République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Benyoucef Benkhedda – Alger 1
Faculté des Sciences
Département de Mathématiques et Informatique
Master 1 Ingénierie des Systèmes Informatiques Intelligents (ISII)

Architecture Logiciel

Projet TP

Élaboré par :

Mlle Hanafi Zohra Groupe 02
Mlle Tesbia Lylia Groupe 02

Enseignante :

Mme k.Guesmia

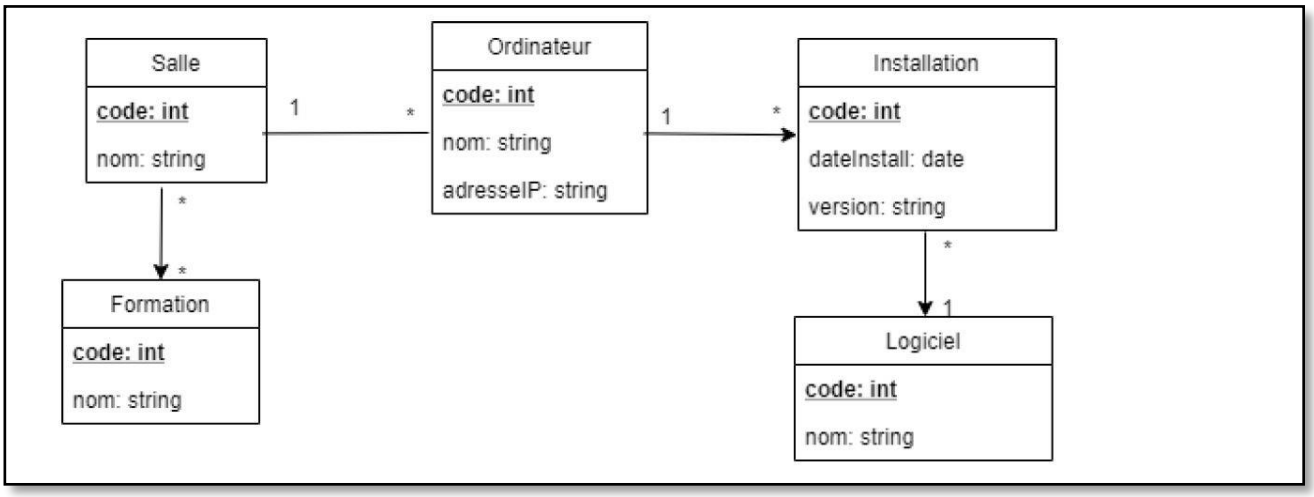
Année universitaire
2019 – 2020

Table des matières

I. Intoduction

On souhaite développer une application web JEE qui permet de gérer un parc informatique, ceci en utilisant **JSF**, **JPA, Maven,** et **H2**.

La figure suivante décrit le digramme de classes proposé.



II. Définitions des composantes :

Commençant par définir nos composants :

JSF :

Java Server Faces (JSF) est une technologie dont le but est de proposer un framework qui facilite et standardise le développement d'applications web avec Java.

JSF est un Framework orienté composants.

Son développement a tenu en compte des différentes expériences acquises lors de l'utilisation des technologies standards pour le développement d'applications web.

L'objectif de JSF est de :

- Fournir un standard JEE spécifié dans une JSR pour le développement des IHM web riches
- Maximiser la productivité des applications web
- Fournir des fonctionnalités récurrentes et avancées (Validations, Conversion, Ajax …)
- Masquer la complexité

JPA :

Java Persistance Api (JPA) est une interface de programmation Java permettant de normaliser l'utilisation et la communication avec la couche de données, d'une application Java, la gestion des données relationnelles, manipulation des données et de leurs relations. C'est une spécification Java EE basé sur le concept ORM

L'utilisation de JPA nécessite un fournisseur de persistance (ORM) qui implémente les spécifications JPA.

Il y a plusieurs fournisseurs de persistance (différentes mises en œuvre) qui implémentent la spécification JPA, tel que Hibernate, TopLink, EclipseLink,Apache Open PA...

Dans ce projet on va utiliser « Hibernate ».

Maven :

Maven est un outil de construction de projets (build) open source développée par la fondation Apache, initialement pour les besoins du projet Jakarta Turbine.

