalpur	86265

Fiche Informations médecin

nom	Dr. Arthur
prenom	
spécialité	médecin généraliste
téléphone	02 99 98 32 12
adresse	8865 Molestie Av.
CP, Ville	42929 Retiro



VIII. Architecture

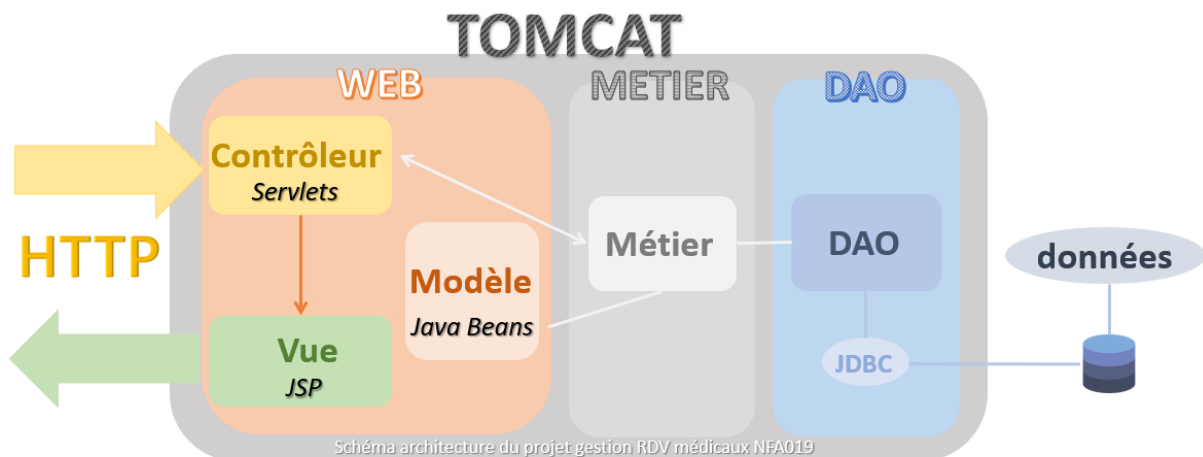
L'architecture du site respecte le **design pattern MVC** (Modèle – Vue – Controller), un modèle indispensable pour toute création d'un site web.

Ce modèle permet une séparation claire des responsabilités , ce qui permet :

- Une plus grande flexibilité
- Un code réutilisable
- Une meilleure lisibilité
- Un couplage faible donc une cohésion forte

Pour accentuer la séparation, nous avons décidé d'utiliser le **design pattern DAO** pour que l'accès à la base de données ne soit pas dépendant du Modèle, et l'utilisation du **design pattern Factory** pour qu'il soit possible de changer de type d'enregistrement sans modifier le reste.

Voici un schéma approximatif de l'architecture du projet :



Il y a des éléments qui ne sont volontairement pas représenté dans le schéma pour ne pas l'alourdir inutilement. Le détail de l'architecture sera expliqué dans les différentes parties de ce chapitre.



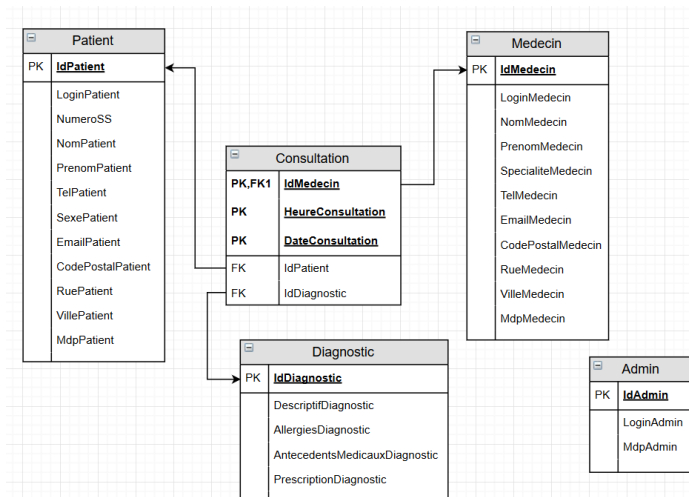
1. Architecture de la Base de données

Il y a 5 tables : Patient, Médecin, Consultation, Diagnostic et Admin.

Par soucis de simplicité, la prescription se résume en une propriété TEXT de la table diagnostic.

Pour la clé primaire de la table Consultation, on a estimé que la combinaison de l'IdMedecin, la date et l'heure de la consultation était une parfaite candidate pour respecter le principe d'unicité.

Voici le MLD (Modèle Logique de donnée) de la base de données :



Pour pouvoir tester notre base de données et lui donner des conditions proches de la réalité, nous l'avons rempli de données factices grâce au site :

<http://www.generatedata.com/>

Pas de surprise si les diagnostics sont remplis de Lorem Ipsum, et que les patients et médecins ont des noms anglophones.

