

PROJET RT0805

J'ai réalisé le projet sur une machine virtuelle Ubuntu 22.

Table des matières

I.	Projet principal : sujet 3.....	2
1.	Présentation du sujet	2
2.	JSP.....	2
a.	Connexion.....	2
b.	Création d'un utilisateur.....	3
c.	Home / accueil.....	3
d.	Groupe.....	4
e.	Inscription.....	5
f.	Administrateur.....	6
3.	Données.....	6
4.	Classes Java.....	8
a.	Exemple avec la classe Lieu_srv	8
b.	Intégration dans le projet	9
c.	Servlets	10
5.	Conclusion	10
II.	Environnement technologique.....	12
1.	Installations	12
a.	JDK	12
b.	Maven.....	12
c.	Tomcat	12
2.	Faire un projet sans dépendance en guise d'exemple.....	13
3.	Faire un projet avec au moins une dépendance en guise d'exemple.....	15
4.	Faire un projet d'exemple en Jakarta EE qui exploite JSP (ou JSF) ainsi que servlet	18
a.	Générer le projet	18
b.	API OpenWeather.....	18
c.	Gérer les dépendances.....	18
d.	Création du fichier JSP	19
e.	Création du Servlet	19
f.	Construction de l'application Web	21

I. Projet principal : sujet 3

1. Présentation du sujet

L'application vise à **faciliter la planification de trajets réguliers** en mettant en relation conducteurs et passagers. Elle offre une interface conviviale permettant aux utilisateurs de créer des profils, de se connecter et de spécifier leurs disponibilités et demandes de trajet via un calendrier interactif. Un rôle d'administrateur est également prévu pour gérer les informations et attribuer les passagers aux conducteurs. Pour la mise en œuvre, la partie serveur sera développée sur la plateforme Jakarta EE avec l'utilisation de Tomcat comme serveur d'application, ainsi que Maven et Git comme outils de gestion de projet.

Malheureusement, il convient de noter que je n'ai pas pu réaliser toutes les fonctionnalités que j'avais envisagées. Toutefois, j'ai réussi à établir les bases essentielles, telles que le traitement du XML, le renvoi dans le XML, la compréhension des JSP, des formulaires et des fonctions des servlets.

2. JSP

Dans cette section, nous procéderons à une analyse approfondie des pages JSP que j'avais initialement planifiées pour ce projet, ainsi que de leur rendu final si j'ai pu les créer ou les finaliser dans les délais impartis. Nous examinerons également le code Java associé à ces pages dans la partie I.4 de ce rapport.

a. Connexion



La page de connexion propose un formulaire demandant un pseudonyme et un mot de passe, ainsi qu'un bouton de connexion. Elle permet également d'accéder à une seconde page qui permet de créer un nouvel utilisateur.

FIGURE 1 - PAGE DE CONNEXION

La page de connexion est conçue avec deux boutons distincts dans le but de différencier les traitements et de rediriger vers les pages appropriées.

- Si le **mot de passe est correct**, l'utilisateur est redirigé vers la page d'accueil.
- Si le **mot de passe est incorrect**, l'utilisateur reste sur la page de connexion et un pop-up informatif apparaît.
- Enfin, si l'utilisateur souhaite **créer un nouvel** utilisateur, il est redirigé vers la page de création.

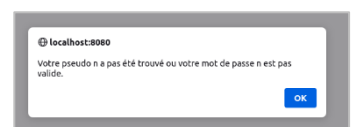
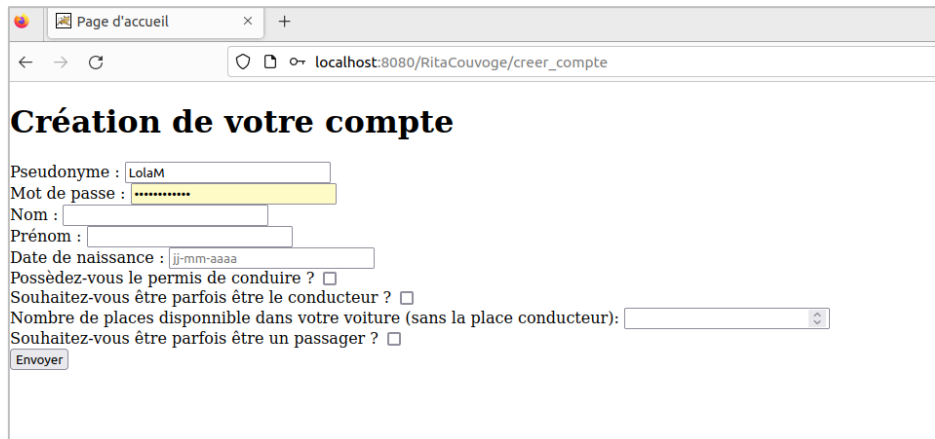


FIGURE 2 - POP-UP POUR MAUVAIS MOT DE PASSE

Pour différencier ces traitements, j'ai ajouté un attribut "name" au formulaire et récupéré sa valeur dans la servlet. Ainsi, en vérifiant si cet attribut est initialisé, nous pouvons déterminer quelle action doit être effectuée.

b. Création d'un utilisateur

La page de création d'utilisateurs comprend un formulaire permettant de saisir les informations personnelles telles que le nom, le prénom, le pseudo, la date de naissance, le permis de conduire, etc. Une fois ces données saisies, elles sont récupérées et utilisées pour créer un nouvel utilisateur, en ajoutant également d'autres informations récupérées par la servlet. En effet, un numéro d'utilisateur est attribué automatiquement, et une liste de groupes lui est associée (initialement vide). Cela permet de créer un profil complet pour le nouvel utilisateur, prêt à être utilisé dans le système.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/RitaCouvoge/creer_compte'. The page title is 'Page d'accueil'. The main heading is 'Création de votre compte'. The form contains the following fields and options:

- Pseudonyme :
- Mot de passe :
- Nom :
- Prénom :
- Date de naissance :
- Possédez-vous le permis de conduire ? ☐
- Souhaitez-vous être parfois être le conducteur ? ☐
- Nombre de places disponible dans votre voiture (sans la place conducteur):
- Souhaitez-vous être parfois être un passager ? ☐
-

