Créateur :

SHKURTI Gledis
BONNORON Clément

Présentation du projet

Le but de ce projet est de créer une plateforme multi-utilisateur et multi-kanbans. Un kanban est un tableau permettant de rendre compte de l'état d'avancement des tâches d'un projet, partagées entre plusieurs utilisateurs et permettant à chacun de prendre en charge une tâche et de la déplacer dans un état.

Fonctionnalitées

Chaque projet (kanban) sera constitué d'une description ainsi que de plusieurs colonnes (entre 2 et 7) permettant de représenter les états d'une tâche. Ces colonnes seront définies lors de la création, en obligeant à avoir 2 colonnes obligatoires : Stories (gauche) et Terminées (droite).
Le kanban pourra également être défini comme public, permettant à n'importe qui de voir les kanbans sans pouvoir le modifier, ou privé ne permettant qu'aux invités de voir et modifier le kanban.

Le créateur du kanban sera le gestionnaire de celui-ci, et pourra inviter/retirer d'autres utilisateurs afin qu'ils puissent eux aussi gérer les tâches.

Chaque tâche sera définie par une description, pourra être attribuée à un utilisateur du projet, et avoir une date limite de réalisation. Une tâche non attribuée pourra être prise par n'importe quel invité, tandis que le gestionnaire pourra attribuer les tâches à n'importe qui.

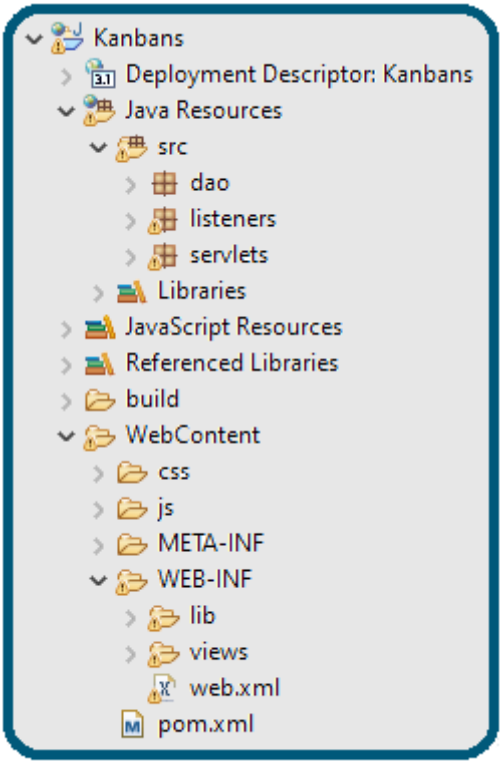
Chaque utilisateur se connectant sur le serveur aura plusieurs possibilités :

- Se connecter
- Créer un compte
- Accéder aux kanbans publics

Une fois connecté, l'utilisateur aura accès à l'ensemble des kanbans dont il est gestionnaire, sur lesquels il est invité, mais également la liste des tâches qui lui sont attribuées sur un kanban ou en général.
Il pourra décider de créer un nouveau kanban, mais également de supprimer un kanban dont il est le gestionnaire.

Structuration projet Eclipse

Avant de commencer toute explication sur les choix techniques utilisés lors du développement de cette application web, ayons un bref aperçu de la structuration d'un projet web ainsi que les principaux fichiers de configuration de celui-ci. Nous ne rentrons pas en détails sur la composition des fichiers dans cette partie, mais uniquement sur leur utilité :



Les deux fichiers permettant la bonne configuration du fichier de déploiement sont les fichiers *web.xml* et *pom.xml*. Le premier va permettre de correctement initialiser les pages utilisées une fois le serveur déployé (utilisateur et d'erreur), tandis que le second va permettre de définir les dépendances utilisées lors du déploiement du serveur.

Le dossier *WebContent* est le dossier qui va contenir l'ensemble des sources utilisés par le projet web :

