# Dossier Réalisation
# Vinci Thermo Green

***Version : v3.1.0***

Mise à jour

| Version | Date | Auteur | Description changement |
|---|---|---|---|
| **3.0.0** | 02/11/2020 | GROUSSET Sylvain | Implémentation multi-sites |
| **3.1.0** | 23/11/2020 | GROUSSET | Identification & JBcrypt |

## Table des matières



**Objet du document**

Ce document décrit l'implémentation Java du projet Vinci Thermo Green.
Il présentera l'application réalisé à partir de l'implémentation du diagramme des classes métier et du diagramme de séquence objet présenté dans le dossier d'Analyse-Conception.
Il montrera les points majeurs, importants et complexes de l'application nécessitant documentation.

## 1. Architecture

L'architecture de l'application a été réalisée avec le pattern de MVC (Modèle Vue Controller). Ce pattern et structuré en 3 couches :

- Modèle
- Vue
- Controller

Chaque couche a un rôle bien spécifique :

- La vue s'occupe de tous les éléments visuels de l'application.
- Le contrôleur s'occupe du traitement des données entre le modèle et la vue
- Et donc le modèle traite les données envoyées par le contrôleur

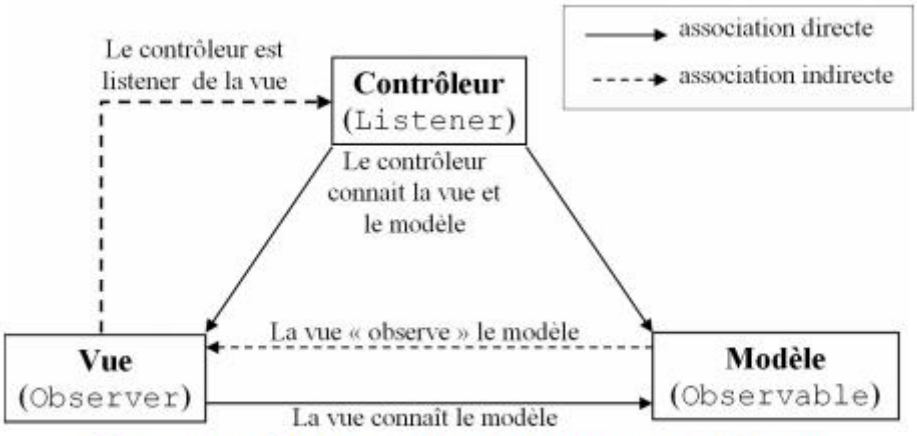



**Figure 1 : principe de fonctionnement du modèle MVC**

## 2. Implémentation des classes métier

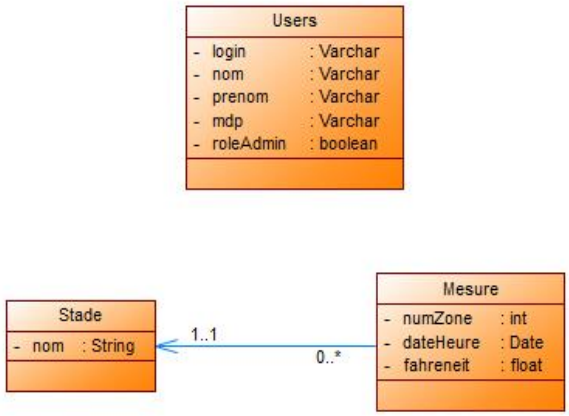



*Figure 2 : Diagramme des classes métiers*

L'analyse de la version v3.1.0 a permis d'établir le diagramme des classes ci-dessus.
Une classe Users sera créée au niveau de la base de données pour permettre de gérer les utilisateurs.

La classe Mesure implémentée en java permettra de stocker les températures en Fahrenheit.

Une méthode ***getCelsius*** permet de convertir une température en Fahrenheit à Celsius par la formule : `(fahrenheit - 32.0f) / 1.8f`

```
package model;
import java.util.Date;

public class Mesure {
	/**
	 * <p>numZone contient le numéro de la zone mesurée</p>
	 */
	private String numZone;
	/**
	 * <p>horoDate contient la date et l'heure de la mesure au format aa-mm-jj hh:mm</p>
	 */
	private Date horoDate;
	/**
	 * <p>valFahrenheit contient la valeur de la température mesurée en degré
Fahrenheit</p>
	 */
	private float fahrenheit;

	public Mesure() {
		this.numZone = new String();
		this.horoDate = new Date();
		this.fahrenheit = 0.0f;
	}

	public Mesure String pZone, Date pDate, float pFahrenheit) {

		this.numZone = pZone;
		this.horoDate = pDate;
		this.fahrenheit = pFahrenheit;
	}
```

```
    public String getNumZone() {
        return numZone;
    }

    public void setNumZone(String numZone) {
        this.numZone = numZone;
    }

    public Date getHoroDate() {
        return horoDate;
    }

    public void setHoroDate(Date horoDate) {
        this.horoDate = horoDate;
    }

    public float getFahrenheit() {
        return fahrenheit;
    }

    public void setFahrenheit(float valFahrenheit) {
        this.fahrenheit = valFahrenheit;
    }

    /**
     * <p>Convertit Fahrenheit en °Celsius</p>
     * @since 2.0.0
     * @return float t°Celsius
     */
    public float getCelsius() {
        //return (float) (valFahrenheit - 32) / 1.8)
        return (fahrenheit - 32.0f) / 1.8f;
    }
}
```