Il permet notamment :

- D'automatiser certaines tâches : compilation ,tests unitaires et déploiement des applications qui composent le projet
- De gérer des dépendances vis-à-vis des bibliothèques nécessaires au projet
- De générer des documentations concernant le projet

H2 :

Est un système de gestion de base de données relationnelles écrit en Java. Il peut être intégré à une application Java ou bien fonctionner en mode client-serveur[3].

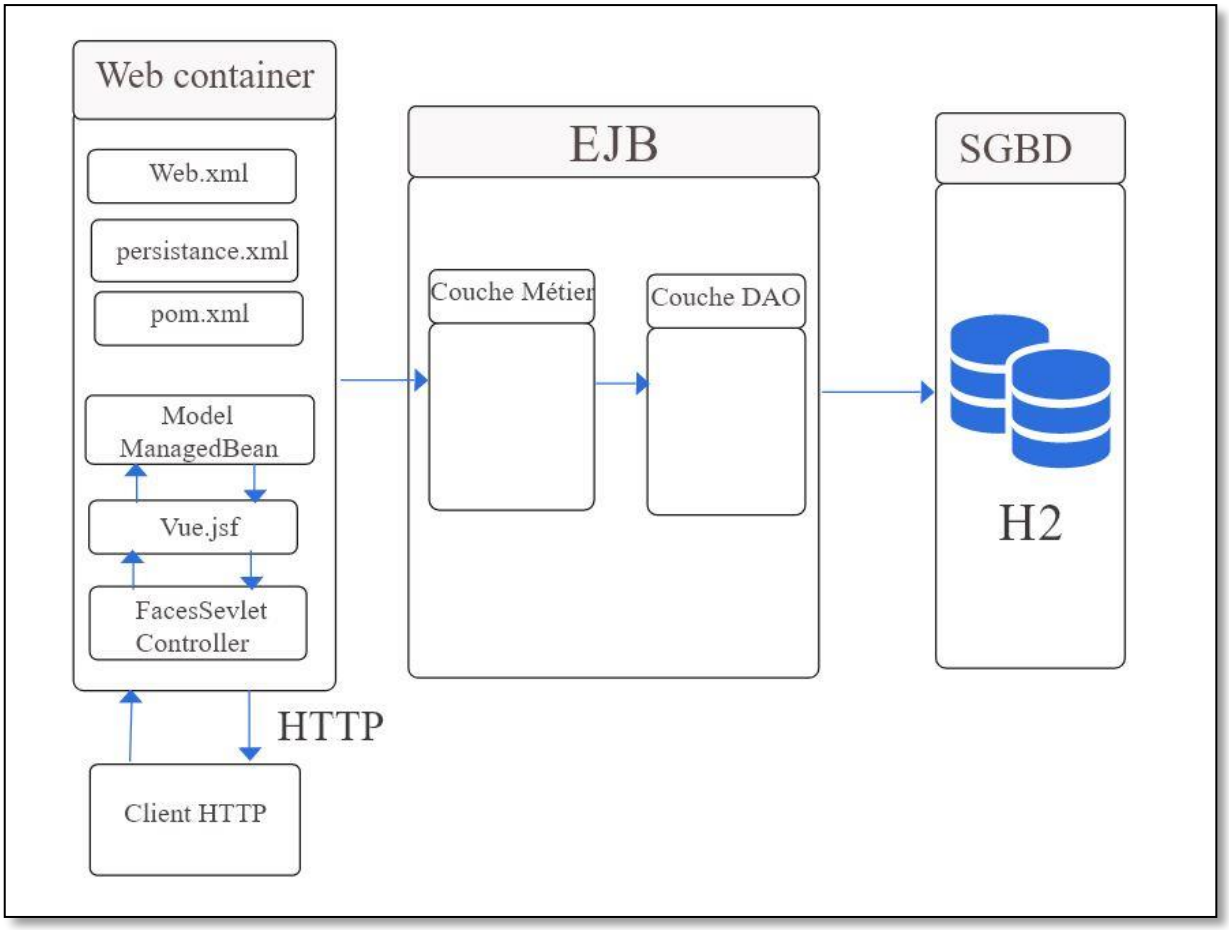
Son fichier jar est petit : environ 1 Mo[4].