FIGURE 3 - PAGE CREATION D'UN UTILISATEUR

c. Home / accueil

Lorsque l'utilisateur réussit à se connecter, son identifiant (créé lors de la création de l'utilisateur) est récupéré et ajouté en tant que paramètre de session. Cette démarche permet de conserver les informations de l'utilisateur tout au long de sa session, y compris ses données personnelles et ses groupes auxquels il appartient. La page d'accueil constitue un exemple concret de cette fonctionnalité, car elle affiche les statistiques de l'utilisateur concernant les trajets effectués, le prochain trajet prévu, les groupes auxquels il est affilié et s'il détient le statut d'administrateur au sein d'un groupe. Ainsi, l'utilisateur peut accéder rapidement à ses informations et visualiser les éléments pertinents dès son arrivée sur la page d'accueil.



The screenshot shows a web page with the following content:

- Menu**
 - [Accueil](#)
 - [Groupe](#)
 - [Page d'administration](#)
 - [Inscription](#)
- Bienvenue CaptainRita**
- Statistiques**
 - Vous avez été passager.e 0 fois.
 - Vous avez conduis 0 fois.
- Mes informations**
 - Prénom : Rita
 - Nom : Covoge
 - Date de naissance : 01-01-2000
 - Vous êtes inscrit.e comme passager.e.
 - Vous êtes inscrit.e comme driver.
- Mes groupes**
 - 11001
- Vous êtes administrateur.trice ...**
 - 11001

FIGURE 4 - PAGE HOME

d. Groupe

La page des groupes permet à l'utilisateur de rejoindre un groupe existant ou de créer un nouveau groupe en fournissant des informations pertinentes. Le paramètre "name" des champs du formulaire est également utilisé ici pour déterminer l'action que l'utilisateur souhaite effectuer, ce qui permet de le rediriger vers la page appropriée. Pour afficher tous les groupes auxquels l'utilisateur appartient, la servlet renvoie un tableau à la JSP, qui parcourt ensuite ce tableau pour l'afficher.

Il convient de noter que l'utilisation des fichiers JSP n'est pas considérée comme une meilleure pratique pour maintenir un code propre et maintenable à long terme, car la vue incorpore des éléments de logique métier qui peuvent devenir difficiles à maintenir. Cependant, du point de vue de l'utilisateur, cela fonctionne parfaitement et est simple à mettre en place. Il suffit de passer le vecteur contenant les données en tant que paramètre dans la requête GET de la servlet pour qu'il soit accessible dans la JSP.

Menu

- [Accueil](#)
- [Groupe](#)
- [Page d'administration](#)
- [Inscription](#)

Groupe

Liste des groupes auxquels vous appartenez :

Nom du groupe	Descriptif du groupe
Amigos	Amis pour la vie

Liste des groupes où vous êtes administrateur.trice :

Nom de votre groupe	Nombre de conducteur	Nombre de passager
Amigos	1	2

FIGURE 5 - PAGE GROUPE

Menu

- [Accueil](#)
- [Groupe](#)
- [Page d'administration](#)
- [Inscription](#)

Créer votre groupe

Nom de votre groupe :

Description de votre groupe :

FIGURE 6 - CREER UN GROUPE

Créer un groupe

Un autre JSP et une autre servlet permet, sur le même principe que pour la création d'un utilisateur de créer un groupe. Celui là va en plus créer un dossier par groupe dans chaque dossier semaine. Mais cela sera expliqué plus convenablement dans les parties suivantes.

Rejoindre un groupe

Cette page offre la possibilité aux utilisateurs d'envoyer une demande pour rejoindre un groupe en utilisant l'identifiant (ID) d'un groupe spécifique. Cela permet d'effectuer une vérification préalable pour les utilisateurs qui souhaitent conduire ou être passagers. Par exemple, certains parents pourraient ne pas être à l'aise de laisser leurs enfants avec des personnes inconnues dans un véhicule.

Cette fonctionnalité informe l'utilisateur si l'identifiant saisi ne correspond à aucun groupe existant, s'il appartient déjà au groupe correspondant à cet identifiant, ou s'il s'agit même de son propre groupe où il est administrateur.

Ces vérifications garantissent que les demandes de participation aux groupes sont appropriées et évitent les doublons ou les tentatives de rejoindre son propre groupe.

Rejoindre un groupe

Pour une raison de sécurité la liste des groupes n'est pas accessible. Vous pouvez rejoindre un groupe qu'en connaissance de son ID. Contactez l'administrateur du groupe pour récupérer cet ID.

Nom du groupe :

Vous êtes déjà administrateur.trice de ce groupe.

FIGURE 7 - REJOINDRE UN GROUPE

e. Inscription

La page d'inscription comporte trois parties.

Menu

- [Accueil](#)
- [Groupe](#)
- [Page d'administration](#)
- [Inscription](#)

Semaine affichée : 26

Groupe affiché : Amigos

Lundi 26 **Mardi 27** **Mercredi 28** **Jeudi 29** **Vendredi 30** **Samedi 1** **Dimanche 2**
to do to do to do to do to do to do to do

Choix de la semaine à afficher :

Afficher les rendez-vous existants du groupe :

Voici la liste des rendez-vous disponibles pour le groupe et la semaine sélectionnés :

1. Départ : Maison des Covoge. - Arrivée : Faculté des sciences de Bordeaux.
2. Départ : Faculté des sciences de Bordeaux. - Arrivée : Plage du Forge.

Créer un autre rendez-vous

FIGURE 8 - PAGE INSCRIPTION

La première est un calendrier permettant de sélectionner une semaine spécifique. Elle affiche la semaine en cours, par exemple la semaine du 28 juin est la semaine 26, et permet à l'utilisateur de choisir la semaine qu'il souhaite afficher. À terme, chaque jour devrait afficher les rendez-vous existants pour un groupe sélectionné (par défaut, le premier groupe de la liste de l'utilisateur ou le groupe qu'il a choisi). Cependant, cette fonctionnalité n'est pas encore implémentée. Pour le moment, la page permet d'afficher les véritables jours de la semaine choisie.