## 3. Implémentation de la base de données

Avant de pouvoir utiliser l'application, il faut que la base de données soit implémentée. Nous utilisons une base de données sous MySQL.

Prérequis :

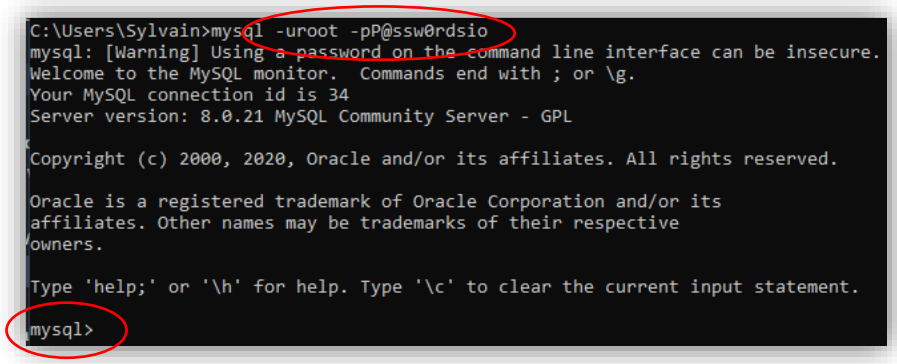
- MySQL Server installé

&

- MySQL Server ajouté au PATH

**Etape 1 :**

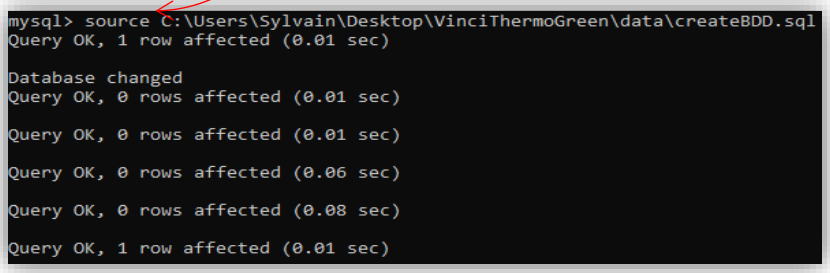
Se connecter au serveur MySQL en invite de commande avec votre utilisateur MySQL.
*mysql –u<Utilisateur> -p<MDP>*




**Etape 2 :**

Lorsque connecté, lancer le script de création de la base de données avec la commande :
*source <chemin vers createBDD.sql>*

Le fichier de création de la base de données '*createBDD.sql*' se trouve dans le dossier *data* du build du projet.




Attention, le script ne fonctionne qu'une seule fois. Si vous tentez de le lancer 2 fois pour x raisons, il vous affichera une erreur au niveau du **CREATE USER** avec une erreur **1396 (HY000).** Pour pallier ce problème il faut mettre un nom d'utilisateur différent de celui créé précédemment. Si vous êtes amenés à changer ce nom d'utilisateur, des modifications dans le code sont nécessaires. Il vous faudra appeler notre service d'aide.

## 4. Base de données

La base de données est constituée de 3 tables :

- STADE
- MESURE
- USERS

| **USERS** |
|---|
| login, varchar |
| nom, varchar |
| prenom, varchar |
| Mdp, varchar |
| roleAdmin, Boolean |

| **STADE** |
|---|
| Nom |

| **MESURE** |
|---|
| numZone, Int |
| dateHeure, dateTime |
| Fahrenheit, float |
| #Nom_stade, varchar |

- Les tables Stade et Mesure sont en relations : Une mesure concerne 1 et 1 seul stade et un stade peut avoir 0 ou plusieurs Mesures. Donc en plus de l'identifiant numZone dans la table Mesure, il y a une référence (clés étrangère) du nom du stade dans Mesure (attribut Nomstade).

