Java Server Faces

M2I Formations 2022

Samih Habbani

Objectifs de la formation

- Mettre en place un environnement de développement
- Démarrer avec JSF
- Lire les valeurs des champs d'un formulaire
- Découverte des différents composants UI JSF
- Gérer la validation des formulaires HTML
- Implémenter la logique business dans nos Bean managés
- Afficher des données en utilisant les listes et les tables
- Construire une web app avec JDBC et effectuer un CRUD

Les points de la formation

- Approche des différents composants JSF
- Lier un formulaire JSF à un Bean managé
- Valider et contrôler les données d'un formulaire
- Appeler les méthodes d'un Bean managé depuis une page JSF
- Utilisation des listes et des tables
- CRUD JDBC avec nos pages JSF

Pré-requis pour ce cours

- Avoir des connaissances en JAVA
- Avoir une approche MVC
- Avoir des connaissances en POO
- Avoir des connaissances en HTML

A qui est destiné ce cours

- Développeurs full-stack

Java Server Faces

Créer des application web complètes avec des UI performantes

Java Server Faces c’est quoi?

- Framework pour créer des applications web
- Standard pour créer des applications d’entreprise
- Populaire
- Basé sur le modèle MVC

Différentes versions

- JSF 1 : 2004 J2EE 1.4
- JSF 1.2 : 2006 J2EE 5
- JSF 2.0 : 2009 J2EE 6
- JSF 2.2 : 2013 J2EE 7
- JSF 2.3 : 2017 : Java 8

Java Server Faces - les avantages

- Façon standard de coder des applications Java
- Permet de réutiliser de nombreux composants
- Permet de gérer l'état de l'application pour les requêtes web
- Mise à disposition d'un processus de validation, de conversion des données d'un formulaire

Structure d’une page JSF

- Extension xhtml
- Une page HTML ‘presque’ normal avec des balises JSF
- On peut mélanger le html avec les balises JSF
- On peut ajouter d’autres librairies tierces (jQuery…)

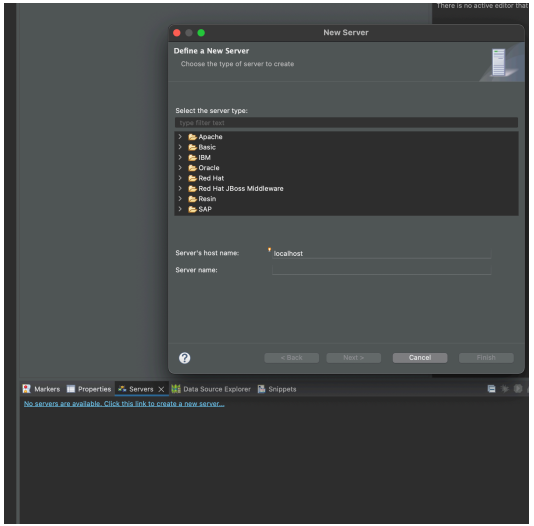
1.