En dehors de la table Diagnostic et Consultations, tous les champs ne peuvent pas être NULL.

Il y a une contrainte d'unicité pour le login et l'email, mais pas pour le numéro de sécurité social (par soucis d'application réelle possible, voir explication chapitre conclusion).

Le champ Sexe ne peut contenir que 'H' (Homme) ou 'F' (Femme).



D'autres champs auraient pu être vérifiés, mais de toute manière nous procéderons à des vérifications à plusieurs reprises, côté client partiellement, puis côté serveur où la vérification sera au cœur du traitement des données.

Dans un souci de sécurité, nous avons créé un user MySQL qui n'est pas root, mais qui a tous les droits pour cette BDD.

Dans le cas où un pirate viendrait à découvrir une brèche (comme par exemple : via une faille SQL), il ne pourra compromettre que cette base de données et pas celles du root.

D'ailleurs nous avons utilisé des méthodes nous permettant de nous prémunir de certaines failles dans notre code Java.

2. Architecture du code Java

- ▼ DossierMedicalPartage
 - > Deployment Descriptor: DossierMedicalPartage
 - > JAX-WS Web Services
 - ▼ Java Resources
 - ▼ src
 - > com.medical.beans
 - > com.medical.config
 - > com.medical.dao
 - > com.medical.filters
 - > com.medical.forms
 - > com.medical.servlets
 - > Libraries
 - > JavaScript Resources
 - build
 - ▼ WebContent
 - > inc
 - > META-INF
 - > panelMedecin
 - > panelPatient
 - > restreint
 - ▼ WEB-INF
 - ▼ lib
 - jsl-1.2.jar
 - mysql-connector-java-5.1.12-bin.jar
 - connexion.jsp
 - taglibs.jsp
 - web.xml

A gauche, l'arborescence du projet

Nous détaillerons chaque information dans les parties suivantes.

Le code est organisé à travers 6 packages :

- Beans
- Config
- Dao
- Filters
- Forms
- Servlets

A cela s'ajoute deux bibliothèques :

- JSTL
- MySQL Connector Java

Ainsi que les pages JSP :

- Dossier panelMedecin
- Dossier panelPatient
- Dossier restreint
- Connexion.jsp
- Taglibs.jsp

Et le fichier de configuration XML :

- Web.xml



3. Diagramme de classe des Beans

- ▼ com.medical.beans
 - > Admin.java
 - > Consultation.java
 - > Diagnostic.java
 - > Medecin.java
 - > Patient.java
 - > Utilisateur.java

Voici le contenu du packages **beans**.

Sans surprise, on retrouve le nom des tables identifiées dans le chapitre sur l'architecture de la base de données.

A l'exception près de la classe « **Utilisateur** » qui sert de classe mère aux classes « **Patient** » et « **Medecin** ».

Nous ne ferons que le diagramme de classe des **3 beans suivants** (Utilisateur, Patient et Medecin).

Les autres ne sont juste que la représentation en langage Java des tables SQL écrite plus haut.

Avant ça, nous souhaitant rappeler ce qu'est un Bean :

Un Bean (ou JavaBeans) est un objet Java qui est réutilisable, autonome et qui peut facilement être assemblé avec d'autres composants Beans pour créer un programme.

Pour qu'une classe Java puisse être appelé Bean, elle doit respecter plusieurs caractéristiques :

- ❖ Posséder un constructeur sans paramètre.
- ❖ Définir des propriétés [visibilité : private]
- ❖ Implémenter l'interface sérialisable (sauvegarde des données)
- ❖ Définir des méthodes utilisables [visibilité : public]

Si on fait abstraction de l'implémentation de l'interface sérialisable (qui n'a pas d'utilité pour nous dans le cadre de ce projet puisque on préfère enregistrer les données en BDD plutôt que l'objet sérialisé lui-même.), nos classes Java respectent ces caractéristiques.



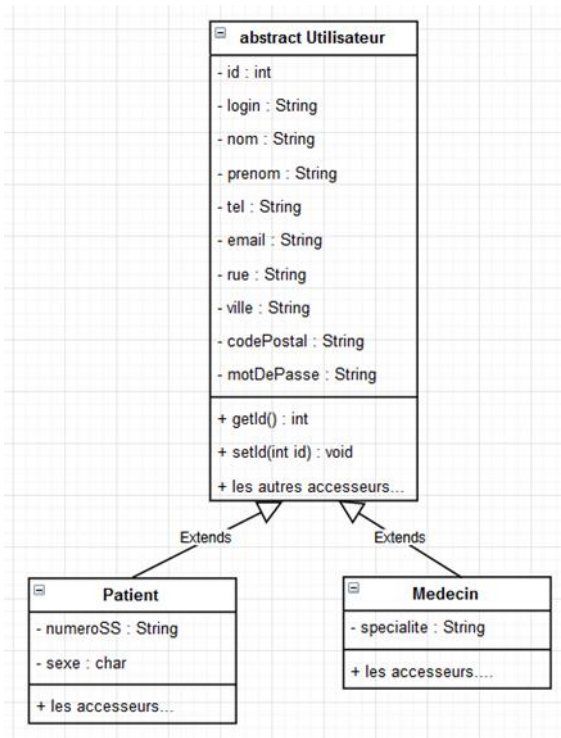
Dans le schéma à droite, vous pouvez voir les classes Patient et Medecin qui héritent de la classe mère abstraite Utilisateur.