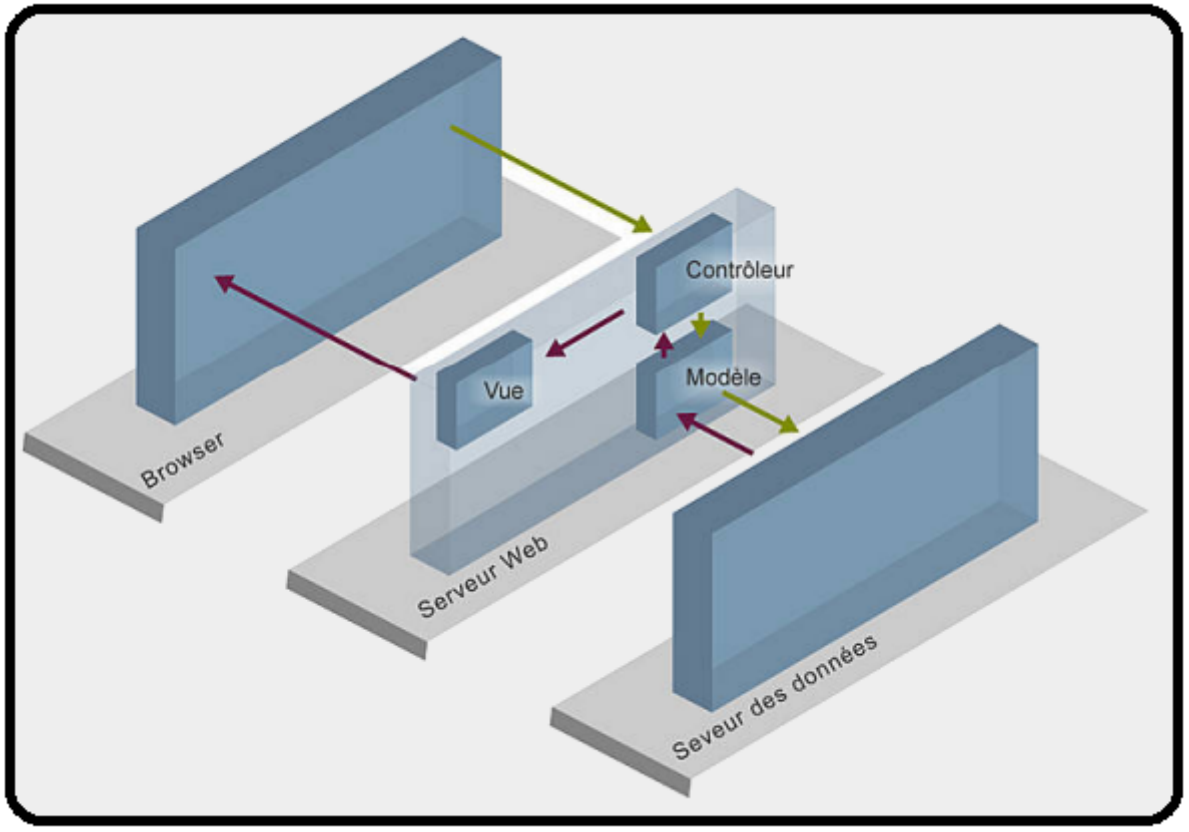
- Nous avons créé deux dossiers, *css* et *js* afin de structurer notre code et accéder aux sources simplement au même endroit.

- Le dossier *WEB-INF* contient plusieurs sous-dossiers, qui ne seront pas directement accessibles sur le serveur :
  - ***lib***: Contient l'ensemble des librairies externes utilisées par le serveur web
  - ***views***: Contient l'ensemble de nos pages web

- Le dossier *Java Resources* contient les fichiers java qui permettent de gérer la partie de la base de données et la partie contrôleurs de l'application.

Le modèle MVC

Pour rappel, revoyons le principe du modèle MVC ainsi que l'implémentation que nous avons décidé de faire afin de suivre ce modèle :



Nous avons trois grandes familles :

- Côté client
- Côté serveur
- Côté base de données

Nous avons respecté ce modèle car nous avons respecté la séparation physique de ces trois éléments.

1. Côté client

Du côté client, nous ne gérons aucune manipulation faite par l'utilisateur. En effet, il s'agit de l'utilisation que fait le client de notre serveur et nous n'avons aucune action à faire, à part récupérer ses manipulations, et lui envoyer ce qu'il doit voir.

2. Côté serveur

Du côté serveur, nous séparons nos actions en 3 blocs :

- La vue
- Les contrôleurs
- Le modèle

Chacun de ces blocs va avoir un intérêt, une action particulière, et des interactions entre eux et avec le côté client/côté base de données.

2.1. Les contrôleurs

Contient l'ensemble des programmes qui vont recevoir l'ensemble des requêtes provenant de l'interface utilisateur. Ces programmes vont avoir pour effet de modifier la vue en fonction des requêtes faites par l'utilisateur, en interagissant avec le modèle.

Dans notre projet, les contrôleurs vont être constitués à l'aide des Servlets se trouvant dans le dossier : ***/Java Resources/src/servlets/***

2.1. Le modèle

Le modèle va avoir pour but d'encapsuler la gestion de l'état de l'application, de gérer les états possibles que peuvent prendre les données. Ce bloc ne va pas interagir directement avec la vue, cependant les données contenues dans le modèle seront utilisées afin d'afficher les informations nécessaires à l'utilisateur.

