Rapport de Projet

Projet de Framework JEE

Fouade Dine Chitou
Ndongo Medoune Ndiaye
Mame Marieme Selbe Toure
Mathurchan Jeyakanthan

Introduction

Le projet consiste à développer une application web pour un support technique, en suivant l'architecture **MVC 2** (Model-View-Controller). L'objectif principal était de concevoir un système fonctionnel et intuitif, permettant d'assurer un suivi efficace des requêtes, depuis leur soumission par un utilisateur jusqu'à leur résolution par un administrateur.
L'application répond à deux objectifs spécifiques :
Pour les utilisateurs : Proposer un formulaire simple et accessible pour soumettre des requêtes techniques.
Pour les administrateurs : Mettre à disposition un tableau de bord interactif pour visualiser, filtrer, et traiter les requêtes selon leur statut (Nouvelle, En cours, Terminée).
Ce projet s'inscrit dans une démarche d'apprentissage des technologies modernes utilisées dans le développement d'applications web Java, notamment JSP, JSF et JPA. En intégrant ces technologies, nous avons cherché à construire une architecture robuste, maintenable, et évolutive tout en répondant aux besoins exprimés.

Objectif général

Concevoir et implémenter une application web en deux versions distinctes :
Une version utilisant **JSP et Servlets**, respectant l'architecture traditionnelle MVC.
Une version avancée avec **JSF** et **JPA**, pour explorer les possibilités offertes par ces technologies modernes et améliorer la gestion des interactions entre l'interface utilisateur, la logique métier, et la persistance des données.

Schéma de Navigation

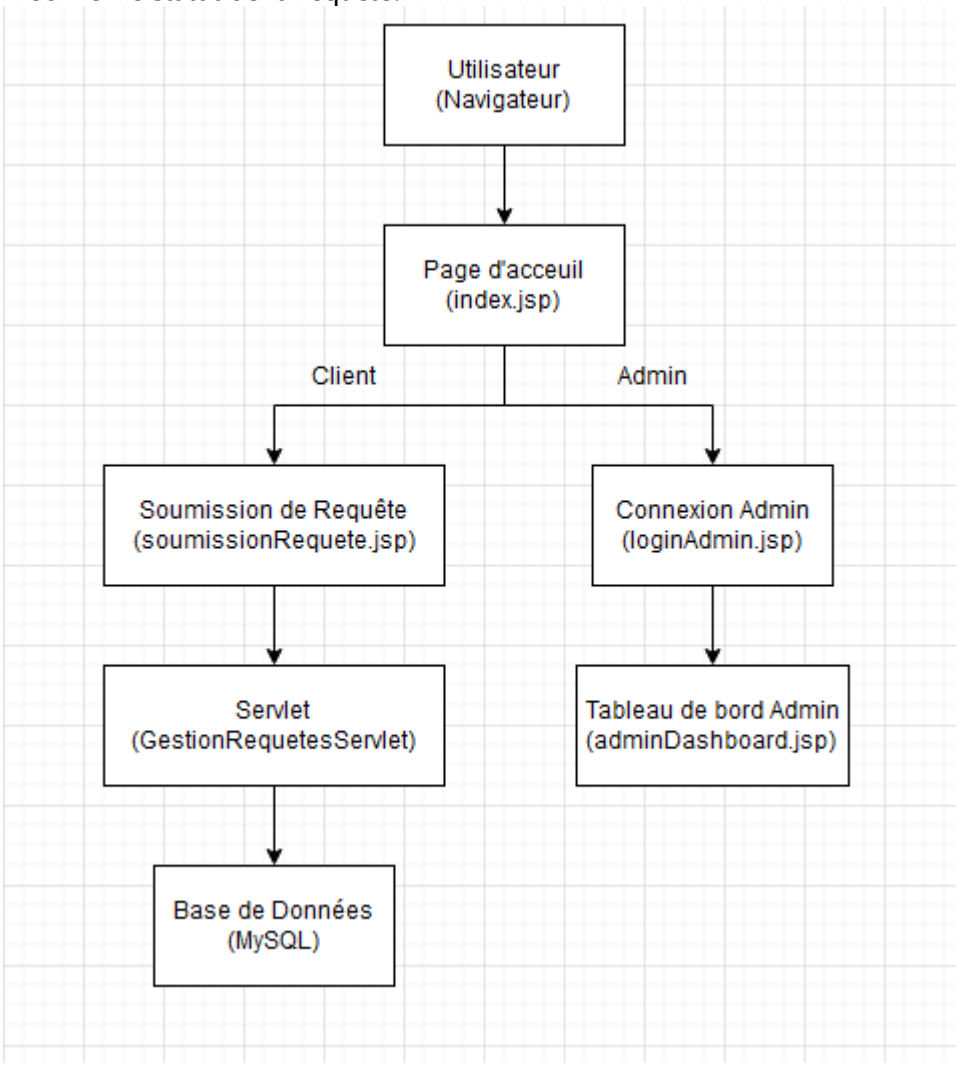
Lors de cette première itération, l'application a été développée en suivant l'architecture **MVC 2**, en s'appuyant sur :