De plus, cette page permet d'afficher les rendez-vous existants, créés par d'autres utilisateurs, pour la semaine et le groupe sélectionnés. Cela offre la possibilité d'accéder à plus d'informations sur ces rendez-vous et de s'inscrire éventuellement.

Incription

Information du rendez-vous sélectionné :

Départ :
Ville : Bordeaux, 33000
Adresse : 9 Rue du Fort-Dauphin
Description : Maison des Covoge.
Heure : 00:00

Arrivée :
Ville : Bordeaux, 33000
Adresse : 146 rue Léo Saignat
Description : Faculté des sciences de Bordeaux.
Heure : 00:00

Inscription
Souhaitez-vous vous inscrire en tant que :

Quels jours ?
☐ Lundi ☐ Mardi ☒ Mercredi ☒ Jeudi ☐ Vendredi ☐ Samedi ☐ Dimanche

FIGURE 9 - PAGE INSCRIPTION QUAND ON CLIQUE

SUR LE BOUTON « CA M'INTERESSE »

Enfin, la dernière partie de cette page serait la création d'un nouveau rendez-vous. Malheureusement, je n'ai pas eu le temps de le développer, car il faudrait d'abord créer une page permettant de créer un nouveau lieu, puis une page pour créer le rendez-vous lui-même. Un rendez-vous consiste à choisir un lieu, une heure de départ et une heure d'arrivée. Cette fonctionnalité sera abordée plus en détail dans la partie de la base de données. Cependant, le fonctionnement reste similaire au reste du projet, car de nombreux liens et données entre les différents fichiers sont gérés avec la même logique, à l'aide de tables de hachage (hash map).

f. Administrateur

Enfin, la dernière page que j'ai eu le temps de mettre en place est la page administrative. Elle offre plusieurs fonctionnalités aux utilisateurs ayant un rôle d'administrateur. Tout d'abord, elle affiche les groupes où l'utilisateur est administrateur, ainsi que l'identifiant du groupe qu'il peut partager avec d'autres personnes.

Ensuite, l'administrateur a la possibilité d'accepter ou de refuser les demandes d'adhésion des utilisateurs au groupe. Cette fonctionnalité permet de contrôler qui peut rejoindre le groupe et garantit une gestion appropriée des membres.

De plus, la page administrative donne accès à la page de gestion de la répartition des conduites. Cependant, cette page est actuellement vide et non remplie, car je n'ai pas eu le temps de la développer. L'idée est que l'administrateur puisse attribuer les passagers aux conducteurs, assurant ainsi une répartition équitable des trajets au sein du groupe.

Ces fonctionnalités permettent à l'administrateur de gérer efficacement les groupes, d'accepter les utilisateurs pertinents et de faciliter la coordination des trajets entre les membres.

Nom du groupe	ID du groupe
Amigos	11001

Demande d'ajout au groupe

Lola Martin

Voulez-vous accepter cette personne ?

☐ Accepter

☐ Refuser

Envoyer

Gérer la répartition des conduites

FIGURE 10 - PAGE ADMINISTRATEUR

3. Données

Les données du projet sont stockées dans plusieurs fichiers XML. Étant donné le nombre de tables et de clés étrangères, il aurait peut-être été plus simple d'utiliser une autre méthode de stockage des données, mais cela respectait les exigences du cahier des charges.

Pour comprendre le fonctionnement de l'organisation, les fichiers XML qui sont communs à tous les groupes et à toutes les semaines sont situés dans le dossier "DATA", incluant les fichiers d'utilisateurs, de lieux, de groupes, etc. Cependant, afin de faciliter la gestion des données dans le temps, les données spécifiques à un moment précis sont stockées dans des dossiers suivant une structure logique.

Il existe en effet un dossier pour chaque année, et à l'intérieur de chaque année, un dossier par semaine correspondant au numéro de semaine. Dans chaque dossier de semaine, on trouve un sous-dossier pour chaque groupe créé. Lorsqu'un groupe est créé, une fonction est appelée pour créer un

sous-dossier correspondant dans chaque semaine, et y placer les fichiers d'initialisation (présents dans le dossier INIT). Malheureusement, je n'ai pas eu le temps de créer des schémas XML, mais ces fichiers d'initialisation permettent de récupérer la première ID, les champs, etc.

Les fichiers spécifiques comprennent, par exemple, les rendez-vous qui sont spécifiques à un groupe et probablement à une semaine donnée. Voici un schéma global pour illustrer ce qui a été expliqué ici.