III. Architecture

Cette application se compose de trois couches DAO, Métier et Présentation.

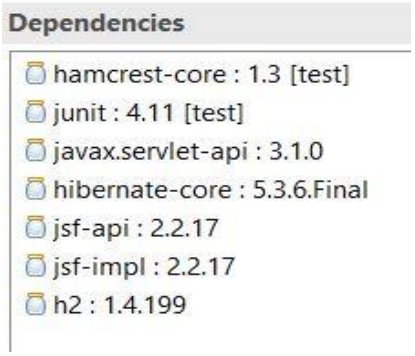
Elle doit être fermée à la modification et ouverte à l'extension.

La figure suivante décrit l'architecture qu'on a utilisée :



On a commencé par la création d'un projet Maven et ajouté les dépendances nécessaire dans le fichier « pom.xml » depuis ce lien
https://mvnrepository.com/?fbclid=IwAR0C3uKee3Gf3wyQyMAFdpXJKCRx9cAyPbRlCv-irOv05d4yxNZQ-1qTDCY

Ensuite on a les installer par un clic droit sur le projet → Run As → Maven Install.




```
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-core</artifactId>
    <version>1.3</version>
    <scope>test</scope>
</dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.3.6.Final</version>
    </dependency>

     <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-api</artifactId>
        <version>2.2.17</version>
    </dependency>

    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>2.2.17</version>
    </dependency>
<
```

De plus, on a créé trois packages dans notre source java qui sont : Model, DAO et Contrôleur afin de respecter l'architecture MVC.

- Modelé : pour la création des fonctions qui traitent les entités.
- DOA : Récupère les entités pour effectuer les traitements et la manipulation des donnes.
- Contrôleur : C'est la couche chargée de router les informations, elle va décider qui va récupérer l'information et la traiter.

1. Partie modèle :

On créer nos **modelés** qui sont : Formation, Installation, Ordinateur, Logiciel et Salle. Ensuite on manipule les associations entre eux.

➢ Table Installation :

On commence par créer les attributs de cette dernière, ensuite, on ajoute les getters/setters.

```java
@Entity
@Table(name="installation")
public class Installation implements Serializable
{
    @Id
    @Column
    private int code_installation;
    @Column
    @Temporal(TemporalType.DATE)
    private Date dateinstall;
    @Column
    private String version;
    public int getCode_installation() {
        return code_installation;
    }
    public void setCode_installation(int code_installation) {
        this.code_installation = code_installation;
    }
    public Date getDateinstall() {
        return dateinstall;
    }
    public void setDateinstall(Date dateinstall) {
        this.dateinstall = dateinstall;
    }
    public String getVersion() {
        return version;
    }
    public void setVersion(String version) {
        this.version = version;
    }
    @Override
```

Et comme la table Installation possède un logiciel on a ajouté l'annotation `@ManyToOne.`

```java
    @ManyToOne
    @JoinColumn(name="code_logiciel", referencedColumnName="code_logiciel",insertable=false, updatable=false)
    private Logiciel logiciel;

    public int getCode_logiciel() {
        return code_logiciel;
    }
    public void setCode_logiciel(int code_logiciel) {
        this.code_logiciel = code_logiciel;
    }
    public Logiciel getLogiciel() {
        return logiciel;
    }
    public void setLogiciel(Logiciel logiciel) {
        this.logiciel = logiciel;
    }
```

➢ Table Logiciel :

Comme la table Logiciel n'est en relation qu'avec la table Installation en unidirectionnel, aucune annotation de relation sera introduite à cette dernière.

```java
@Entity
@Table(name="logiciel")
public class Logiciel implements Serializable
{
    @Id
    @Column
    private int code_logiciel;


    public int getCode_logiciel() {
        return code_logiciel;
    }

    public void setCode_logiciel(int code_logiciel) {
        this.code_logiciel = code_logiciel;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    @Column
    private String nom;
```

➢ Table Ordinateur :