Un des intérêts est d'éviter de réécrire plusieurs fois des données communes aux classes.

On pourra désormais écrire des méthodes qui pourront être utilisées à la fois par des objets Patients et des objets Médecins.

Avec une utilisation du **Wildcard** on pourra étendre ce puissant mécanisme avec des listes d'Utilisateurs.

Pour alléger le schéma à droite, nous n'avons pas retranscrit l'entière des accesseurs (getters/setters).



4. Utilisation du pattern DAO

Dans ce chapitre, nous allons vous détailler l'utilisation des classes DAO après vous avoir expliqué la raison de l'utilisation de ce pattern.

Nous profiterons en parallèle du package Dao, de vous parler de celui du package Config qui y est intimement lié.

Avant toute chose, nous aurions pu opter pour un ORM tel que Hibernate, qui implémente l'interface JPA, pour nous simplifier la vie et gérer le mapping à la base de données plus sereinement.

Nous ne l'avons pas fait pour plusieurs raisons, d'une part nous nous sentions plus à l'aise avec le pattern DAO puisque nous l'avons appris lors de l'étude du cours Java EE sur OpenClassrooms, d'autre part, nous avons envie de mettre en pratique les



méthodes apprises dans plusieurs cours du CNAM et écrire « manuellement » toutes les requêtes SQL.

Cela nous permet aussi d'être pleinement en contrôle sur notre base de données.

A cela s'ajoute la classe « DAOFactory », qui en plus d'être une fabrique d'objet, s'occupe de la connexion à travers le connecteur Java de MySQL.

La « DaoFactory » n'est probablement pas pleinement respecté, puisque nous n'avons pas d'objet DAO unique qui serait la mère des autres DAO liés aux objets.

Mais il garde l'essence du concept du design pattern Factory, qui est d'être un instanciateur d'objets DAO.

La DaoFactory contient une méthode statique nommée « getInstance() », qui sert d'instanciation de la connexion.

Cette méthode va récupérer des informations dans un fichier texte « dao.properties » qui contient les données de connexion à la base de données, c'est-à-dire l'url, le driver, le login et le mot de passe pour l'accès à la BDD.

Cette méthode va ensuite créer la DaoFactory et la retourner.

C'est par le biais de cette classe que nous pourrions obtenir les interfaces AdminDao, PatientDao, MedecinDao...

Il y a une convention d'écriture, le nom du Bean + DAO

- ▼ com.medical.config
 - > InitialisationDaoFactory.java
- ▼ com.medical.dao
 - > AdminDao.java
 - > AdminDaoImpl.java
 - > ConsultationDao.java
 - > ConsultationDaoImpl.java
 - > DaoConfigurationException.java
 - > DaoException.java
 - > DaoFactory.java
 - > DiagnosticDao.java
 - > DiagnosticDaoImpl.java
 - > MedecinDao.java
 - > MedecinDaoImpl.java
 - > PatientDao.java
 - > PatientDaoImpl.java
 - > dao.properties

Pour résumé, le package dao est constitué de :

- Un DaoFactory
- Un BeanDAO + son implémentation
Exemple:
AdminDAO, AdminDAOImpl
MedecinDao, MedecinDaoImpl

Il y a donc 2 x (nombre de Beans utiles)
- Les exceptions
(DaoConfigurationException et DaoException)

2 types d'exceptions pour identifier l'origine
- Un fichier properties



Pour éviter de surcharger le document, nous ne détaillerons qu'un Bean Dao ainsi que son implémentation. Les autres n'apporteront pas plus d'informations après en avoir vu un.

Prenons par exemple : MedecinDao et son implémentation.

```
public interface MedecinDao {

    * <b>Objectif</b> : Vérifier identification Utilisateur
    boolean estValideIdentification( Utilisateur utilisateur ) throws DaoException;

    * <b>Objectif</b> : Retourner la liste des Medecins
    List<Medecin> lister() throws DaoException;

    * <b>Objectif</b> : Récupérer les informations d'un medecin
    void recuperer( Medecin medecin ) throws DaoException;

    * <b>Objectif</b> : Récupérer les informations d'un medecin (méthode
    Medecin recuperer( int idMedecin ) throws DaoException;

    * <b>Objectif</b> : Ajouter un medecin
    void ajouter( Medecin medecin ) throws DaoException;

    * <b>Objectif</b> : Modifier un medecin
    void modifier( Medecin medecin ) throws DaoException;

    * <b>Objectif</b> : Supprimer un medecin
    void supprimer( int idMedecin ) throws DaoException;

}
```

MedecinDao est une interface qui contient les signatures méthodes qui devra implémenter MedecinDaoImpl.