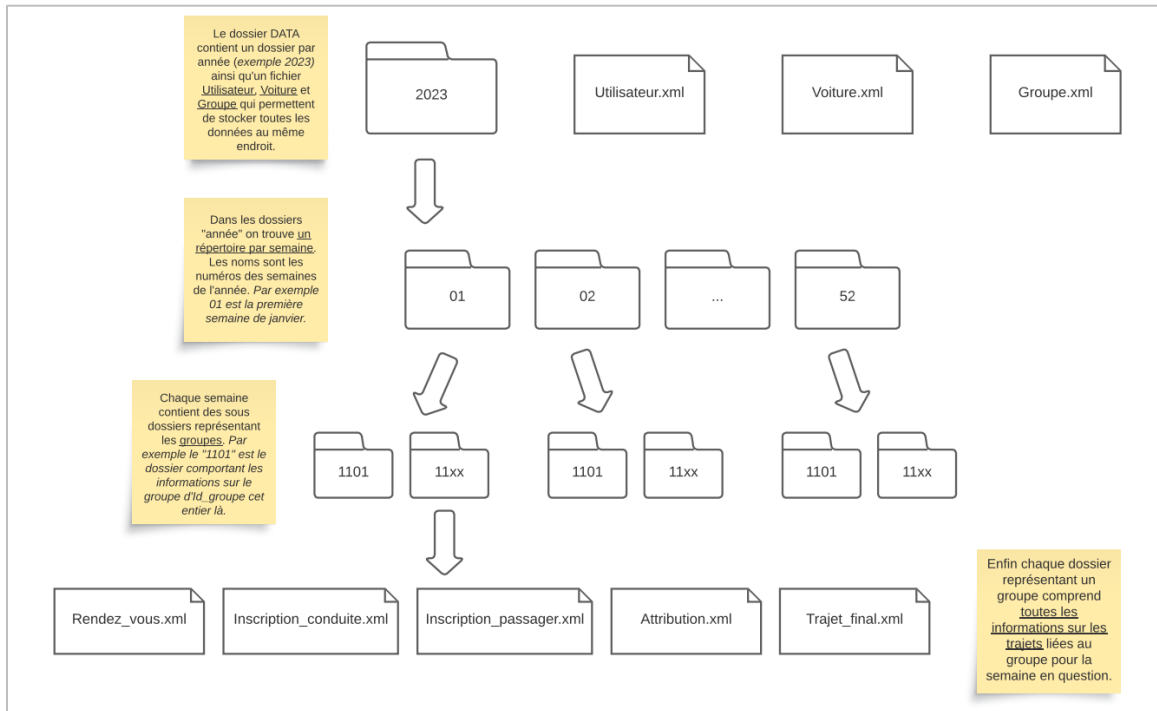


FIGURE 11 - LOGIQUE DU STOCKAGE DES FICHIERS XML

Voici également mes tables de données :

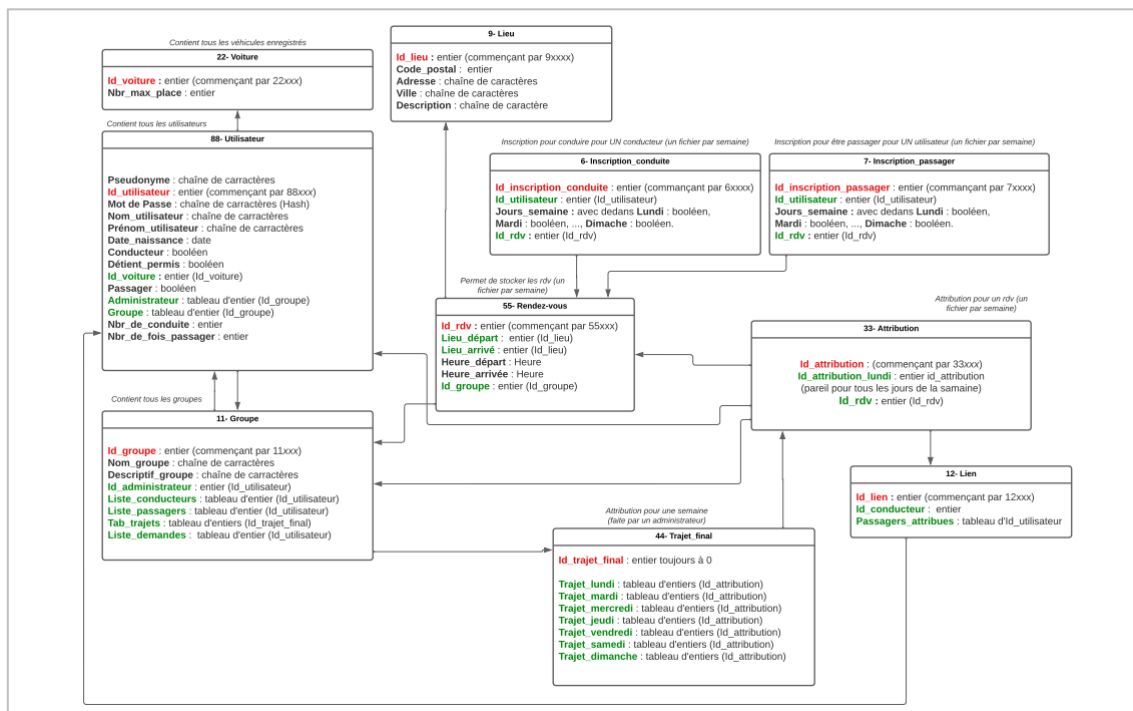


FIGURE 12 - UML DES TABLES DE DONNEES

4. Classes Java

Dans le cadre de mon projet, j'ai adopté une approche de conception orientée objet, où chaque entité possède sa propre classe Java dédiée. Cette approche me permet d'implémenter efficacement les opérations CRUD (Create, Read, Update, Delete) pour chaque table, grâce aux méthodes telles que get, set, add et del présentes dans chaque classe. En conséquence, mon application web bénéficie d'une structure modulaire et organisée, ce qui facilite la maintenance et la flexibilité du système. Afin de garantir la qualité de mon développement, j'ai préalablement créé un jeu de tests pour valider le bon fonctionnement de chaque classe servlet et assurer la fiabilité de mon application.

a. Exemple avec la classe Lieu_srv

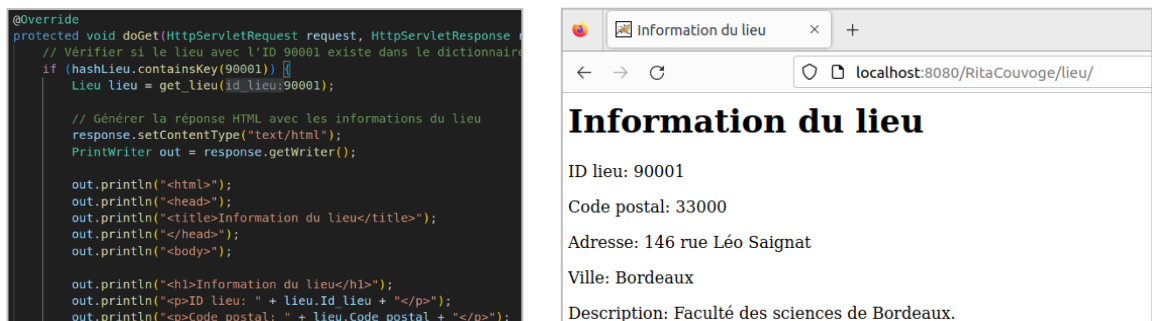
Dans le cadre de mon projet, j'ai initié le développement en créant une classe dédiée à la gestion des lieux, en me basant sur le fichier Lieu.xml.