Les Servlets : Chargées de centraliser les traitements et de rediriger vers les pages JSP appropriées.
Les Pages JSP : Utilisées comme vues pour afficher les données et collecter les entrées utilisateur.
Les JavaBeans : Utilisés pour représenter les données et fournir une structure simple pour échanger des informations entre la couche de contrôle (Servlets) et la vue (JSP).

Description du Flux de Navigation

1. **Accueil (index.jsp)** :
   - L'utilisateur accède à la page principale.
   - Les options disponibles permettent soit de soumettre une requête (utilisateur), soit de se connecter (administrateur).
2. **Soumission de Requête (soumissionRequete.jsp)** :
   - L'utilisateur est redirigé vers cette page pour remplir un formulaire de requête technique.
   - Une fois soumis, le formulaire envoie les données à une servlet dédiée, qui traite et enregistre la requête dans la base de données.
3. **Connexion Administrateur (loginAdmin.jsp)** :
   - L'administrateur accède à cette page pour s'authentifier.
   - En cas de succès, il est redirigé vers le tableau de bord administrateur.

4. **Tableau de Bord Administrateur (adminDashboard.jsp)** :
    - L'administrateur visualise les requêtes techniques dans une table.
    - Il peut filtrer les requêtes par statut, mettre à jour leur état, ou consulter les détails d'une requête spécifique.
5. **Détail d'une Requête (detailRequete.jsp)** :
    - L'administrateur peut visualiser les informations détaillées d'une requête sélectionnée.
    - Un formulaire est disponible pour répondre directement à l'utilisateur concerné ou pour modifier le statut de la requête.



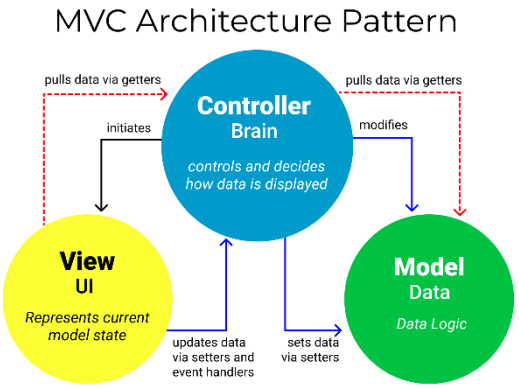
Interaction entre Composants

- **Les Servlets** :
    - Centralisent les actions et redirigent les requêtes vers les pages JSP.
    - Exemples :
        - GestionRequetesServlet : Traite les soumissions de requêtes utilisateur.
        - AuthenticationServlet : Gère l'authentification de l'administrateur.
- **Les JSP** :
    - Collectent les données via des formulaires (ex : formulaire de soumission de requête).
    - Affichent les résultats ou les données sous forme de tables (ex : tableau de bord administrateur).
- **Les JavaBeans** :

- Servent de modèle de données pour stocker temporairement les informations échangées entre les Servlets et les JSP.