La question qu'on peut se poser lorsqu'on écrit un DAO, c'est quelles sont les méthodes qui seront utilisées pour l'objet Médecin ?

On peut se baser sur le principe du CRUD (Create Read Update Delete), car après tout, ce sont les principales requêtes faites à la base de données.

Nous avons pris le soin de documenter toutes les méthodes, vous pourrez générer la javadoc et comprendre leurs fonctionnements.

Nous avons aussi pris le soin de n'utiliser que des requêtes préparées (PreparedStatement) pour éviter une attaque liée à la faille SQL.

En effet, la concaténation des données transmises avec la requête via un simple Statement peut être comprise via une transmission de données astucieuses. (Les commentaires --, la fermeture de la requête avec un simple quote, des assertions tel 1=1..)

Avant de passer à la partie suivante, il faut vous expliquer le principe du package Config contenant une unique classe nommée InitialisationDaoFactory.



Pour faire simple, cette classe hérite de `ServletContextListener`, qui lui permet de redéfinir – entre autres – la méthode `contextInitialized`.

Nous allons pouvoir transmettre à la `ServletContext` un attribut qui sera la `DaoFactory` instancié.

Cela évitera d'une part d'instancier plusieurs fois la `DaoFactory`, mais surtout de permettre aux servlets de récupérer les `BeansDao`, une seule fois, lors de l'unique instanciation de ces mêmes servlets, puisqu'elles les récupéreront via leur méthode `init()`.

Voici une capture écran de la méthode `Init()` d'une servlet :

```
@Override
public void init() throws ServletException {
    this.adminDao = (DaoFactory) getServletContext().getAttribute( CONF_DAO_FACTORY ).getAdminDao();
    this.patientDao = (DaoFactory) getServletContext().getAttribute( CONF_DAO_FACTORY ).getPatientDao();
    this.medecinDao = (DaoFactory) getServletContext().getAttribute( CONF_DAO_FACTORY ).getMedecinDao();
}
```

5. Séparation des accès avec l'objet Filtre

Comment empêcher l'accès d'une page JSP à un utilisateur qui devinerait l'adresse URL ?

Ou encore comment faire en sorte qu'un médecin qui a un accès privé, ne puisse pas accéder aux pages d'un patient ayant un autre accès privé ?

L'objet `Filter` nous permet de répondre à ces deux questions en se positionnant juste avant la servlet lors de la récupération des objets `HttpServletRequest` (request et response).

```
▼ com.medical.filters
  > RestrictionAdminFilter.java
  > RestrictionMedecinFilter.java
  > RestrictionPatientFilter.java
```

Il y a trois Filter car trois accès privés
(Admin, Médecin et Patient)

Convention d'écriture :
Rescription+nomBean+Filter

Les objets implémentent l'objet `Filter`, donc ils peuvent implémenter la méthode `doFilter` qui reçoit en paramètre `ServletRequest`, `ServletResponse` et `FilterChain`.

On cast les deux premiers objets en `HttpServletRequest` et `HttpServletResponse`.

(On a d'ailleurs tendance à dire `Servlet` pour faire allusion à un type spécifique de `Servlet` qui est `HttpServlet`. La `Servlet` ne se résume pas au protocole HTTP).

Ensuite on vérifie si la session contient un attribut `Session` associé au type d'utilisateur, si ce n'est pas le cas on lui renvoie la page `Connexion`.

Si oui, on appelle la méthode `doFilter` de l'objet `FilterChain` reçu en paramètre en lui transmettant l'objet request et response.



C'est ça manière de laisser passer les informations à la servlet ou à un autre Filtre qui le succéderait.

Comment savoir quand le filtre sera appelé ?

Il faut paramétrer cela en mappant l'url pattern avec le filtre.

On associe un url pattern au filtre concerné, par exemple : « /panelPatient/* » pour indiquer tout le contenu à l'intérieur du dossier nommé panelPatient.

Ce paramétrage peut se faire soit via le fichier web.xml (balises XML) ou en utilisant les annotations directement sur la classe concernée.

Nous avons opté pour le fichier web.xml car cela nous permet de définir l'ordre d'appels des filtres. (Bien que l'ordre n'ait pas été utile dans le cas de notre programme)

6. Objets Traitement

Les objets traitement sont contenu dans le package Forms (pour formulaires).

```

▼ com.medical.forms
  > AdminForm.java
  > ConsultationForm.java
  > DiagnosticForm.java
  > FormValidationException.java
  > MedecinForm.java
  > PatientForm.java
  > UtilisateurForm.java
  > UtilitaireForm.java
  
```

Il y a donc autant de Forms que de beans, soit 6 beans.