Un des premiers tests que j'ai réalisés visait à vérifier l'implémentation du dictionnaire (hash) qui stocke tous les lieux présents dans le fichier Lieu.xml lors du lancement du serveur. Cela m'a permis de m'assurer que les données étaient correctement extraites et stockées dans le dictionnaire.



FIGURE 13 - EXEMPLE DE TESTS

Par la suite, j'ai testé les différentes fonctions de récupération (get) en fournissant un identifiant (ID) en tant que paramètre. Ces tests m'ont permis de vérifier si les fonctions renvoyaient les valeurs attendues en fonction de l'ID fourni, notamment le code postal, l'adresse, la ville et la description associés à un lieu spécifique.



Ainsi de suite pour toutes les fonctions CRUD.

Je n'ai malheureusement pas gardé tous les tests unitaires mais ils ont été réalisés pour chacune des fonctions.

Ces étapes intermédiaires de test effectuées sur le serveur Tomcat m'ont permis de valider progressivement la mise en place de chaque fonctionnalité dans ma classe de gestion des lieux. Cela m'a également aidé à détecter et corriger d'éventuelles erreurs ou incohérences au fur et à mesure de l'avancement du projet.

b. Intégration dans le projet

J'ai créé une classe statique qui permet l'instanciation des hash maps et qui peut être appelée de n'importe où dans l'application, dont dans les servlets. Cette approche centralise les données des fichiers XML et évite une utilisation excessive de XPath. En connaissant simplement l'identifiant de l'utilisateur, du groupe, etc., il est possible d'accéder rapidement à n'importe quel champ requis.

La classe statique est construite une seule fois lors de la page de connexion, ce qui évite le rechargement fréquent de la page. Cela garantit une performance optimale et évite des frais généraux inutiles.

Les avantages de cette approche sont multiples. Elle réduit la nécessité de modifier fréquemment les fichiers XML, car toutes les opérations métier sont effectuées à partir des instances de la classe statique. De plus, elle simplifie la gestion des données et améliore l'efficacité du traitement en centralisant les données et en évitant des opérations de lecture répétées sur les fichiers XML.

En résumé, l'utilisation de cette classe statique offre les avantages suivants :

- Élimination des problèmes de rechargement fréquent de la page.
- Réduction des modifications des fichiers XML.
- Amélioration des performances grâce à la manipulation des données à partir des instances de la classe statique.
- Centralisation des données pour une gestion plus efficace et des opérations métier simplifiées.

```
public class Variables_globales {
    public static final Attribution_srv g_attribution;
    public static final Groupe_srv g_groupe;
    public static final Utilisateur_srv g_utilisateur;
    public static final Voiture_srv g_voiture;
    public static final Lieu_srv g_lieu;
    public static final Map<String, Rendez_vous_srv> g_rdv;
    public static final Map<String, Trajet_final_srv> g_trajet_final;
    public static final Map<String, Inscription_conduite_srv> g_conduite;
    public static final Map<String, Inscription_passager_srv> g_passager;
    public static final Map<String, Lien_srv> g_lien;

    static {
        g_attribution = new Attribution_srv();
        g_groupe = new Groupe_srv();
        g_utilisateur = new Utilisateur_srv();
        g_voiture = new Voiture_srv();
        g_lieu = new Lieu_srv();
        g_rdv = new HashMap<>();
        g_trajet_final = new HashMap<>();
        g_conduite = new HashMap<>();
        g_passager = new HashMap<>();
        g_lien = new HashMap<>();
    }
}
```

FIGURE 14 - CLASSE STATIQUE

D'une manière générale, l'identifiant qui sert de clé dans les hash maps correspond à l'ID de chaque entité (utilisateur, lieu, etc.). Cependant, pour les tables spécifiques à chaque semaine, la classe statique comprend un hash map avec une clé composée de l'année, de la semaine et de l'ID du groupe. Cette clé permet d'accéder à la hash map correspondant à l'entité recherchée.

c. Servlets

Jusqu'à présent, je n'ai pas abordé les servlets dans les détails, mais dans mon projet, chaque servlet est associée à un fichier JSP. Le lien entre la servlet et le JSP est établi dans le fichier web.xml.

J'ai utilisé les méthodes `doGet()` et `doPost()` des servlets afin de définir des attributs à afficher dans les JSP et de récupérer les champs des formulaires des JSP.

Dans la méthode `doGet()`, j'ai utilisé des attributs pour stocker les données que je souhaitais afficher dans le JSP correspondant.

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    HttpSession session = req.getSession();
    Integer id_utilisateur = (Integer) session.getAttribute(name:"Id_utilisateur");

    if (id_utilisateur != null) {
        req.setAttribute(name:"pseudo", Variables_globales.g_utilisateur.get_pseudonyme(id_utilisateur));
        req.setAttribute(name:"nbr_fois_passager", Variables_globales.g_utilisateur.get_nbr_de_fois_passager(id_utilisateur));
        req.setAttribute(name:"nbr_fois_conducteur", Variables_globales.g_utilisateur.get_nbr_conduite(id_utilisateur));
    }
}
```

Par la suite, j'ai récupéré ces attributs dans le JSP en utilisant, entre autres, des expressions EL (Expression Language) pour les afficher dynamiquement.

Quant à la méthode `doPost()`, elle m'a permis de récupérer les champs des formulaires des JSP en utilisant les paramètres de la requête HTTP. J'ai extrait les valeurs de ces champs et les ai utilisées pour effectuer des opérations de traitement ou de mise à jour des données.

```
@Override
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String pseudo = request.getParameter(name:"pseudo");
    String password = request.getParameter(name:"password");
    String creer_compte = request.getParameter(name:"creer_compte");
    String formulaire = request.getParameter(name:"formulaire");
}
```

Pour conserver des informations spécifiques, telles que l'identifiant d'un rendez-vous sélectionné, j'ai également passé des paramètres dans les URL en utilisant la méthode GET. Cela m'a permis de les récupérer dans les JSP correspondants et de les afficher.