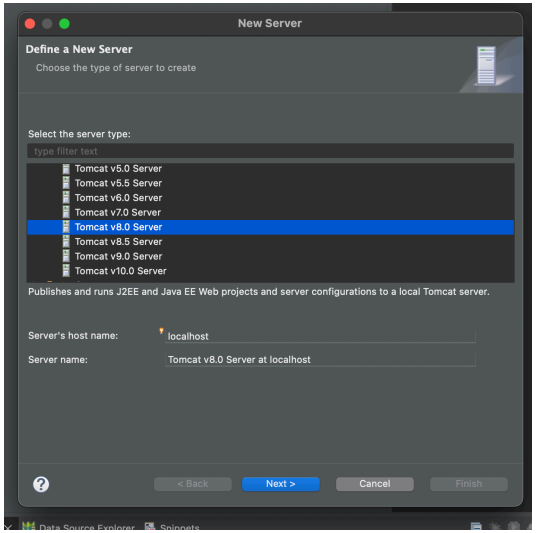
Installation de l’environnement

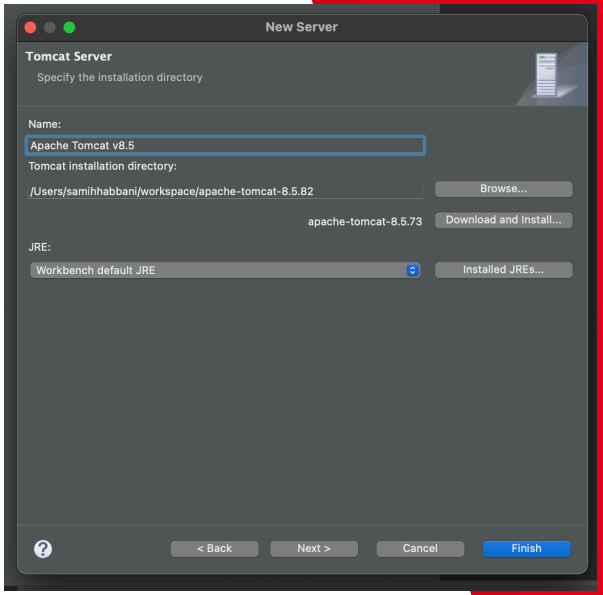
Installations

- JAVA 17
- Tomcat 8-9
- Eclipse pour développeurs web

Connecter Eclipse à Tomcat



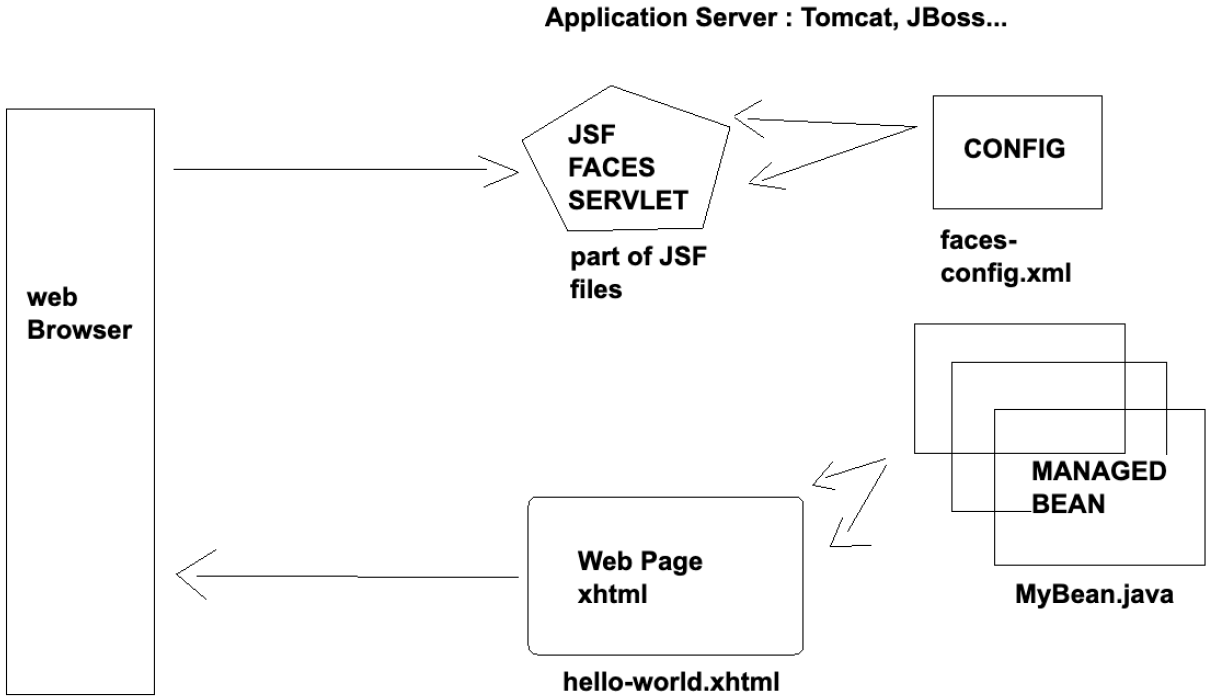




2.