A cela s'ajoute un objet Exception spécifique au package.

+ un objet UtilitaireForm qui contient des méthodes dites utilitaires.

C'est dans ces objets que nous vérifierons la conformité des données.

Exemple : le code postal doit être constitué uniquement de 5 nombres.

On y fait usage de **REGEX** (Expressions Régulières), de **TRY/CATCH** et de lancement d'exception (**THROWS**).

La classe UtilisateurForm est une classe abstraite qui contient des méthodes communes à PatientForm et MedecinForm.

Pour un souci de récupération des informations d'erreurs, certaines méthodes sont en abstract et doivent être redéfinies par les classes héritées.

Toutes les classes Form sont constituées d'une part des méthodes validation, qui s'occupent de vérifier la conformité des données et de lancer des exceptions le cas échéant.



D'autres part, des méthodes de traitement qui s'occupent de faire appel à ces méthodes de validation, catcher les exceptions et incluent les messages d'erreurs dans un objet **Map** réservé à ça. Si aucune erreur ils s'occupent – parfois – de remplir un objet Bean reçu en paramètre, via son setter, de la donnée vérifiée et validée.

Comme indiqué plus haut, ce package ne fait pas exception, toutes les méthodes sont documentées comme il le faut pour permettre un confort à l'utilisateur/programmeur.

Il était important de ne pas faire l'impasse sur la documentation, que nous estimons essentiel pour le débogage.

7. Structuration des Servlets

▼ com.medical.servlets

- > AdminPanel.java
- > Connexion.java
- > Deconnexion.java
- > MedecinPanel.java
- > PatientPanel.java

Il y a 5 servlets :

- Connexion (page d'identification)
- Déconnexion (suppression de la session)
- AdminPanel
- MedecinPanel
- PatientPanel

Les Servlets (Controller) sont appelées par le biais d'un submit de formulaire HTML.

Elles s'occupent ensuite d'aiguiller les informations aux objets Form selon l'intention de l'utilisateur.

Après le traitement des objets de traitement (Form), il s'occupe de renvoyer au navigateur une page JSP associée en modifiant/renvoyant les sessions si nécessaire.

Certaines récupérations d'informations se font par le biais des paramètres URL.

On a décidé d'avoir une servlet pour un utilisateur, mais nous aurions pu avoir une servlet pour une action utilisateur. (Ou une page Jsp comme conseillé).

Il nous a paru plus pratique de ne privilégier qu'une page par Utilisateur et nous ne voulions pas alourdir l'architecture.

Mais il est clair que dans le cadre d'un projet plus ambitieux, il serait probablement plus prudent de respecter autant que possible la règle d'une page JSP = une Servlet pour bien gérer le traitement des flux vers le Controller.



8. Librairies externes utilisées

▼ lib

📄 jstl-1.2.jar

📄 mysql-connector-java-5.1.12-bin.jar

Il n'y a que deux bibliothèques, la JSTL et MySQL Connector.

Le Jar MySQL Java Connector est évidemment essentiel pour que Java puisse piloter la base de données MySQL.

Concernant la JSTL (Java Server page Standard) , elle sert à étendre la JSP (Java Server Pages) en ajoutant une bibliothèque de balises pratiques qui nous permettra d'éviter de mettre du code Java dans nos pages JSP.

Le but c'est d'obtenir un langage de balisage proche de ceux utilisés par les webdesigners (HTML/XML).

Le code de la VUE est plus agréable et propre avec la JSTL, il n'y a désormais plus de Java dans le code et la séparation est pleinement faite.

Exemple d'affichage d'une valeur transmise via la request :

```
<c:out value="${admin.login}"/>
```

Utilisation de la JSTL + Utilisation de L'EL (Expression Language)

Il faut au préalable définir la bibliothèque JSTL qu'on appellera c

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

9. Architecture des pages HTML/CSS



Les pages HTML sont en fait des pages JSP dans ce projet, il y a donc principalement du code HTML avec une utilisation de la JSTL pour les récupérations de variables et les boucles par exemple.