```
if (indexParam != null) {
    int id_rdv = liste_id_rdv.get(Integer.parseInt(indexParam));
    response.sendRedirect(request.getContextPath() + "/rdv_inscription?IndexRDV=" + id_rdv + "&Id_fichier_rdv=" + id_fichier_rdv);
}
```

En résumé, j'ai utilisé les méthodes `doGet()` et `doPost()` des servlets pour interagir avec les JSP. J'ai utilisé des attributs pour transmettre des données aux JSP et j'ai récupéré les champs des formulaires en utilisant les paramètres de la requête HTTP. J'ai également utilisé la méthode GET pour passer des paramètres dans les URL afin de conserver et d'afficher des informations spécifiques dans les JSP.

5. Conclusion

Dans ce rapport, les détails du fonctionnement du code ne sont pas abordés en profondeur. Cependant, vous trouverez dans la seconde partie des explications sur les configurations ainsi que des exemples de code issus des projets d'introduction. Le code est également disponible sur Git, ce qui vous permettra de l'examiner plus en détail. De plus, une vidéo de démonstration devrait être disponible pour illustrer le fonctionnement de l'application.

Malheureusement, je n'ai pas eu suffisamment de temps pour réaliser tout ce que je souhaitais ni pour rédiger le rapport de la manière que j'aurais souhaitée. Cependant, je suis satisfaite de ce que j'ai accompli, car le code que j'ai développé, bien qu'il puisse comporter quelques bugs selon les cas d'utilisation, est capable de parser des fichiers XML, d'écrire des données et de les rechercher. Ce projet m'a permis de découvrir et d'utiliser de nombreuses fonctionnalités des JSP (JavaServer Pages)

et des servlets, et m'a également plongée dans l'univers du Java. Malgré les contraintes de temps, je suis fière du travail accompli et des connaissances que j'ai acquises grâce à cette expérience.

II. Environnement technologique

1. Installations

Pour l'installation des différentes composantes nécessaires au projet, les étapes suivantes ont été suivies.

a. JDK

Le JDK est un kit de développement logiciel essentiel pour créer des applications Java. Pour l'installation, les commandes suivantes ont été exécutées :

```
sudo apt update
sudo apt-get install openjdk-8-jre
```

La version de Java installée peut être vérifiée avec la commande :

```
java -version
```

Il est également important d'ajouter le chemin d'installation à la variable d'environnement PATH en éditant le fichier ~/.bashrc et en ajoutant la ligne suivante :

```
nano ~/.bashrc
export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
source ~/.bashrc
```

b. Maven

Maven simplifie le processus de construction et de gestion des projets Java. Pour l'installer, la commande suivante a été utilisée :

```
sudo apt install maven
```

La commande mvn -version permet de vérifier si Maven est correctement installé.

```
mvn -version
```

c. Tomcat

Tomcat est un serveur web et de servlet open-source qui offre un environnement d'exécution pour les servlets Java et les pages JSP. Il permet de développer et de déployer des applications web dynamiques en utilisant les technologies Java.

Les étapes suivantes ont été suivies pour son installation :

```
cd ~
wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.75/bin/apache-tomcat-9.0.75.tar.gz
tar -xf apache-tomcat-9.0.75.tar.gz
sudo mv apache-tomcat-9.0.75 /opt/
sudo mv /opt/apache-tomcat-9.0.75/ /opt/tomcat9
nano ~/.bashrc
export CATALINA_HOME="/opt/tomcat9"
source ~/.bashrc
cd /opt/tomcat9
chmod +x bin/*.sh
/opt/tomcat9/bin/startup.sh
```

Pour vérifier si le serveur Tomcat est correctement lancé, il suffit d'accéder à l'URL <http://localhost:8080> dans un navigateur et de s'assurer que la page d'accueil de Tomcat s'affiche.

2. Faire un projet sans dépendance en guise d'exemple



But : Afficher le fameux « Hello world ! ».

Pour générer un nouveau projet Maven se placer dans le dossier de son choix et rentrer la commande suivante :

```
mvn archetype:generate -DgroupId=com.example -DartifactId=sans_dep -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Explication des différents champs de cette commande :

mvn archetype:generate	Permet de générer un nouveau projet à l'aide d'un archetype dans Maven. Ici l'archetype choisi est souvent utilisé pour créer des projets Java simple avec Maven.
-DgroupId=com.example	Identifiant du groupe pour le projet. Il est souvent basé sur le reverse domain name et identifie le groupe qui crée le projet.
-DartifactId=sans_dep	Spécifie l'identifiant de l'artefact, généralement le nom du projet.
-DarchetypeArtifactId=maven-archetype-quickstart	Spécifie l'identifiant de l'archetype à utiliser pour générer le projet. Dans ce cas, nous utilisons l'archetype "maven-archetype-quickstart" qui est un archetype standard pour les projets Java simples.
-DinteractiveMode=false	Désactive le mode interactif.

On se met dans le dossier créé :

```
cd sans_dep
```

Il faut modifier le fichier sans_dep/pom.xml afin de rajouter la version de Java utilisé :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <artifactId>sans_dep</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>sans_dep</name>
  <url>http://maven.apache.org</url>
</project>
```

Supprimer les lignes faisant référence aux dépendances. En effet elles ne sont pas utiles dans cette partie.

Un fichier App.java a été créé automatiquement dans le dossier /main/java/com/example. Il contient un code très simple :

```
package com.example;
public class App {
    public static void main( String[] args ){
```

```
}        System.out.println( "Hello World!" );  
}
```

Compiler le code avec la commande :

```
mvn compile
```

Puis pour exécuter le projet rentrer la commande suivante :

```
mvn exec:java -Dexec.mainClass="com.example.App"
```

✓ La phrase « Hello world! » qui s'affiche dans le terminal.

3. Faire un projet avec au moins une dépendance en guise d'exemple



But : Créer un document XML et afficher le nom des chevaux nées un jour pair grâce aux dépendances DOM et Apache Commons Lang.

Pour générer un nouveau projet sans interface Web avec Maven se placer dans le dossier de son choix et rentrer la commande suivante :

```
mvn archetype:generate -DgroupId=com.example -DartifactId=avec_dep -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

L'archetype « maven-archetype-webapp » est destiné aux projets web. Il génère une structure de projet adaptée au développement d'applications web, en incluant les répertoires de ressources tels que `src/main/webapp` pour les fichiers HTML, CSS, JavaScript, les fichiers JSP, les fichiers de configuration de servlet, etc. Il est préconfiguré pour utiliser les technologies web telles que Servlets, JSP et peut être utilisé pour le développement d'applications basées sur Jakarta EE.