Description de l'Architecture Utilisée

Comme dit au début 'architecture de cette application repose sur le modèle **MVC 2 (Modèle-Vue-Contrôleur)**. Cette organisation permet de séparer clairement les responsabilités entre la présentation des données (Vue), la gestion de la logique métier (Contrôleur), et la manipulation des données (Modèle). En adoptant cette architecture, nous avons cherché à rendre l'application plus modulaire, facile à maintenir et évolutive.



1. La Vue : Interface Utilisateur avec JSP

La **Vue** est la partie visible de l'application, celle avec laquelle les utilisateurs interagissent directement. Ici, elle est construite à l'aide de **pages JSP** (JavaServer Pages). Ces pages servent à afficher les données récupérées depuis la base de données et à collecter les informations saisies par l'utilisateur à travers des formulaires.
Utilité :

- Les pages JSP permettent d'intégrer dynamiquement des données dans une page web en utilisant des expressions Java simples.
- Elles facilitent l'interaction utilisateur en affichant des formulaires ou des tableaux de données.

Exemples dans le projet :

1. **Page de Soumission de Requête (soumissionRequete.jsp)** :
    - Permet aux utilisateurs de soumettre des requêtes techniques en remplissant un formulaire.
    - Ces informations sont ensuite envoyées au contrôleur pour traitement.
2. **Page de Tableau de Bord Administrateur (adminDashboard.jsp)** :
    - Affiche une liste des requêtes soumises.
    - Permet aux administrateurs de mettre à jour le statut des requêtes ou d'accéder à leurs détails.

Les pages JSP sont essentielles pour offrir une expérience utilisateur fluide et intuitive.

2. Le Contrôleur : Servlets

Les **Servlets** sont le cœur de la logique métier de l'application. Elles agissent comme des intermédiaires entre la Vue (JSP) et le Modèle (JavaBeans et DAO). Chaque servlet reçoit les requêtes HTTP des utilisateurs, exécute les traitements nécessaires, et redirige vers la page JSP appropriée.

Utilité :

- Elles centralisent le traitement des actions, comme l'enregistrement d'une nouvelle requête ou l'authentification d'un utilisateur.
- Elles garantissent que la logique métier reste séparé de la présentation, ce qui facilite la maintenance.

Exemples dans le projet :

1. **Servlet GestionRequetesServlet** :
    - Reçoit les données d'un formulaire de soumission de requête.
    - Valide les données, les transforme en objet Requete, et les enregistre dans la base de données via un DAO.
    - Redirige l'utilisateur vers une page de confirmation après le traitement.
2. **Servlet AuthenticationServlet** :
    - Valide les identifiants fournis par un administrateur lors de la connexion.
    - Si l'authentification réussit, redirige vers le tableau de bord administrateur.

Grâce aux servlets, la gestion des actions utilisateur est centralisée et organisée de manière cohérente.

3. Le Modèle : JavaBeans et DAO

Le **Modèle** représente les données manipulées par l'application. Dans ce projet, il est composé de **JavaBeans** pour modéliser les entités (comme les utilisateurs ou les requêtes) et de **DAO** pour interagir avec la base de données.

JavaBeans

Un JavaBean est une classe Java qui encapsule les données et fournit des méthodes pour y accéder (getters et setters). Chaque entité clé de l'application est représentée par un JavaBean.

Utilité :

- Les JavaBeans offrent une structure claire pour manipuler les données de manière cohérente.
- Ils facilitent le transfert des données entre les servlets et les pages JSP.

Exemples dans le projet :

1. **UtilisateurBean** :
    - Contient les informations sur les utilisateurs, comme leur nom, email, mot de passe, et rôle (ADMIN ou UTILISATEUR).
2. **RequeteBean** :

- Représente une requête soumise par un utilisateur.
- Contient des informations comme le sujet, la description, et le statut de la requête (NOUVELLE, EN_COURS, TERMINEE).