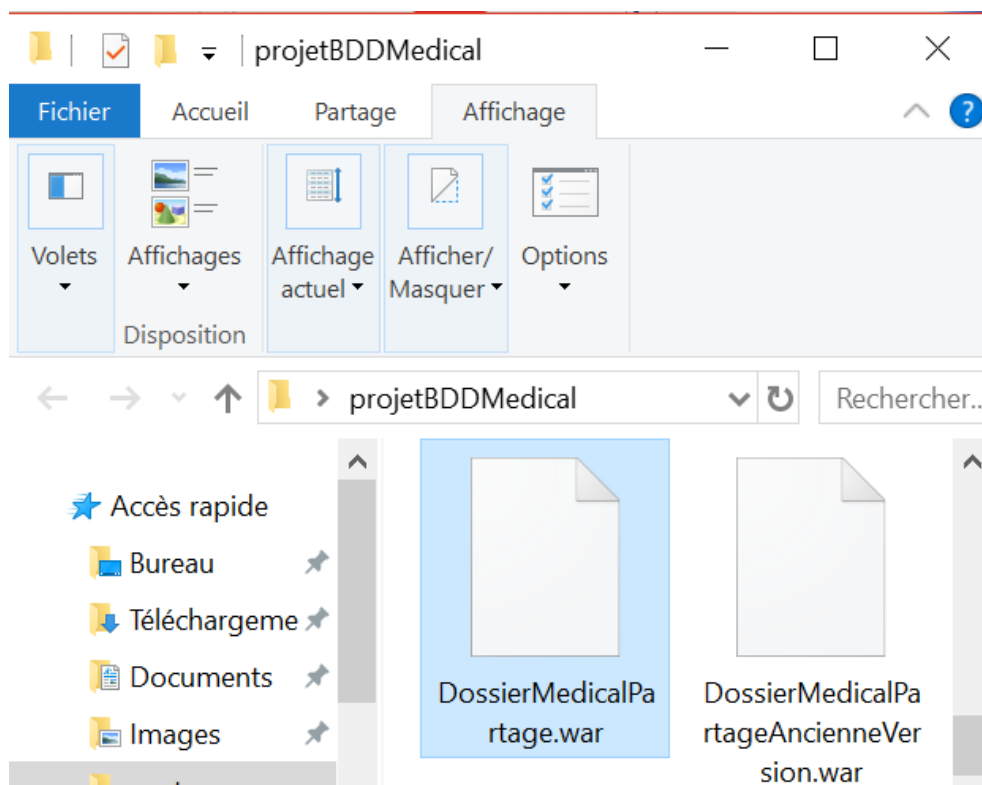
La JSTL ne sert pas uniquement à cacher le code Java, il permet d'utiliser des données de manière sécurisée en empêchant l'exploitation d'une faille célèbre : XSS (Cross Site-Scripting).

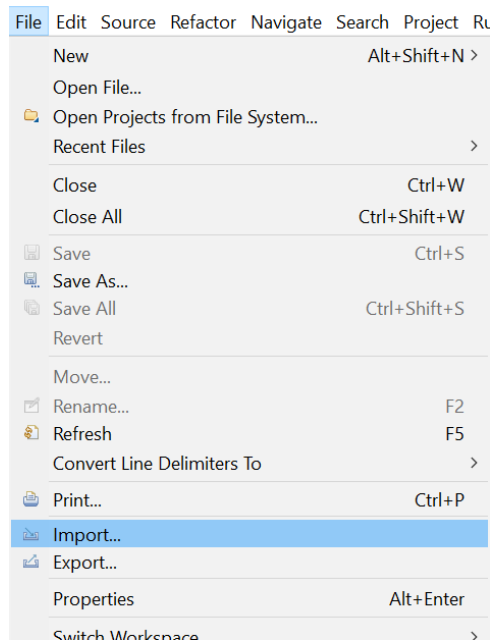
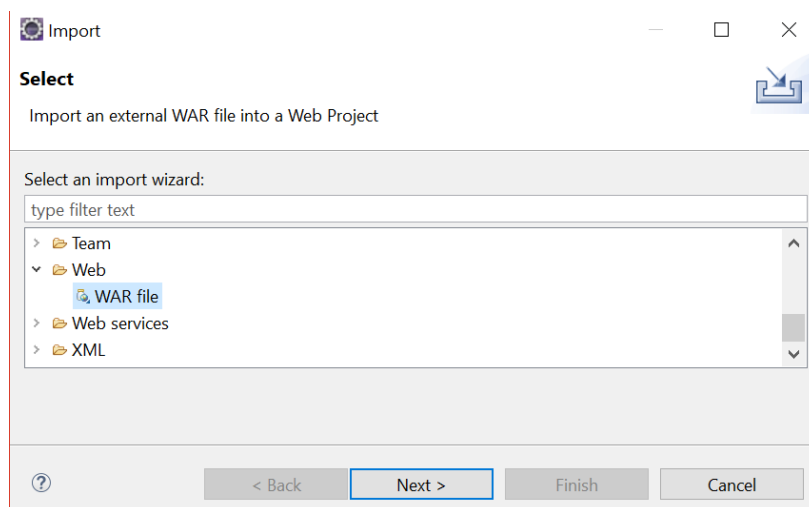
Ainsi le pirate qui souhaiterait ajouter des balises dans un formulaire dans le but de les faire interpréter par le site (côté client et serveur) et notamment des balises script contenant du code JavaScript, sera empêchée par l'usage des balises de la JSTL.

En fait il s'occupe notamment de transformer les caractères tel que <, >, ' & en <, >, ", ' et &.