JSF - Gestion des formulaires

Comment fonctionne JSF



Nouveau projet

- New Project
- Dynamic Web Project
- Ajouter JavaServerFaces
- Générer web.xml

Télécharger la librairie JSF

- https://maven.java.net/content/repositories/releases/org/glassfish/javax.faces/2.2.8/
- javax.faces-2.2.8.jar
- A ajouter dans WEB-INF/lib

Les composants UI JSF

Tag	Functions	Rendered As	Appearance
h:inputText	It allows a user to input a string.	An HTML <input type="text"> element	A field
h:outputText	It displays a line of text.	Plain text	Plain text
h:form	It represents an input form.	An HTML <form> element	No appearance
h:commandButton	It submits a form to the application.	An HTML <input type=value> element for which the type value can be "submit", "reset", or "image"	A button
h:inputSecret	It allows a user to input a string without the actual string appearing in the field.	An HTML <input type="password"> element	A field that displays a row of characters instead of the actual string entered.
h:inputTextarea	It allows a user to enter a multiline string.	An HTML <textarea> element	A multirow field
h:commandLink	It links to another page or location on a page.	An HTML <a href> element	A link
h:inputSecret	It allows a user to input a string without the actual string appearing in the field.	An HTML <input type="password"> element	A field that displays a row of characters instead of the actual string entered.
h:inputHidden	It allows a page author to include a hidden variable in a page.	An HTML <input type="hidden"> element	No appearance

UI COMPONENT ET NAMESPACES

CORE COMPONENTS
http://xmlns.jcp.org/jsf.core

HTML COMPONENTS
http://xmlns.jcp.org/jsf/html

Facelets COMPONENTS
http://xmlns.jcp.org/jsf/facelets

Composite COMPONENTS
http://xmlns.jcp.org/jsf/composite

CORE COMPONENTS

- <f:selectItem itemValue="C#" itemLabel="C#"></f:selectItem>

HTML COMPONENTS

```HTML
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Titre de la page</title>
  <link rel="stylesheet" href="style.css">
  <script src="script.js"></script>
</head>
<body>
  ...
  <!-- Le reste du contenu -->
  ...
</body>
</html>
```

FACELETS COMPONENTS

- Facelets est un langage déclaratif qui dérive de HTML et XML utilisé pour écrire des pages web XHTML par JSF; en plus d'autres librairies qu'il peut utiliser, le langage de balises Facelets utilise, sa propre librairie de tag, la librairie de tags JSTL et le langage d'expression EL.

COMPOSITE COMPONENTS

- Un composant composé d'un ensemble de balises et autres composants. Il est ré-utilisable et créer par l'utilisateur et permet d'être entièrement customisé. Il définit des fonctionnalisés et peut avoir des validateurs, des converters et des listeners tout comme n'importe quel autre composant JSF.

MANAGED BEAN

- Constructeur sans argument
- Mise en place d'attributs avec getters et setters
- @ManagedBean
- C'est une classe Java classique
- Sert à contenir des données
- Peut aussi contenir de la logique business
- Créer et managé par JSF
- NE pas confondre avec les EJB (Java Beans)

Il exécute la logique métier, gère la navigation entre les pages ainsi que l'état les données d'une page. Une application JSF typique contient un ou plusieurs beans managé – Plusieurs pages peuvent partager le même bean managé.

JSF Expression Language

- Expression Language (EL) est un langage de script permettant l'accès à des composants Java (les JavaBeans) à travers des JSP ou JSF.

STRUCTURE D'UNE PAGE JSF

student_table_demo.xhtml

WebContent > student_table_demo.xhtml > html

```
1   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4   <html
5
6       xmlns="http://www.w3.org/1999/xhtml"
7       xmlns:h="http://xmlns.jcp.org/jsf/html"
8       xmlns:f="http://xmlns.jcp.org/jsf/core">
9
10      <h:head>
11          <title>Hello World - Input Form</title>
12          <h:outputStylesheet library="css" name="style.css"/>
13      </h:head>
14
15      <h:body>
16
17      </h:body>
18
19
20  </html>
21
```

JSF FORMS

```
<h:form id="user-form">
    <h:outputLabel for="username">User Name</h:outputLabel>
    <h:inputText id="username" value="#{user.name}" required="true" requiredMessage="Username is required"/><br/>
    <h:commandButton id="submit-button" value="Submit" action="response.xhtml"/>
</h:form>
```

JSF DROP DOWN LIST

```
<h:selectOneMenu value="#{user.favCoffee1}">
    <f:selectItem itemValue="Cream Latte" itemLabel="Coffee3 - Cream Latte" />
    <f:selectItem itemValue="Extreme Mocha" itemLabel="Coffee3 - Extreme Mocha" />
    <f:selectItem itemValue="Buena Vista" itemLabel="Coffee3 - Buena Vista" />
</h:selectOneMenu>
```