DAO (Data Access Object)

Les DAO encapsulent toute la logique d'accès à la base de données. Ils utilisent des requêtes SQL pour ajouter, modifier, ou récupérer des données.

Utilité :

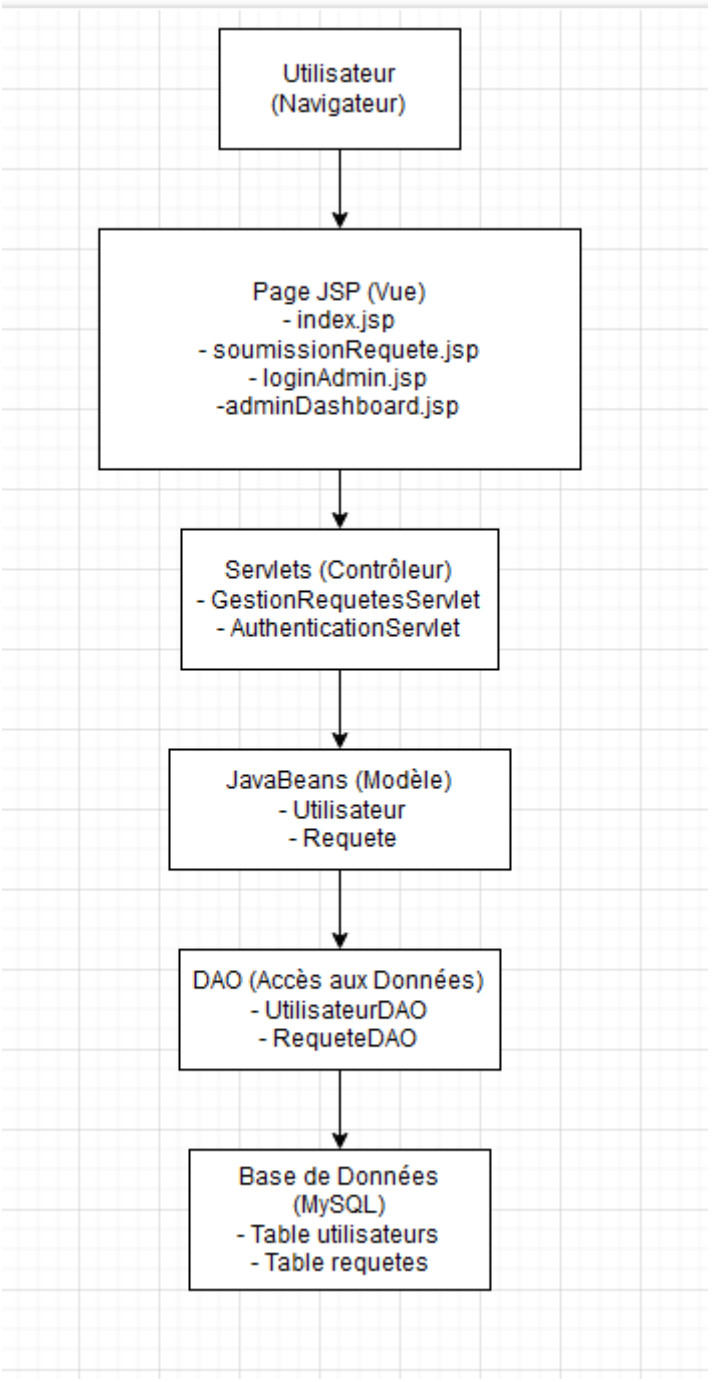
- Ils isolent la logique de persistance du reste de l'application, ce qui rend le code plus propre et maintenable.
- Ils permettent de centraliser toutes les opérations sur la base de données dans un seul composant.