IX. Procédure d'installation

ETAPE 1 : Récupération de l'objet WAR (Web application Archive)



**ETAPE 2 : Sur votre Eclipse, cliquez sur Import****ETAPE 3 : Sélectionner WAR File (dans le dossier Web)**



ETAPE 4 : Ajouter le client du fichier WAR récupéré et donner lui un nom de projet, puis appuyez sur **NEXT>**

IMPORTANT : mettre la version 9 d'Apache Tomcat (ou supérieure) dans Target runtime.

Import

WAR Import

Import a WAR file from the file system

WAR file: C:\Users\adelk\Desktop\projetBDDMedical\DossierMedicalPartage.war Browse...

Web project: DossierMedicalPartagee

Target runtime: Apache Tomcat v9.0 New...

EAR membership

☐ Add project to an EAR

EAR project name: EAR New Project...

< Back Next > **Finish** Cancel

ETAPE 5 : Ne pas oublier de cocher toutes les bibliothèques puis cliquer sur **FINISH**

Import

WAR Import: Web libraries

Select the web library jars from the list below to be imported as web library projects. Unselected web libraries will be imported as jars in the WEB-INF/lib directory.

☒ jstl-1.2.jar

☒ mysql-connector-java-5.1.12-bin.jar

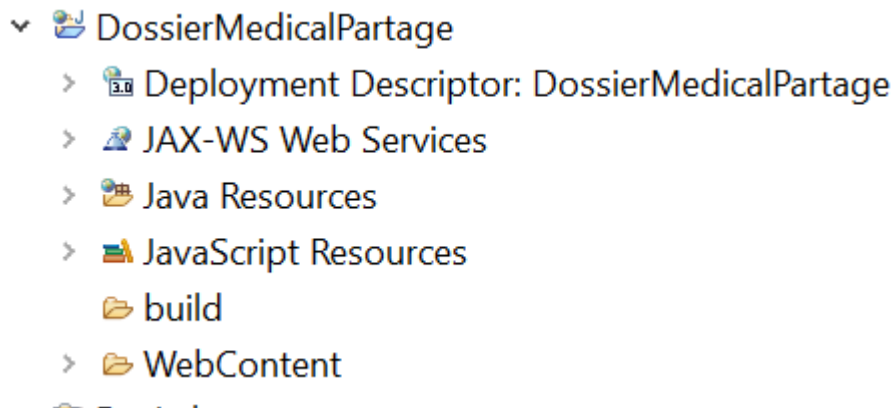
Select All Deselect All

< Back Next > **Finish** Cancel



ETAPE 6 : Il n'y a pas d'erreur dans la récupération du projet : **BRAVO** vous avez récupéré le projet.

Si ce n'est pas le cas, veuillez vérifier si vous avez bien respecté les précédentes étapes et recommencez si nécessaire.



ETAPE 7 : Récupération des fichiers MySQL

BdDossierMedical.sql contient la création de la base de données et des tables.

BdDossierMedical	08/06/2019 14:37	Fichier SQL	4 Ko
------------------	------------------	-------------	------

insertionBdDossierMedical.sql : l'insertion des données dans la base existante.

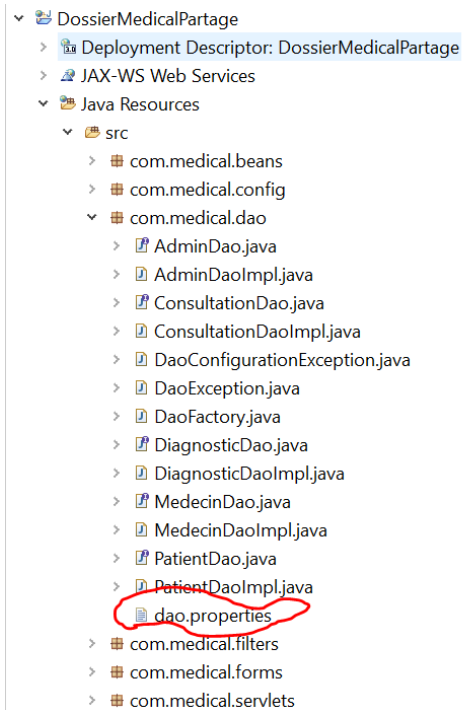
insertionBdDossierMedical	08/06/2019 14:44	Fichier SQL	92 Ko
---------------------------	------------------	-------------	-------

ETAPE 8 : Lancement du script **BdDossierMedical.sql** dans MySQL en mode Root

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SOURCE C:\Users\adelk\Desktop\projetBDDMedical\BdDossierMedical.sql
```

ETAPE 9 : Lancement du script **insertionBdDossierMedical.sql** dans MySQL en mode Root

Pour éviter un problème d'encodage indépendant de notre volonté, nous vous conseillons de coller directement le contenu du fichier dans votre invité de commande MySQL.

**ETAPE 10** : Modification des données du fichier dao.properties**AJOUTER VOTRE NOM UTILISATEUR + MOT DE PASSE MYSQL**