D'après le diagramme de classe on voit bien que cette table est en relation avec la table Installation en 1 : N en unidirectionnel et en N :1 en bidirectionnel avec la table Salle.

```java
@ManyToOne
@JoinColumn(name="code_salle", referencedColumnName="code_salle",insertable=false, updatable=false)
private Salle salle;
public int getCode_salle() {
    return code_salle;
}
public void setCode_salle(int code_salle) {
    this.code_salle = code_salle;
}
public Salle getSalle() {
    return salle;
}
public void setSalle(Salle salle) {
    this.salle = salle;
}
```

```java
@Entity
@Table
public class Ordinateur implements Serializable
{
    @Id
//  @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int code;
    @Column
    private String nom;
    @Column
    private String adresseIP;
    @Column
    public int getCode() {
        return code;
    }
    public void setCode(int code) {
        this.code = code;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getAdresseIP() {
        return adresseIP;
    }
    public void setAdresseIP(String adresseIP) {
        this.adresseIP = adresseIP;
    }
    @Override
```

- **Table Formation :**

Comme on n'a pas d'information sur la table formation avec salle donc cette table n'aura pas d'association, elle a des attributs privés, getters et setter et un constructeur sans et avec paramètre.

```java
@Entity
@Table(name="formation")
public class Formation implements Serializable
{

    @Id
    @Column
    private int code_formation;
    @Column
    private String nom;
    public int getCode_formation() {
        return code_formation;
    }


    public void setCode_formation(int code_formation) {
        this.code_formation = code_formation;
    }


    public String getNom() {
        return nom;
    }


    public void setNom(String nom) {
        this.nom = nom;
    }
```

➢ Table Salle :

Dans une salle, on a plusieurs ordinateurs et plusieurs formations. Donc on ajoute une association de 1 : N avec Ordinateur en bidirectionnel et N :N avec Formation en unidirectionnel.

Dans le cas de N :N en unidirectionnel on créer une nouvelle table d'association qui possède les deux clés primaire de formation et salle.

Le code de la table est le suivant :

```java
@Id
private int code_salle;

@Column
private String nom;

@Column
public int getCODE_SALLE() {
    return code_salle;
}

public void setCODE_SALLE(int code_salle) {
    this.code_salle = code_salle;
}

public String getNom() {
    return nom;
}

public void setNom(String nom) {
    this.nom = nom;
}


@OneToMany(targetEntity=Ordinateur.class,mappedBy="salle")
private List<Ordinateur> ordinateurs = new ArrayList<>();


@ManyToMany(cascade = {CascadeType.ALL})
@JoinTable( name = "salle_formation",
            joinColumns = @JoinColumn( name = "code_salle" ),
            inverseJoinColumns = @JoinColumn( name = "code_formation" ) )
private List<Formation> formations = new ArrayList<>();

public List<Formation> getFormations() {
    return formations;
}
```

Entity Manager:

Pour assurer la persistance d'une entité dans la BD (CRUD), il est nécessaire d'invoquer une interface JPA spécifique, appelée EntityManager.

On va l'utiliser pour assurer la liaison entre notre base de données et Java et fournir les méthodes courantes d'accès aux données.

```java
public class JPAUtil {

    private static final String PERSISTENCE_UNIT_NAME = "PERSISTENCE";
    private static EntityManagerFactory factory;

    public static EntityManagerFactory getEntityManagerFactory() {
        if (factory==null) {
            factory=Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
        }
        return factory;
    }

    public static void shutdown() {
        if (factory!=null) {
            factory.close();
        }
    }

}
```

2. Partie DAO :

On commence par créer des interfaces qui contiennent les méthodes nécessaires pour la manipulation de nos données.

Voici un exemple d'une interface Installation :

```java
public interface IInstallationDAO
{
    public void sauvegarder(Installation installation);
    public void modifier(Installation installation);
    public Installation chercher(int code);
    public List<Installation> selectionner();
    public void supprimer(int code);
}
```

Ensuite on crée des classes afin d'implémenter les interfaces :

