

# PAGE DE SERVICE

**Référence :** Vinci Thermo Green

**Plan de classement :** stadium-technic-analyse-conception-thermo-green

**Niveau de confidentialité :** confidential

### Mises à jour

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Auteur** | **Description du changement** |
| 3.1.0 | 02-11-2020 | Achot BARSEGHYAN | Authentification avec MDP haché. |

### Livraison

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Auteur** | **Description du changement** |
| 3.1.0 | 23-11-2020 | Achot BARSEGHYAN | Authentification avec MDP haché. |

# OBJET DU DOCUMENT

Ce document décrit l'implémentation Java du projet Vinci Thermo Green. Ce projet vise à produire une application réalise l'implémentation du diagramme des classes métier et du diagramme de séquence objet présenté dans la documentation d'analyse conception.

# SOMMAIRE

[PAGE DE SERVICE 1](#_Toc56982538)

[Mises à jour 1](#_Toc56982539)

[Livraison 1](#_Toc56982540)

[OBJET DU DOCUMENT 1](#_Toc56982541)

[SOMMAIRE 2](#_Toc56982542)

[1. ARCHITECTURE 3](#_Toc56982543)

[2. IMPLEMENTATION DES CLASSES METIERS 4](#_Toc56982544)

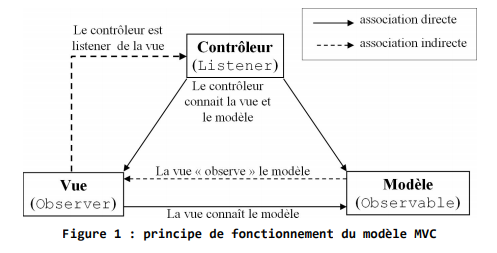
[3. LA CONNEXION A LA BASE DE DONNEES 4](#_Toc56982545)

[4. LA FONCTION QUI PERMET DE VERIER LE MOT DE PASSE 5](#_Toc56982546)

[5. LA FONCTION PERMETTANT LA RÉCUPÉRATION DES UTILISATEURS 5](#_Toc56982547)

# ARCHITECTURE

Conformément à l'analyse conception, la réalisation de l'application est structurée en trois couches selon une structure qui ressemble à un design-pattern MVC (Model View Controler) mais qui en réalité ne l'est pas vraiment. Cependant cette structure du code permet d'envisager dans une version ultérieure une évolution vers un modèle réellement MVC.

Le modèle MVC fonctionne selon le principe illustré par le schéma ci-dessous1 :

La conception de la v.2.0.0 de l'application Vinci Thermo Green prévoit la mise en œuvre d'un contrôleur. Ce contrôleur représente non pas le "listener" au sens MVC du terme mais le Data Access Object (DAO)2 d'une architecture n-tiers. Cependant, l'utilisation de la bibliothèque graphique Swing permet d'écouter la vue au sens propre du design-pattern MVC. Cela pourra être abordé lors d'une version ultérieure.

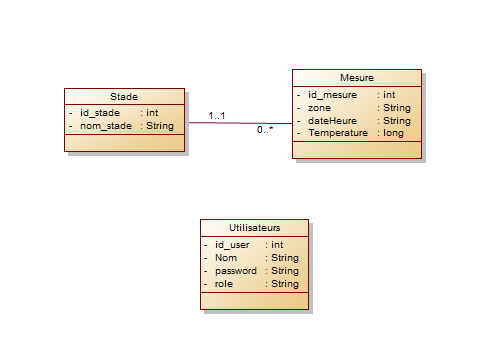
L'utilisation d'un DAO permet de s'abstraire de la façon dont les données sont stockées au niveau des objets métier. Ainsi, le changement du mode de stockage ne remet pas en cause le reste de l'application. Seules les classes dites "techniques" seront à modifier.

Les objets en mémoire vive sont liés à des données persistantes (stockées en base de données, dans des fichiers, dans des annuaires, etc…). Le modèle DAO regroupe les accès aux données persistantes dans des classes techniques spécifiques, plutôt que de les disperser. Il s'agit surtout de ne pas écrire ces accès dans les classes "métier", qui ne seront modifiées que si les règles de gestion métier changent.