Exemples dans le projet :

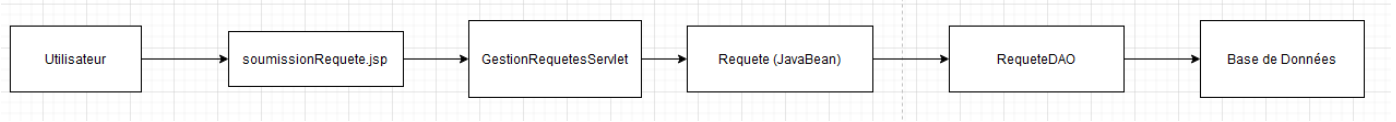
1. **UtilisateurDAO** :
   - Gère les opérations liées aux utilisateurs, comme la vérification des identifiants lors de la connexion.
2. **RequeteDAO** :
   - Gère les requêtes, avec des méthodes pour récupérer toutes les requêtes, mettre à jour leur statut, ou les supprimer.

Flux des Données dans l'Application

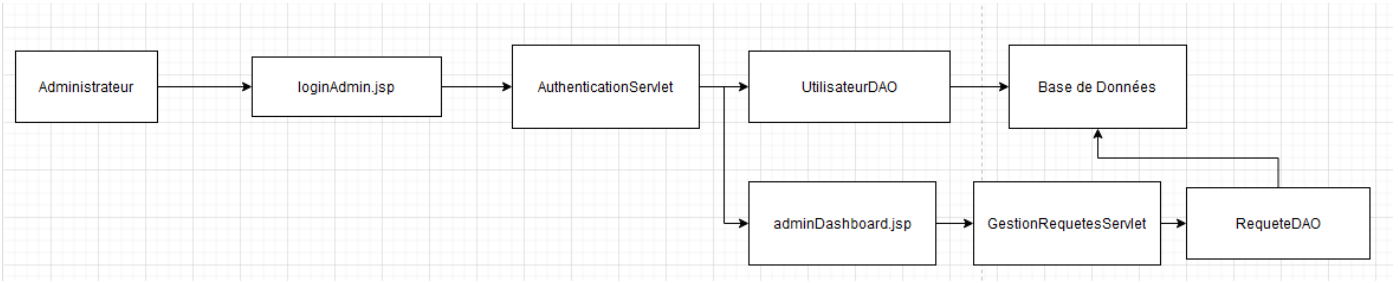
1. **Soumission d'une Requête** :
   - L'utilisateur remplit un formulaire sur la page JSP soumissionRequete.jsp.
   - Les données sont envoyées à la servlet GestionRequetesServlet via une requête HTTP POST.
   - La servlet valide les données, les transforme en objet Requete, et appelle le DAO pour les enregistrer dans la base de données.
2. **Connexion Administrateur** :
   - L'administrateur saisit ses identifiants sur la page loginAdmin.jsp.
   - Les données sont transmises à la servlet AuthenticationServlet.
   - La servlet vérifie les identifiants via UtilisateurDAO. Si la connexion est valide, l'administrateur est redirigé vers le tableau de bord.
3. **Gestion des Requêtes** :
   - Dans le tableau de bord, l'administrateur peut mettre à jour le statut des requêtes ou consulter leurs détails.
   - Ces actions déclenchent des appels à RequeteDAO via GestionRequetesServlet.



Représentation des Flux Utilisateur (Soumission de Requête) :



Représentation Administrateur (Connexion et Gestion des Requêtes) :



Listing de l'application commentés

Nous allons présenter les extraits de code les plus important de l'application en ajoutant des commentaires détaillés pour expliquer leur rôle et leur fonctionnement.

1. Gestion des Utilisateurs : Servlet d'Authentification
Cette servlet gère la connexion des administrateurs en validant leurs identifiants et en les redirigeant vers le tableau de bord en cas de succès.