JSF RADIO BUTTON

```
<h:selectOneRadio value="#{user.favColor1}">
    <f:selectItem itemValue="Red" itemLabel="Color1 - Red" />
    <f:selectItem itemValue="Green" itemLabel="Color1 - Green" />
    <f:selectItem itemValue="Blue" itemLabel="Color1 - Blue" />
</h:selectOneRadio>
```

JSF CHECKBOXES

```
<h:selectManyCheckbox value="#{user.favNumber1}">
    <f:selectItem itemValue="1" itemLabel="Number1 - 1" />
    <f:selectItem itemValue="2" itemLabel="Number1 - 2" />
    <f:selectItem itemValue="3" itemLabel="Number1 - 3" />
</h:selectManyCheckbox>
```

PRE-REMPLIR LES CHAMPS D'UN FORMULAIRE

```java
@ManagedBean
public class StudentFour {


    private String firstName;
    private String lastName;
    private String city;
    private List<String> cities;
    private String favoriteLanguage;
    private String[] favoriteLanguages;

    public StudentFour() {

        cities = new ArrayList<>();
        cities.add("Casablanca");
        cities.add("Rabat");
        cities.add("Fes");
        cities.add("Berkane");

        firstName = "Samih";
        lastName = "Habbani";

    }
```

3.

Validation des données

AFFICHER LES MESSAGES D'ERREURS

```
<h:form>

<!-- Display Error messages -->
<h:messages styleClass="error" />

    <!-- Setter mes valeurs -->
    FirstName :<h:inputText id="firstName" value="#{studentOne.firstName}" label="FirstName" />
    LastName :<h:inputText id="lastName" value="#{studentOne.lastName}" label="LastName" required="true" />
    Email :<h:inputText id="email" value="#{studentOne.email}" label="Email" required="true" />

    <h:commandButton value="Enregistrer" action="student_confirmation_one"/>

</h:form>
```

REQUIRED

```
<h:form>

<!-- Display Error messages -->
<h:messages styleClass="error" />

    <!-- Setter mes valeurs -->
    FirstName :<h:inputText id="firstName" value="#{studentOne.firstName}" label="FirstName" />
    LastName :<h:inputText id="lastName" value="#{studentOne.lastName}" label="LastName" required="true" />
    Email :<h:inputText id="email" value="#{studentOne.email}" label="Email" required="true" />

    <h:commandButton value="Enregistrer" action="student_confirmation_one"/>

</h:form>
```

CUSTOMISER UN MESSAGE

```
<h:form>

    <!--  Enlever les msg qui étaient ici -->

    <!-- Setter mes valeurs -->
    FirstName :
    <h:inputText id="firstName"
    value="#{studentOne.firstName}"
    label="FirstName" />

    <br/>
    LastName :
    <h:inputText id="lastName"
        value="#{studentOne.lastName}"
        label="LastName"
        required="true"
        requiredMessage="Votre nom est obligatoire"/>
    <h:message for="lastName" styleClass="error"/>

    <br/>
    Email :
    <h:inputText id="email"
    value="#{studentOne.email}"
    label="Email"
    required="true"
    requiredMessage="Votre email est obligatoire"/>
    <h:message for="email" styleClass="error"/>

    <br/>
    <h:commandButton value="Enregistrer" action="student_confirmation_one"/>

</h:form>
```

VALIDATE NUMBER RANGE

```
<br/>
Age :
<h:inputText id="age"
value="#{studentTwo.age}"
label="Age"
validatorMessage="Veuillez insérer un age compris entre 1 et 100 ans"
>

    <f:validateLongRange minimum="1" maximum="100"/>

</h:inputText>
```

VALIDATE LENGTH

```
<br/>
Code postal :
<h:inputText id="postalCode"
value="#{studentTwo.postalCode}"
label="Code postal"
validatorMessage="Veuillez insérer un code postal avec 5 caractères"
>

    <f:validateLength minimum="5" maximum="5"/>

</h:inputText>

<h:message for="postalCode" styleClass="error"/>
```

VALIDATE REGEX

```
<br/>
Phone number
<h:inputText id="phone"
value="#{studentThree.phone}"
label="Phone Number"
validatorMessage="Votre numéro de téléphone doit suivre le format suivant : xx-xx-xx-xx-xx"
>

    <f:validateRegex pattern="\d{2}-\d{2}-\d{2}-\d{2}-\d{2}"/>

</h:inputText>

<h:message for="phone" styleClass="error"/>
```