Ce modèle complète le modèle MVC (Modèle - Vue - Contrôleur), qui préconise de séparer dans des classes les différentes problématiques :

* des "vues" (charte graphique, ergonomie)
* du "modèle" (cœur du métier)
* des "contrôleurs" (tout le reste : l'enchaînement des vues, les autorisations d'accès, etc…)

# IMPLEMENTATION DES CLASSES METIERS

L'analyse a permis de modéliser les classes métiers selon le diagramme ci-dessous (non documenté, cf. document d'analyse-conception) :

L’application récupère les données depuis une base de données, toutes les mesures, les utilisateurs et les stades. La classe "Mesure" stocke les T°Fahrenheit dans un attribut float et propose une méthode getCelsius() qui retourne la T°Celsius. L'échelle Fahrenheit est calée sur l'échelle Celsius par la relation :

T(°F) = 1,8 T(°C) + 32

T(°C) = (T(°F) -32)/1,8

# LA CONNEXION A LA BASE DE DONNEES

**public** BDD() {

// DATABASE

**try** {

*myConn* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/vincithermogreen?zeroDateTimeBehavior=CONVERT\_TO\_NULL&serverTimezone=UTC","root", "");

*myStmt* = *myConn*.createStatement(ResultSet.***TYPE\_SCROLL\_INSENSITIVE***, ResultSet.***CONCUR\_READ\_ONLY***);

} **catch** (SQLException e) {

e.getMessage();

}

}

Dans la classe BDD on implémente un try catch avec un DriverManager et createStatement qui permette de connecter notre application a la base de données.

# LA FONCTION QUI PERMET DE VERIER LE MOT DE PASSE

**public** **boolean** getCorrectPassword() **throws** SQLException {

String correcPass = *Bdd*.getPasswordfrom(*userName*);

**if** (BCrypt.*checkpw*(password, correcPass)) {

txtNomDutilisateur.setText(**null**);

passwordField.setText(**null**);

info\_connection.setVisible(**false**);

**return** **true**;

}**else** {

info\_connection.setVisible(**true**);

**return** **false**;

}

}

Cette fonction qui se trouve dans la classe LoginGUI, prend en paramètre le nom d’utilisateur et retourne un booléen.

Elle vérifie le mot de passe crypté présent dans la base de données et le mot de passe rentré dans le champ password en clair de la page connexion, grâce la une méthode présent la librairie BCrypt :

BCrypt.*checkpw*(password, CryptedPassword)

# LA FONCTION PERMETTANT LA RÉCUPÉRATION DES UTILISATEURS

**public** **void** getUsersFromDatabase() **throws** SQLException {

tab = Bdd.getAllUsers();

//Parcours lignes et colonnes pour les deux for, respectivement

**for**(**int** i=0; i<tab.length; i++)

{

String tab0 = tab[i][0];

String tab1 = tab[i][1];

model.addRow(**new** Object[] { tab0, tab1});

}

}

Cette fonction appelle une autre fonction présent la classe BDD qui récupère tous les utilisateurs de la base de données et renvoi un tableau à deux dimensions avec l’identifient et le password de chaque utilisateurs:

**public** String[][] getAllUsers() **throws** SQLException {

ResultSet reqUsers = **null**;

reqUsers = *myStmt*.executeQuery("select id\_user,login from user");

reqUsers.last();

**int** row = reqUsers.getRow();

reqUsers.first();

String[][] lesUsers = **new** String[row][2];

System.***out***.println(row);

**int** i = 0;

**while** (reqUsers.next()) {

lesUsers[i][0] = reqUsers.getString("id\_user");

lesUsers[i][1] = reqUsers.getString("login");

System.***out***.println(Arrays.*toString*(lesUsers[i]));

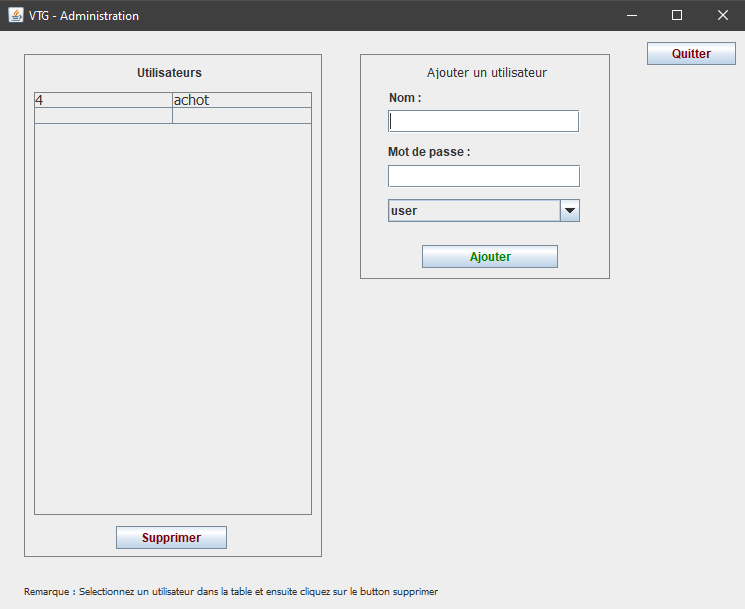
i++;

}

**return** lesUsers;

}

Une fois tous les utilisateurs récupérés, on les rentre dans une Jtable :



La Jtable qui contient tous les utilisateurs