- **void sauvegarder(Installation installation)** : pour ajouter une installation.
- **void modifier(Installation installation) :** pour modifier une installation.
- **Installation chercher (int code)** : pour cheecher une installation en donnant son code.
- **List<Installation> selectionner():** pour récupérer la liste des installation qui existe dans notre bdd.
- **void supprimer(int code):** pour supprimer une installation en donnant son code.

```java
public class InstallationDAO implements IInstallationDAO {
    EntityManager entity = JPAUtil.getEntityManagerFactory().createEntityManager();
    @Override
    public void sauvegarder(Installation installation) {
        entity.getTransaction().begin();
        entity.persist(installation);
        entity.getTransaction().commit();

    }

    @Override
    public void modifier(Installation installation) {
        entity.getTransaction().begin();
        entity.merge(installation);
        entity.getTransaction().commit();

    }

    @Override
    public Installation chercher(int code) {
        Installation o = new Installation();
        o = entity.find(Installation.class, code);
        // JPAUtil.shutdown();
        return o;
    }

    @Override
    public List<Installation> selectionner() {
        List<Installation> listeInstallation = new ArrayList<>();
        Query query = entity.createQuery("select o from Installation o");
        listeInstallation = query.getResultList();
        return listeInstallation;
    }

    @Override
    public void supprimer(int code) {
        Installation installation = new Installation();
        installation = entity.find(Installation.class, code);
```

Dans cette partie on a donné un exemple d'entité « installation », on a suivi le même traitement pour les autres entités.

3. Partie Contrôleur :

C'est la couche chargée de router les informations, elle va décider qui va récupérer l'information et la traiter. Elle gère les requêtes des utilisateurs et retourne une réponse avec l'aide de la couche Modèle et Vue.

```java
@ManagedBean(name="installationBean")
@RequestScoped
public class InstallationBean
{

    public String nouveau() {
        Installation i = new Installation();
        Map<String, Object> sessionMap = FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
        sessionMap.put("installation", i);
        return  "/faces/creerInstallation.xhtml";
    }

    public String sauvegarder(Installation installation) {

        IInstallationDAO installationDAO= new InstallationDAO();
        installationDAO.sauvegarder(installation);
        return  "/faces/index.xhtml";
    }

    public List<Installation> selectionner()
    {

        IInstallationDAO  installationDAO = new InstallationDAO();

        return installationDAO.selectionner();
    }


    public String modifier(int code)
    {
            IInstallationDAO installationDAO = new InstallationDAO();
            Installation i = new Installation();
            i = installationDAO.chercher(code);
            System.out.println("****************************************");
            System.out.println(i);

            Map<String, Object> sessionMap = FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
```

IV. Fichier persistence :

Dans le répertoire src/main/java/META-INF, nous avons un fichier persistence.xml qui est un fichier de configuration standard de JPA et qui contient toutes les classes déclarées comme entité, et qui est chargé de la connexion avec la base de données.

```xml
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
            http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="PERSISTENCE">
    <description>Projet JEE</description>
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <class>com.ecodeup.model.Ordinateur</class>
    <class>com.ecodeup.model.Salle</class>
    <class>com.ecodeup.model.Formation</class>
    <class>com.ecodeup.model.Installation</class>
    <class>com.ecodeup.model.Logiciel</class>
    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/test" />
      <property name="javax.persistence.jdbc.user" value="sa" />
      <property name="javax.persistence.jdbc.password" value="" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
  </persistence-unit>

</persistence>
```

V. Interfaces :

✓ Présentation de l'interface d'accueil :

Entité Formation

NOUVELLE FORMATION

Code de la formation	Nom de la formation	Actualiser	Modifier	Supprimer
1	js	ACTUALISER	MODIFIER	SUPPRIMER
2	web application	ACTUALISER	MODIFIER	SUPPRIMER
3	anglais	ACTUALISER	MODIFIER	SUPPRIMER
4	langues	ACTUALISER	MODIFIER	SUPPRIMER

Entité Installation

NOUVELLE INSTALLATION

Code de l'installation	Date de l'installatiton	Version	Nom de l'ordinateur	Code du logiciel	Actualiser	Modifier	Supprimer
1	2020-10-20	2	asus	netbeans	ACTUALISER	MODIFIER	SUPPRIMER
2	2020-05-14	1	dell	android	ACTUALISER	MODIFIER	SUPPRIMER
3	2019-01-01	1	hp	ccleaner	ACTUALISER	MODIFIER	SUPPRIMER
4	2018-07-19	3	condor	kaspersky	ACTUALISER	MODIFIER	SUPPRIMER