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

    UtilisateurDAO utilisateurDAO;

    @Override
    public void init() throws ServletException {
        utilisateurDAO = new UtilisateurDAO();
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Récupérer les paramètres du formulaire de connexion
        String email = request.getParameter("email");
        String password = request.getParameter("password");

        try {
            // Vérifier les informations d'identification de l'utilisateur
            Utilisateur utilisateur = utilisateurDAO.verifierIdentifiants(email, password);

            if (utilisateur != null) {
                // Stocker les informations de l'utilisateur dans la session
                request.getSession().setAttribute("utilisateurConnecte", utilisateur);

                // Rediriger en fonction du rôle de l'utilisateur
                if ("admin".equals(utilisateur.getRole())) {
                    response.sendRedirect("gestionRequetesAdmin");
                } else {
                    response.sendRedirect("mesRequetes");
                }
            } else {
                // Si les identifiants sont incorrects, renvoyer vers la page de connexion avec un message d'erreur
                request.setAttribute("errorMessage", "Email ou mot de passe incorrect.");
                request.getRequestDispatcher("/login.jsp").forward(request, response);
            }

        } catch (IOException e) {
            throw new ServletException("Erreur lors de l'authentification de l'utilisateur", e);
        }
    }
}
```

Les identifiants sont récupérés depuis le formulaire soumis par la page loginAdmin.jsp.
La méthode verifierIdentifiants du DAO interroge la base de données pour valider les informations.
En cas de succès, l'utilisateur est enregistré dans la session et redirigé vers le tableau de bord.

2. Gestion des Requêtes : Servlet de Soumission

Cette servlet traite les formulaires soumis par les utilisateurs pour créer de nouvelles requêtes techniques.

```java
public class GestionRequeteUtilisateurServlet extends HttpServlet {

    private RequeteDAO requeteDAO;

    @Override
    public void init() throws ServletException {
        requeteDAO = new RequeteDAO();
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Récupérer l'utilisateur connecté
        Utilisateur utilisateur = (Utilisateur) request.getSession().getAttribute("utilisateurConnecte");

        if (utilisateur == null) {
            response.sendRedirect("login.jsp");
            return;
        }

        // l'utilisateur doit pouvoir afficher la liste de ses requetes

        Requete requete = requeteDAO.recupererRequeteParId(utilisateur.getId());
        request.setAttribute("requete", requete);
        request.getRequestDispatcher("/gestionRequetesUtilisateur.jsp").forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Récupérer l'utilisateur connecté
        Utilisateur utilisateur = (Utilisateur) request.getSession().getAttribute("utilisateurConnecte");

        if (utilisateur == null) {
            response.sendRedirect("login.jsp");
            return;
        }
```

- Les données du formulaire (sujet et description) sont extraites de la requête HTTP.
- Un objet Requete est créé et initialisé avec les données collectées.
- La méthode ajouterRequete du DAO enregistre la requête dans la base de données.
- L'utilisateur est redirigé vers une page de confirmation (confirmation.jsp).

3. JavaBean : Requete

Le JavaBean Requete représente une requête technique soumise par un utilisateur. Voici les parties essentielles du code.

```java
@Entity
@Table(name = "requetes")
public class Requete implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "utilisateur_id", nullable = false)
    private Utilisateur utilisateur;

    @Column(nullable = false)
    private String sujet;

    @Column(nullable = false, length = 500)
    private String description;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "date_creation", nullable = false, updatable = false)
    private Date dateCreation;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Statut statut;

    // Enumération pour le statut de la requête (Nouvelle, En cours, Terminée)
    public enum Statut {
        NOUVELLE,
        EN_COURS,
        TERMINEE
    }

    // Constructeur par défaut requis par JPA
    public Requete() {
        this.dateCreation = new Date();  // Initialise la date de création à l'instance courante
        this.statut = Statut.NOUVELLE;   // Par défaut, le statut est "Nouvelle"
    }

    // Constructeur avec paramètres
    public Requete(Utilisateur utilisateur, String sujet, String description) {
        this.utilisateur = utilisateur;
        this.sujet = sujet;
        this.description = description;
        this.dateCreation = new Date();  // Date de création au moment de la requête
        this.statut = Statut.NOUVELLE;   // Statut initial par défaut
    }
```