On se met dans le dossier créé :

```
cd avec_dep
```

Il faut modifier le fichier `avec_dep/pom.xml` afin de rajouter la version de Java utilisé et nos dépendances. Ici j'ai rajouté en plus de la dépendance par défaut les dépendances DOM et Common Langs d'Apache :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <artifactId>avec_dep</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>avec_dep</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.w3c</groupId>
      <artifactId>dom</artifactId>
      <version>2.3.0-jaxb-1.0.6</version>
    </dependency>
    <dependency>
      <groupId>org.apache.commons</groupId>
```

```
<artifactId>commons-lang3</artifactId>
<version>3.12.0</version>
</dependency>
</dependencies>
</project>
```

Avant d'implémenter le fichier App.java créer un fichier xml qui regroupe les informations de 3 chevaux. Place ce document dans le même répertoire que le fichier java, c'est-à-dire src/main/java/com/example sous le nom de chevaux.xml :

```
<!-- Données récupérées du site https://www.ifce.fr/ -->
<chevaux>
  <cheval>
    <nom>BLUEMEISTER</nom>
    <sire>23325513F</sire>
    <race>Pur-sang</race>
    <sexe>Femelle</sexe>
    <robe>Alezan</robe>
    <date_naissance>2023-03-22</date_naissance>
  </cheval>
  ...
</chevaux>
```

Compléter par la suite le fichier App.java créé automatiquement dans le dossier /main/java/com/example. Voici quelques explications sur certaines parties du code :

- Les valeurs sont récupérées grâce aux noms des balises :

```
NodeList chevalNodes = document.getElementsByTagName("cheval");

// Parcours des nœuds cheval
for (int i = 0; i < chevalNodes.getLength(); i++) {
    Element chevalElement = (Element) chevalNodes.item(i);

    // Récupération des informations du cheval
    String nom = getElementText(chevalElement, "nom");
```

- Pour cela la méthode suivante permet de récupérer le texte d'un élément dans le document :

```
private static String getElementText(Element element, String tagName) {
    NodeList nodeList = element.getElementsByTagName(tagName);
    Element tagElement = (Element) nodeList.item(0);
    return tagElement.getTextContent();
}
```

- Utilisation de la dépendance d'Apache Commons Lang pour :
 - Récupère seulement le jour de naissance (similaire à cut en bash)
 - Vérifie que la valeur récupérée soit bien un nombre
 - Vérifie si le nombre est pair

```
boolean isJourPair =
StringUtils.isNumeric(StringUtils.substring(dateNaissanceStr, 8, 10))
&& Integer.parseInt(StringUtils.substring(dateNaissanceStr, 8, 10)) % 2 == 0;
```

Taper ensuite la commande :


```
mvn install
```

Cette commande fait les choses suivantes :

- Elle compile les fichiers source du projet
- Elle exécute les tests unitaires présents dans le projet pour vérifier que le code fonctionne correctement
- Maven crée un artefact (.jar) à partir des fichiers compilés et des ressources du projet
- Maven installe l'artefact dans le référentiel local de Maven.

En résumé la commande « `mvn compile` » est suffisante pour compiler le projet mais n'est pas suffisante pour inclure les dépendances du projet. Il faut donc bien utiliser la commande « `mvn compile` ».

Il faut ensuite appeler les deux commandes suivantes :

```
mvn compile
mvn exec:java -Dexec.mainClass="com.example.App"
```

✓ Le nom des deux chevaux nés un jour pair s'affiche comme attendu :

```
Nom du cheval né un jour pair : BLUEMEISTER
Nom du cheval né un jour pair : IBARS
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.836 s
[INFO] Finished at: 2023-05-30T22:37:20+02:00
[INFO]
```

4. Faire un projet d'exemple en Jakarta EE qui exploite JSP (ou JSF) ainsi que servlet



But : Générer une page web qui affiche les données météo du jour.

a. Générer le projet

Pour générer un nouveau projet avec une interface Web avec Maven se placer dans le dossier de son choix et rentrer la commande suivante :

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -
DarchetypeArtifactId=maven-archetype-webapp -DgroupId=com.example -
DartifactId=servlet_exemple
-Dversion=1.0-SNAPSHOT
```

L'archetype « maven-archetype-webapp » est destiné aux projets web. Il génère une structure de projet adaptée au développement d'applications web, en incluant les répertoires de ressources tels que src/main/webapp pour les fichiers HTML, CSS, JavaScript, les fichiers JSP, les fichiers de configuration de servlet, etc. Il est préconfiguré pour utiliser les technologies web telles que Servlets, JSP et peut être utilisé pour le développement d'applications basées sur Jakarta EE.

Se placer dans le dossier suivant :

```
cd servlet_exemple
```

b. API OpenWeather

L'objectif étant d'afficher la météo du jour j'ai décidé d'utiliser les données météorologique d'OpenWeather qui propose une API. Pour utiliser cette API il faut dans un premier temps se créer un compte sur [leur site](#).