Entité Logiciel

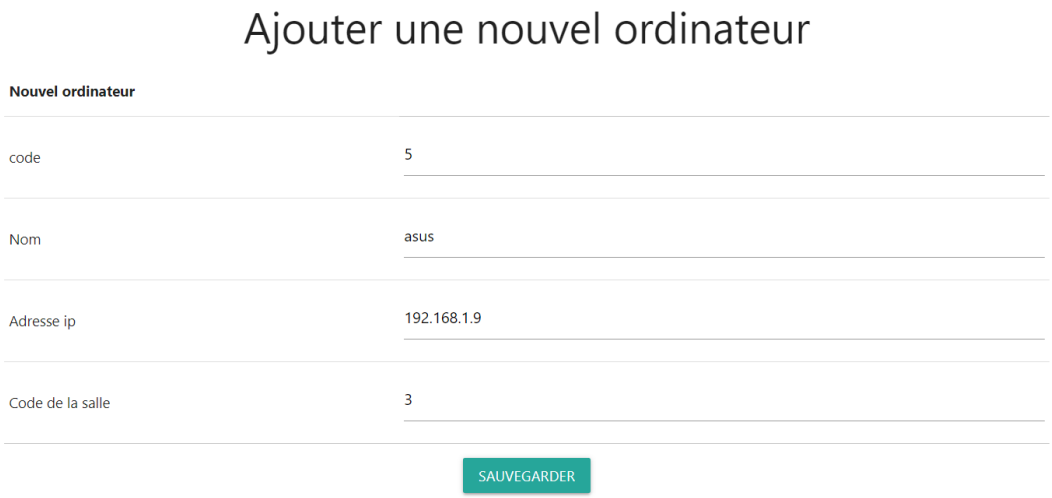
NOUVEAU LOGICIEL

Code du logiciel	Nom du logiciel	Actualiser	Modifier	Supprimer
1	android	ACTUALISER	MODIFIER	SUPPRIMER

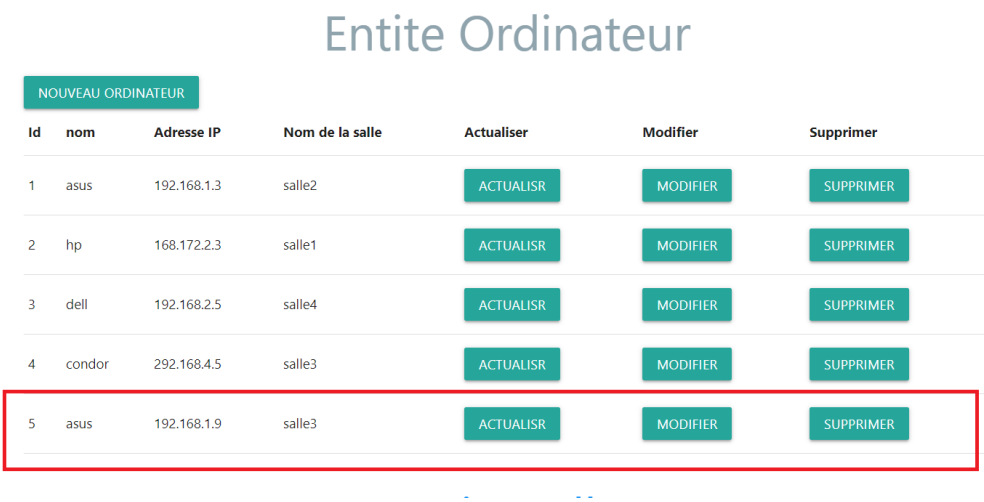
Pour chaque entité on peut visualiser ses enregistements (read) modifier une ou plusieurs lignes (update), supprimer une ou plusieurs lignes (delete) ou bien créer un nouvel enregistrement de donnée (create) et même actualiser toute la table.