CUSTOM VALIDATION

```
<br/>
Course Code
<h:inputText id="courseCode"
value="#{studentFour.courseCode}"
label="CourseCode"
validator="#{studentFour.validateTheCourseCode}"/>


<h:message for="courseCode" styleClass="error"/>
```

```
public void validateTheCourseCode(FacesContext context,
        UIComponent component,
        Object value) throws ValidatorException {

    if (value == null) {
        return;
    }

    String data = value.toString();

    if(!data.startsWith("ACADEMIE_WS_")) {

        // FacesMessage represents a single validation (or other) message,
        // which is typically associated with a particular component in the view.
        FacesMessage message = new FacesMessage("Votre cours ne commence pas par ACADEMIE_WS_");
        throw new ValidatorException(message);
    }

}
```

4.

Implémentation de la logique business

APPELER LES MÉTHODES D'UN MANAGED BEAN

```
<h:form>

    Select a tour:
    <h:selectOneMenu value="#{tourBean.kindOfTour}">

        <f:selectItem itemLabel="City" itemValue="city"/>
        <f:selectItem itemLabel="Country" itemValue="country"/>


    </h:selectOneMenu>

    <h:commandButton value="Submit" action="#{tourBean.startTheTour()}" />

</h:form>
```

```
@ManagedBean
public class TourBean {

    private String kindOfTour;

    public TourBean() {

    }

    public String getKindOfTour() {
        return kindOfTour;
    }

    public void setKindOfTour(String kindOfTour) {
        this.kindOfTour = kindOfTour;
    }

    public String startTheTour() {

        System.out.println(kindOfTour);

        if(kindOfTour != null && kindOfTour.equals("city")) {

            return "city_tour";

        } else {

            return "country_tour";
        }

    }

}
```

LES DIFFÉRENTS SCOPES

- @SessionScoped
- @RequestScoped
- @ApplicationScoped

5.

Afficher des données en utilisant les tables et les listes

JSF UI LISTS

```
<h:body>

    <h3>Student List Demo</h3>

    <ul>
        <ui:repeat var="student" value="#{studentData.students}">
            <li>#{student.firstName} #{student.lastName}</li>
        </ui:repeat>
    </ul>

</h:body>
```

```
@ManagedBean
@ApplicationScoped
public class StudentData {

    private List<Student> students;

    public StudentData() {
        loadSampleData();
    }

    public void loadSampleData() {

        students = new ArrayList<>();
        students.add(new Student("Samih", "Habbani", "contact@mail.com"));
        students.add(new Student("Tom", "Ford", "contact21@mail.com"));
        students.add(new Student("Nicolas", "Valontino", "team@mail.com"));
    }

    public List<Student> getStudents() {
        return students;
    }
}
```

JSF UI TABLE

```xml
<h:body>

    <h3>Student Table Demo</h3>

    <h:dataTable value="#{studentData.students}" var="student" border="1"
        styleClass="demo-table"
        headerClass="demo-table-header"
        rowClasses="demo-table-odd-row, demo-table-even-row"
    >

        <h:column>
            <f:facet name="header">First Name</f:facet>
            #{student.firstName}
        </h:column>

        <h:column>
            <f:facet name="header">Last Name</f:facet>
            #{student.lastName}
        </h:column>

    </h:dataTable>

</h:body>
```

```java
@ManagedBean
@ApplicationScoped
public class StudentData {

    private List<Student> students;

    public StudentData() {
        loadSampleData();
    }

    public void loadSampleData() {

        students = new ArrayList<>();
        students.add(new Student("Samih", "Habbani", "contact@mail.com"));
        students.add(new Student("Tom", "Ford", "contact21@mail.com"));
        students.add(new Student("Nicolas", "Valontino", "team@mail.com"));
    }

    public List<Student> getStudents() {
        return students;
    }
}
```

TP - CRUD

Créer une application connectée à une BDD avec JDBC

INSTALLATIONS

- Installer MySQL Workbench

TÉLÉCHARGEMENT

- JDBC DRIVER
- A placer dans WEB-INF/LIB

DÉFINIR UN POOL DE CONNECTION

- context.xml
- web.xml

RÉCUPÉRER LE POOL DE CONNECTION

- Injection avec les servlets
- Java Naming Directory Interface (JNDI)