Une fois le compte créé il faut se rendre dans l'onglet « API keys » afin d'obtenir une clef d'API gratuite. Cette clef permet d'accéder aux données météorologiques.

c. Gérer les dépendances

Modifier le fichier avec_dep/pom.xml afin de rajouter [la version de Java](#) utilisée et [les dépendances nécessaires](#). Dans cet exemple il est nécessaire de faire des requêtes sur l'API. Ainsi il faut inclure les dépendances pour le HttpClient d'Apache.

```
<!-- JSP -->
<dependency>
  <groupId>jakarta.platform</groupId>
  <artifactId>jakarta.jakartaee-api</artifactId>
  <version>8.0.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<!-- Apache HttpClient -->
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.13</version>
</dependency>
```

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpmime</artifactId>
  <version>4.5.13</version>
</dependency>

<!-- JSON -->
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20230227</version>
</dependency>
```

Il est possible de vérifier les versions des dépendances sur ce [site](#).

d. Création du fichier JSP

Un fichier JSP est une combinaison de balises HTML et de code Java qui permet de générer du contenu dynamique côté serveur. Les fichiers JSP sont généralement utilisés pour la présentation et l'affichage des données, en incorporant du code Java pour récupérer les données et les manipuler avant de les afficher dans la page web générée.

Se placée dans le dossier `src/main/webapp/` et ouvrir le fichier `index.jsp` :

```
nano index.jsp
```

Dans ce dernier fichier, y placer le code suivant permettant d'afficher la météo de la ville Texarkana aux Etats Unis.

La première ligne comporte une directive de page JSP.

La deuxième ligne est nécessaire pour importer la bibliothèque JSTL (JavaServer Pages Standard Tag Library) et utiliser des fonctionnalités de l'EL dans le fichier JSP.

La fin du fichier comprend des EL (Expression Language). Elles permettent d'accéder aux objets, aux propriétés et aux variables dans une manière plus concise.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
  <title>Météo du jour</title>
</head>
<body>
  <h1>Voici la météo du jour :</h1>
  <p><%= request.getAttribute("weatherData") %></p>
</body>
</html>
```

e. Création du Servlet

Les servlets sont des composants côté serveur qui permettent de gérer la logique de l'application, de traiter les données de la requête, d'effectuer des opérations et de générer une réponse dynamique.

Les servlets sont généralement utilisés pour gérer des tâches telles que la récupération et le traitement de données, l'interaction avec des bases de données, la génération de contenu dynamique, etc. Les servlets sont configurés dans le fichier de configuration (comme le fichier `web.xml` ou par annotations) pour spécifier les URL auxquelles ils répondent.

Créer dans un premier temps la classe « MeteoServlet.java » dans le répertoire « src/main/webapp/WEB-INF/classes/com/example ». Créer les sous dossiers si nécessaires.

Dans le fichier « src/main/webapp/WEB-INF/web.xml » remplacer la ligne « web-app... » par les lignes suivantes :

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-
app_4_0.xsd"
  version="4.0">

  <display-name>Servlet Exemple</display-name>

  <servlet>
    <servlet-name>MeteoServlet</servlet-name>
    <servlet-class>com.example.MeteoServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>MeteoServlet</servlet-name>
    <url-pattern>/meteo</url-pattern>
  </servlet-mapping>

</web-app>
```

Rajouter le code suivant dans le fichier « MeteoServlet.java » :

```
import java.io.IOException;

...

public class MeteoServlet extends HttpServlet {

    private final String apiKey = "8e83e2097e3966aad80c288d75c049d7";
    private final String latitude = "33.44";
    private final String longitude = "-94.04";

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

        try (CloseableHttpClient httpClient = HttpClient.createDefault();
            CloseableHttpResponse httpResponse = httpClient.execute(new
HttpGet("https://api.openweathermap.org/data/2.5/weather?lat="
+ latitude + "&lon=" + longitude + "&appid=" + apiKey)))
        {

            String responseBody =
EntityUtils.toString(httpResponse.getEntity());

            // Passer les données à la vue JSP
```

```

        request.setAttribute("weatherData", responseBody);
    }

    request.getRequestDispatcher("index.jsp").forward(request, response);
}
}

```

On y retrouve les coordonnées de la ville choisie

f. Construction de l'application Web

Après avoir sauvegardé rentrer les commandes suivantes :

```

mvn clean package
cp target/servlet_exemple.war /opt/tomcat9/webapps/servlet_exemple.war
sudo /opt/tomcat9/bin/startup.sh

```



Il faut penser à chaque fois à supprimer les fichiers/dossiers `servlet_exemple*` dans `webapps` de Tomcat ainsi que d'exteindre et rallumer le serveur à chaque fois, ce n'est pas optionnel. Ainsi après chaque modification de fichiers il faut faire :

```

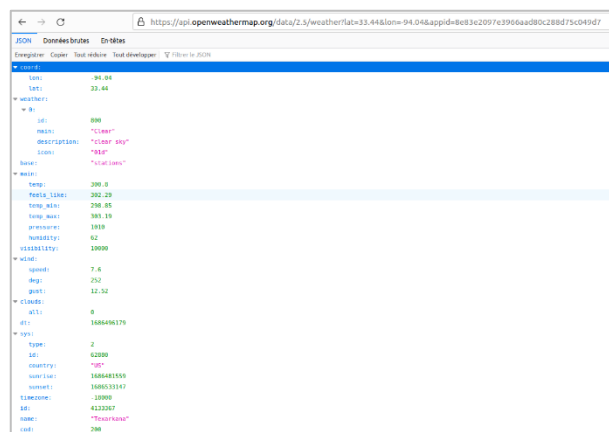
sudo /opt/tomcat9/bin/shutdown.sh
sudo rm -r /opt/tomcat9/webapps/servlet_exemple*
mvn clean package
cp target/servlet_exemple.war /opt/tomcat9/webapps/servlet_exemple.war
sudo /opt/tomcat9/bin/startup.sh

```

Pour le moment le rendu est :



Je ne sais pas pourquoi impossible de récupérer les JSON alors que l'API fonctionne (cf. capture d'écran ci-dessous).



✕ Voici un exemple de projet en Jakarta EE qui utilise JSP (ou JSF) et des servlets pour créer une application web. Veuillez noter que cet exemple est partiel et nécessite des ajustements pour être pleinement fonctionnel.