```java
    // Getters et Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Utilisateur getUtilisateur() {
        return utilisateur;
    }

    public void setUtilisateur(Utilisateur utilisateur) {
        this.utilisateur = utilisateur;
    }

    public String getSujet() {
        return sujet;
    }

    public void setSujet(String sujet) {
        this.sujet = sujet;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Date getDateCreation() {
        return dateCreation;
    }

    public void setDateCreation(Date dateCreation) {
        this.dateCreation = dateCreation;
    }

    public Statut getStatut() {
        return statut;
    }
}
```

- Ce JavaBean encapsule les données associées à une requête, comme son sujet, sa description, et son statut.
- Les getters et setters permettent de manipuler ces données facilement depuis les servlets ou DAO.

4. Vue : Exemple de Page JSP
Voici un extrait de la page soumissionRequete.jsp, utilisée pour collecter les données d'une nouvelle requête.

```
<form action="GestionRequeteUtilisateurServlet" method="post">
    <label for="sujet">Sujet :</label>
    <input type="text" id="sujet" name="sujet" required />

    <label for="description">Description :</label>
    <textarea id="description" name="description" rows="4" required></textarea>

    <button type="submit">Envoyer la Requête</button>
</form>
```

- Ce code utilise la méthode POST pour envoyer les données à la servlet GestionRequetesServlet.
- Les champs sont obligatoires (required), ce qui garantit que l'utilisateur ne peut pas soumettre un formulaire incomplet.

Migration vers JSF

La migration de JSP vers JSF (JavaServer Faces) avait pour objectif de moderniser l'application en adoptant un framework plus structuré et adapté aux besoins d'interfaces dynamiques. JSF offre plusieurs avantages par rapport à JSP, notamment une gestion simplifiée des composants d'interface, des fonctionnalités de validation intégrées et une meilleure intégration avec les Managed Beans.

Approche de Migration
La migration a été réalisée en suivant les étapes principales suivantes :

1. **Création des Fichiers XHTML** :
    - Les pages JSP existantes ont été réécrites en XHTML pour tirer parti des composants JSF.
    - Par exemple, la page soumissionRequete.jsp a été transformée en soumissionRequete.xhtml, avec des composants comme <h:form>, <h:inputText> et <h:commandButton>.

Exemple de migration :

JSP :

```
<form action="GestionRequetesServlet" method="post">
    <input type="hidden" name="id" value="${requete.id}" />
    <textarea name="reponse" required></textarea>
    <button type="submit" name="action" value="respond">Envoyer</button>
</form>
```

JSF :

```
<h:form>
    <h:outputLabel for="reponse" value="Réponse :" />
    <h:inputTextarea id="reponse" value="#{requeteBean.reponse}" rows="5" required="true" />
    <h:commandButton value="Envoyer Réponse" action="#{requeteBean.repondreARequete}" />
</h:form>