INJECTION AVEC SERVLETS

```java
@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {

    @Resource(name = "jdbc/student_tracker")
    private DataSource dataSource;

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        response.setContentType("text/plain");


        // TESTER LA CONNECTION ICI

        Connection connection = null;
        Statement stmt = null;
        ResultSet res = null;

        try {

            connection = dataSource.getConnection();
            String sql = "select * from student;";
            stmt = connection.createStatement();

            res = stmt.executeQuery(sql);

            while(res.next()) {
                String email = res.getString("email");
                out.print(email); // affiche sur la page web
                System.out.println(email); // affiche en console
            }


        } catch(Exception e) {
            e.printStackTrace();
            out.print(e.getMessage());
        }
```

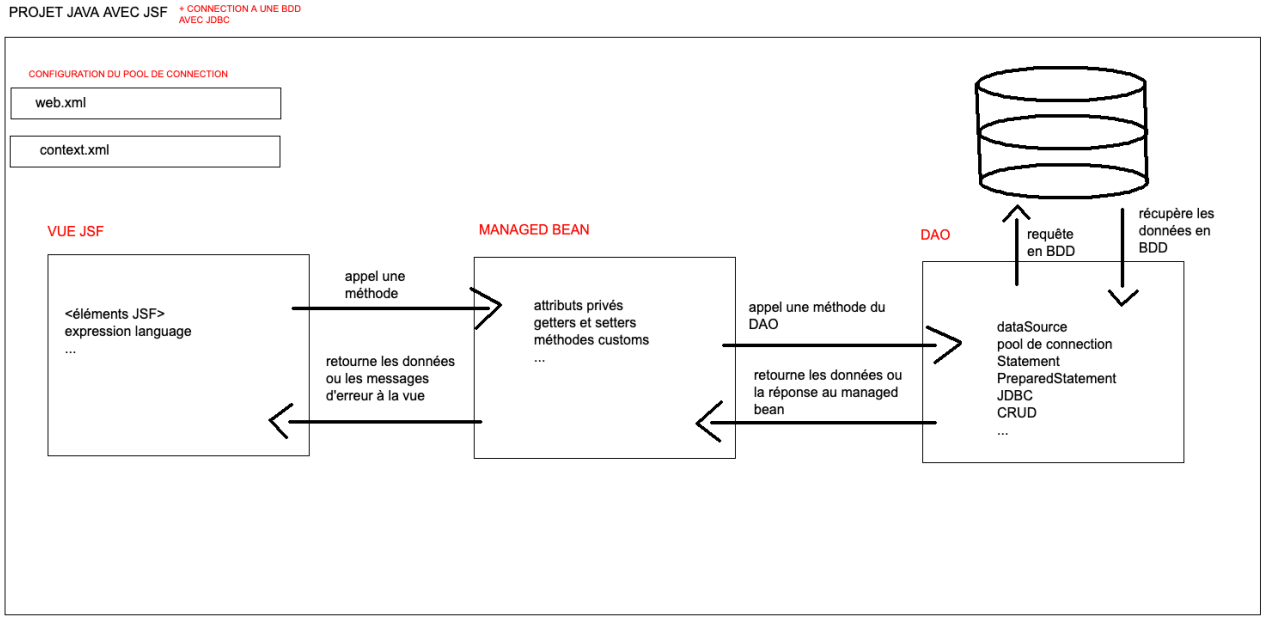
Java Naming Directory Interface (JNDI)

```java
public class StudentDbUtil {

    private static StudentDbUtil instance;
    private DataSource dataSource;
    private String jndiName = "java:comp/env/jdbc/student_tracker";

    public static StudentDbUtil getInstance() throws Exception {
        if (instance == null) {
            instance = new StudentDbUtil();
        }

        return instance;
    }

    private StudentDbUtil() throws Exception {
        dataSource = getDataSource();
    }

    private DataSource getDataSource() throws NamingException {
        Context context = new InitialContext();

        DataSource theDataSource = (DataSource) context.lookup(jndiName);

        return theDataSource;
    }


    // CI DESSOUS LES DIFFÉRENTES REQUÊTES À EFFECTUER


    // SELECT

    private Connection getConnection() throws SQLException {

        Connection myConnection = dataSource.getConnection();

        return myConnection;
    }
```

DESIGN PATTERNS

- Vue JSF
- Managed Bean
- Data Access Object

SCHÉMA EXPLICATIF



THE END.

Avec tous nos remerciements et toutes nos félicitations pour avoir suivi ce cursus.

Samih Habbani : s.habbani@coderbase.io