- Un stade est identifié par son nom.

- La table USERS ne possède pas d'identifiant, celle-ci est indépendante des 2 autres tables présente dans la base de données.

Un script d'insertion de base est fourni avec le script de création des tables pour que l'application soit un minium peuplée lors de son implémentation.
Ce script comprend : L'insertions de 2 stades avec des températures pour chacun des stades ainsi qu'un utilisateur avec le rôle d'administrateur ayant comme identifiant : **SGROUSSET** et comme mot de passe : « **bpsen** ».

## 5. Connexion à la base de données.

La connexion à la base de données se fait dans la classe **données** via la méthode « openDatabase » mais est exécutée depuis le contrôleur.

```
public void openDatabase() {

		String url = "jdbc:mysql://localhost:3306/vinci?serverTimezone=UTC";
		String username = "adminVinci";
		String password = "vinciThermogreen";

		try {

			Connection conn = DriverManager.getConnection(url, username,
password);
			System.out.println("Connexion base OK !");
			setConn(conn);

		} catch (SQLException e) {

			throw new IllegalStateException("Cannot connect database !", e);
		}
```

Cette méthode appelle la méthode « getConnection » de DriverManager avec en paramètre les informations pour se connecter à la base de données : l'URL, l'username et le mot de passe de l'username.

Puis le tout est encapsulé dans un try/catch pour attraper les éventuelles erreurs.

## 6. Cryptage avec JBcrypt

Les mots de passe des utilisateurs sont cryptés via la librairie JBcrypt.
Les méthodes concernant le cryptage sont appelées à 2 endroits dans le programme :

- Lorsque l'utilisateur Admin créer un compte (le MDP est crypté)
- Lorsque l'utilisateur se connecte

Lorsque l'utilisateur ADMIN crée un compte :

```
public void actionPerformed(ActionEvent arg0) {

                        try {
                                control = new Controller();
                        } catch (ParseException | SQLException e) {
                                e.printStackTrace();
                        }

                        @SuppressWarnings("deprecation")
                        String mdp = BCrypt.hashpw(passwordField.getText(),
BCrypt.gensalt(10));
                        String nom = textFieldNom.getText();
                        String prenom = textFieldPrenom.getText();
                        String login = prenom.charAt(0)+nom;
                        boolean adminCheckBox = chckbxAdmin.isSelected();

                        try {
                                control.createAccount(login, nom, prenom, mdp,
                                adminCheckBox);
                                JOptionPane.showMessageDialog(null, "Compte ajouté avec
                                succès !");

                                textFieldNom.setText("");
                                textFieldPrenom.setText("");
                                passwordField.setText("");

                        } catch (SQLException e) {
                                e.printStackTrace();
```

Ce bout de code correspond à lorsque l'utilisateur clique sur le bouton Valider pour créer un compte.
Le bout de code utilisé pour crypter le mot de passe est le suivant :

```
String mdp = BCrypt.hashpw(passwordField.getText(), BCrypt.gensalt(10));
```

La méthode BCrypt.hashpw prend 2 paramètres : Le String à hacher, et du salage.
Ici, on lui fournit en 1er paramètre le passwordField.getText() (le mot de passe entré dans le passwordField) qui est la valeur à hacher, et en second paramètre une méthode de BCrypt (genSalt) qui permet de générer du sel.

Lorsque l'utilisateur se connecte :

```
public boolean connexion(String login, String mdp) throws SQLException {

                stmt = conn.createStatement();

                ResultSet rs = stmt.executeQuery("SELECT USERS.MDP FROM USERS WHERE login =
'"+login+"'");

                while (rs.next()) {

                        if(BCrypt.checkpw(mdp, rs.getString("mdp")) == true){
                                return true;
                        }
                }

                return false;
```

```
}
```

Voici le bout de code permettant de vérifier si le mot de passe et le login entré par l'utilisateur est correct.

Cette méthode est dans la classe données et prend en paramètre le login et le MDP entré dans le formulaire de connexion Login.

Une requête SQL est exécutée pour récupérer le mot de passe haché correspondant à l'utilisateur donné en paramètre.

Puis la méthode ***checkpw*** de **Bcrypt** est appelée. Cette méthode prend 2 paramètres : Un string classique et un String qui est le résultat d'un hachage effectué précédemment.
Si cette méthode renvoie **true**, cela veut dire que le mot de passe entré par l'utilisateur et le mot de passe haché récupéré depuis la base de données correspondant au login entré par l'utilisateur est correct.
La méthode renvoie **true** au **contrôleur** et affiche l'application via l'instanciation d'un nouvel objet ConsoleGUI.