<h:form>
    <h:commandButton value="Marquer comme Terminée" action="#{requeteBean.terminerRequete}" />
</h:form>
```

Utilisation des Managed Beans :

- Les servlets ont été remplacées par des Managed Beans, qui assurent la logique métier et permettent une interaction directe avec les composants JSF.
- Par exemple, un Managed Bean RequeteBean a été créé pour gérer les opérations liées aux requêtes.

Navigation avec faces-config.xml :

- Un fichier faces-config.xml a été ajouté pour définir la navigation entre les différentes pages JSF.

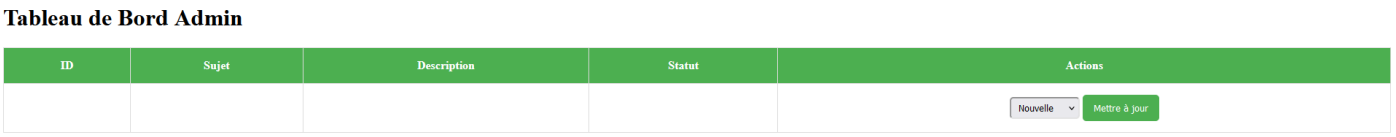
Ajout des Validations :

- Les composants JSF offrent des mécanismes de validation intégrés (ex : required="true") pour s'assurer que les données saisies sont valides avant d'être traitées.

Résultats Obtenus :

- Les pages soumissionRequete.xhtml, loginAdmin.xhtml, adminDashboard.xhtml et detailRequete.xhtml ont été créées avec JSF.
- Les Managed Beans AuthBean et RequeteBean ont été implémentés pour gérer les interactions et la logique métier.

Page d'accueil sous un serveur tomcat



Dashboard pour l'admin sur la version jsp

Problèmes Rencontrés :

1. **Erreur 500 sur la Page de Connexion (loginAdmin.xhtml)** :
    - Lors de la soumission des identifiants via JSF, une erreur 500 était systématiquement rencontrée.
    - L'erreur était liée à une mauvaise configuration ou un problème dans la résolution des Managed Beans (AuthBean).

2. **Absence de Résolution des Managed Beans** :
    - Certains Managed Beans comme AuthBean et RequeteBean n'étaient pas correctement instanciés ou liés aux composants JSF, ce qui rendait impossible l'exécution des actions associées.

Conclusion

Ce projet avait pour objectif de développer une application de gestion de requêtes techniques en adoptant une architecture claire et structurée basée sur le modèle MVC 2. Nous avons utilisé des composants élémentaires tels que les servlets, les pages JSP et les JavaBeans pour répondre aux exigences initiales. Par la suite, nous avons amorcé une migration vers JSF pour moderniser l'application et tirer parti des fonctionnalités avancées offertes par ce framework.

Résumé des Réalisations :

1. **Architecture MVC 2** :
    - Mise en place d'une structure modulaire avec une séparation claire entre la Vue (JSP), le Contrôleur (servlets) et le Modèle (JavaBeans et DAO).
    - Gestion centralisée des actions utilisateur via des servlets, avec une logique métier encapsulée dans des DAO.
2. **Fonctionnalités Clés** :
    - Soumission de requêtes techniques par les utilisateurs via une interface intuitive.
    - Gestion des requêtes par les administrateurs, incluant la mise à jour des statuts et l'affichage des détails des requêtes.
    - Authentification des administrateurs pour sécuriser l'accès aux fonctionnalités sensibles.
3. **Migration vers JSF** :
    - Création de nouvelles pages en JSF (soumissionRequete.xhtml, loginAdmin.xhtml, etc.).
    - Introduction des Managed Beans pour gérer les interactions avec les pages JSF.
    - Tentative d'adoption des mécanismes avancés de validation et de navigation offerts par JSF.

Points d'Amélioration

1. **Correction des Erreurs dans la Migration JSF** :
    - Résoudre les erreurs 500 rencontrées, notamment celles liées à l'instanciation des Managed Beans (AuthBean, RequeteBean).
    - Tester et finaliser la configuration de navigation entre les pages JSF.
2. **Validation et Sécurité** :
    - Améliorer la validation des entrées utilisateur pour éviter les saisies incorrectes ou malveillantes.
    - Implémenter un hachage des mots de passe pour renforcer la sécurité.

En conclusion, ce projet a servi d'introduction aux concepts fondamentaux des frameworks pour le développement d'applications web en Java. En explorant Jakarta Servlet, JSP, et une migration partielle vers JSF, nous avons acquis une meilleure compréhension de l'architecture MVC et des outils modernes pour la création d'interfaces utilisateur dynamiques.

L'objectif principal était de démontrer comment différents frameworks s'intègrent pour répondre à des besoins spécifiques, comme la gestion des requêtes techniques, tout en illustrant les avantages et les limites de chacun. Bien que l'application soit encore en développement, notamment sur le frontend où des erreurs subsistent, elle constitue une base solide pour appliquer et approfondir les compétences apprises en cours.

Ce projet met en évidence le rôle clé des frameworks dans la simplification et l'accélération du développement web, tout en soulignant l'importance de l'apprentissage pratique pour surmonter les défis techniques.