```
dao.properties
1url = jdbc:mysql://localhost:3306/BdDossierMedical?zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimezone=Europe/Paris&useSSL=false
2driver = com.mysql.jdbc.Driver
3nomutilisateur = ica
4motdepasse = rius
```

Si vous ne savez pas vos login mysql, c'est que vous n'avez sûrement pas de mot de passe.

Mettez alors pour le nom d'utilisateur et le mot de passe :

nomutilisateur=root

motdepasse=

Laisser le champ mot de passe vide, revient à dire qu'il n'y a pas de mot de passe.

**ETAPE 10 (BIS) : Création d'un utilisateur spécifique dans MySQL pour les droits d'accès**

Si vous ne souhaitez pas modifier les propriétés (url, login et password) présentent dans le fichier properties dans le package dao du programme, vous pouvez éventuellement créer un utilisateur qui aura le même nom et login que celui que nous avons utilisé pour ce programme.

On vous rappelle les commandes à saisir pour la création d'un utilisateur en base de données MySQL.

Dans notre cas ce sera :

```
CREATE USER 'ica'@'localhost' IDENTIFIED BY 'rius';  
  
GRANT ALL PRIVILEGES ON bdDossierMedical.* TO 'ica'@'localhost';
```

Avec **ica** comme nom d'utilisateur et **rius** comme mot de passe.

Une référence à Icare de la mythologie grecque, qui se brûla les ailes en s'approchant trop près du soleil.

X. Documentation ressources externes

Le tutoriel de Java EE d'Openclassrooms a été un site qui nous a servi de modèle dans la construction de l'architecture du site.

<https://openclassrooms.com/fr/courses/626954-creez-votre-application-web-avec-java-ee>

L'ensemble des icônes ont été récupérées sur ce site :

<https://www.flaticon.com/>

La chaîne de Dominique Liard a été une aide précieuse pour la compréhension des résolutions sur les problèmes de failles SQL et XSS en Java. (Dans sa playlist JDBC et sa playlist Jakarta EE).

<https://www.youtube.com/channel/UCI8T9GRhma8C2PaRfGljOtA/playlists>

Ce site nous a servi de modèle pour le HTML/CSS du formulaire de connexion.

http://webdesigntutsplus.s3.amazonaws.com/tuts/297_simpleAdmin/demo/login.html



XI. Conclusion

La limite de temps et notre obligation liée à la production d'autres projets ont empêché une amélioration significative du projet, bien que nous ayons – ils nous semblent – rempli toutes les exigences fonctionnelles du produit.

Nous voulions mettre en place un numéro de Sécurité Social qui respecte le modèle de la réalité mais nous n'avions pas, à travers les objets utilisateurs, les données nécessaires à la création de ces numéros de sécurité social.

Puisqu'il faut notamment des informations comme le mois de naissance, le département et la commune.

Avec une base de données fictive, que nous nous sommes imposée dès le début et qui nous a été d'ailleurs très utile, il aurait été contraignant de mettre une contrainte d'unicité sur un champ (numéro de SS) qui n'aurait pas été respecté lors des tests.

Un autre problème, c'est de ne pas avoir eu le temps de trouver des données liées au monde médical pour avoir des données plus saisissantes. (Notamment dans la rédaction des diagnostics et prescriptions.)

Nous n'avons trouvé que des données répertoriant les médicaments en France, que nous aurions pu intégrer dans la base, mais qui ne paraissaient pas suffisamment intéressantes pour qu'on l'y figure. (Nous n'avons pas de champ médicament)

Si nous n'étions pas surchargés par d'autres projets, nous aurions pu créer une table Prescription qui permettrait d'avoir des champs comme Médicament.

Aussi ajouter une boîte mail pour les utilisateurs, leurs permettant ainsi de dialoguer directement via leur accès entre eux.

Nous avons aussi prévu d'ajouter la possibilité de renvoyer un mail à l'utilisateur dans le cas où le mot de passe était perdu, par le biais de JavaMail, mais c'était un détail cosmétique vis-à-vis du besoin mis en place par le projet, et il valait mieux se concentrer sur l'efficacité des besoins fonctionnels.

Dernière chose, nous aurions pu ajouter un pool de connexion avec bonecp (pour la gestion des multiples connexions) et mettre le projet en ligne pour concrétiser le projet.

En dehors de ça, nous sommes pleinement satisfaits du produit réalisé au vu des heures que nous avons engagées.

On aurait aimé y ajouter aussi un planning de projet pour montrer comment le travail a été géré au niveau du temps.

Peut-être que certains éléments changeront au fil des jours, si nous venons à conclure les autres projets et qu'ils nous restent encore un peu de temps pour peaufiner le projet.

On espère en tout cas qu'il répondra aux attentes du client.