✓ Ajouter un enregistrement :

Lorsqu'on clique sur le bouton nouvel ordinateur par exemple, on sera redirectionné vers une autre interface qui est présentée comme suit :



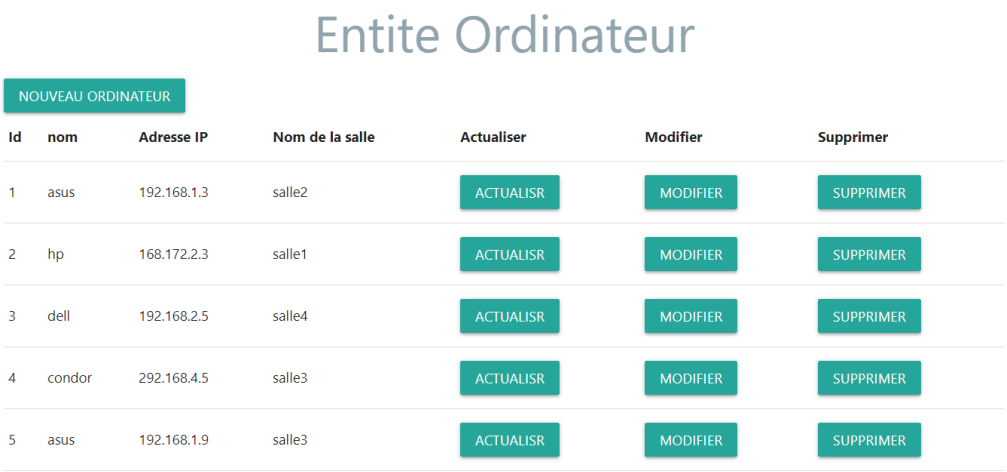
Dès qu'on clique sur le bouton sauvegardé on revient à l'interface d'accueil avec la nouvelle ligne créée.



✓ Supprimer un enregistrement :

On reste dans la table « ordinateur » par exemple.

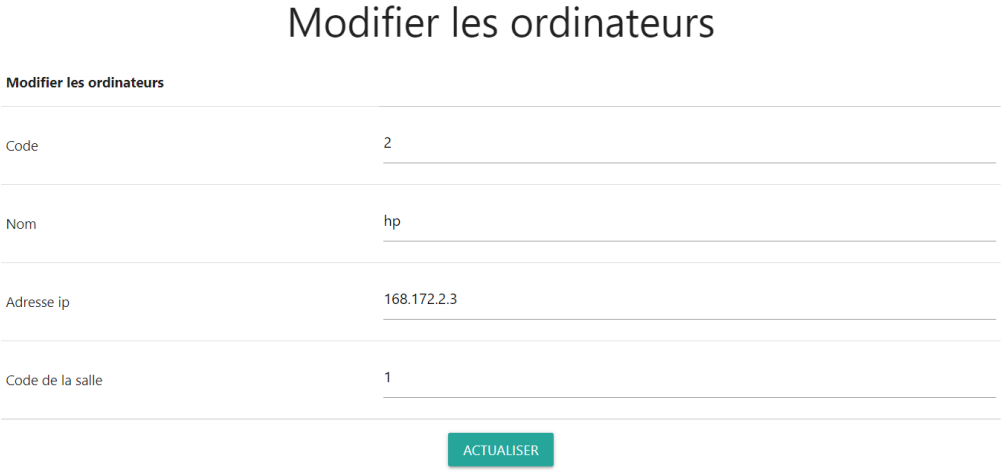
Avant la suppression :



Après la suppression :

✓ Modifier un enregistrement :

Toujours avec la table ordinateur : dès qu'on clique sur le bouton modifier sur n'importe quelle ligne, on sera redirectionné vers une autre interface qui contient des champs remplis des informations de l'enregistrement choisi.



Avant la modification :

Après la modification :

Entite Ordinateur

NOUVEAU ORDINATEUR

Id	nom	Adresse IP	Nom de la salle	Actualiser	Modifier	Supprimer
1	asus	192.168.1.3	salle2	ACTUALISR	MODIFIER	SUPPRIMER
2	hp	168.172.10.10	salle4	ACTUALISR	MODIFIER	SUPPRIMER
3	dell	192.168.2.5	salle4	ACTUALISR	MODIFIER	SUPPRIMER
4	condor	292.168.4.5	salle3	ACTUALISR	MODIFIER	SUPPRIMER