Dans notre projet, le modèle va se trouver dans plusieurs dossiers :

/Java Resources/src/dao/
/Java Resources/src/domain/

2.1. La vue

La vue quant à elle va contenir toutes les bibliothèques/classes permettant de mettre en forme les informations que nous souhaitons afficher à l'utilisateur (données, erreurs, etc…)

Dans notre projet, la vue va se trouver dans plusieurs dossiers :

/WebContent/css/
/WebContent/js/

/WebContent/WEB-INF/views/

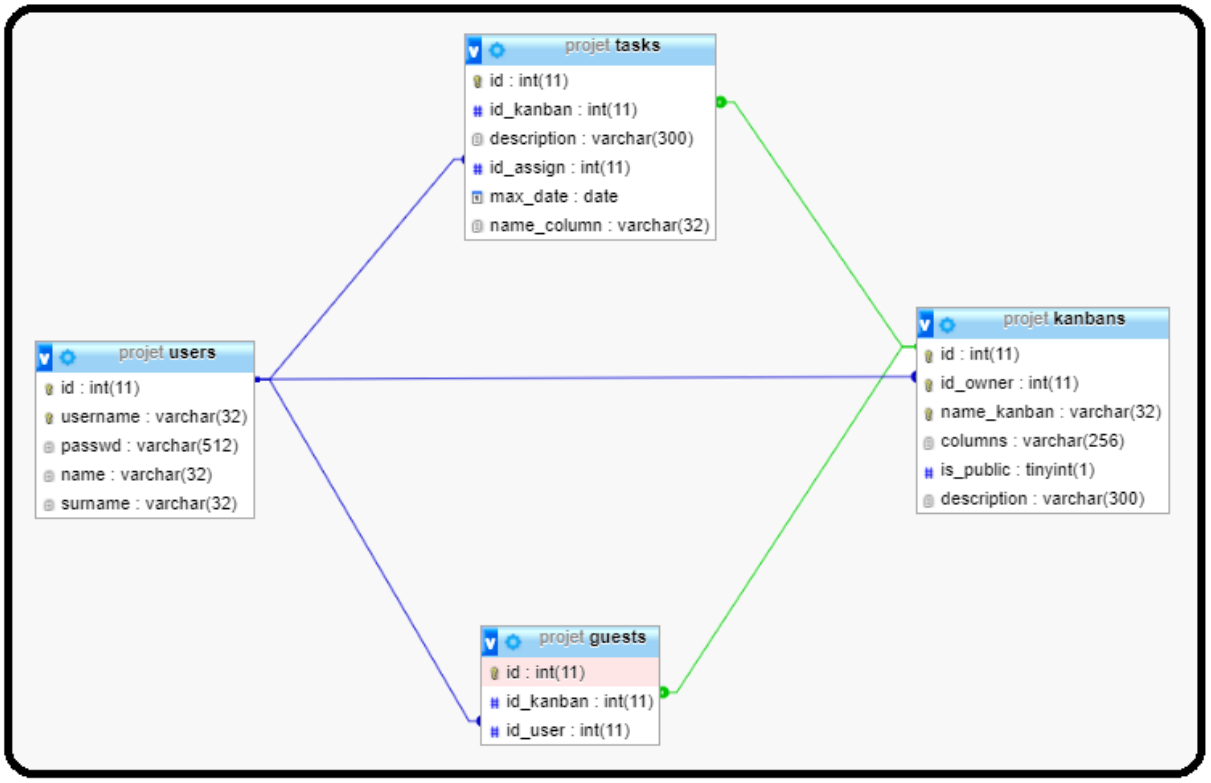
3. Côté base de données

Le côté base de données va permettre de stocker/manipuler de manière permanente les données nécessaires pour le fonctionnement de l'application web. Ces données vont être transmises côté serveur à l'aide du modèle.

La Base de Données (Modèle)

Conception

Maintenant que nous avons vu le principe du modèle MVC, nous allons expliquer comment est composée notre base de données, afin de permettre de correctement manipuler et gérer la manipulation des kanbans. Pour cela nous avons décidé d'utiliser 4 tables afin de gérer les utilisateurs ainsi que les kanbans :



1. Users

Cette table va permettre de stocker l'ensemble des données utilisateurs en contenant leur nom d'utilisateur, leur prénom, leur nom ainsi que leur mot de passe et un identifiant unique qui nous permet de faciliter les requêtes faites dans la base de données.

Nous pouvons voir que l'attribut passwd est de taille 512 afin de pouvoir stocker des hachages de mot de passe d'une taille suffisamment grande. Actuellement, les mots de

passe sont stockés en SHA-256, avec du sel afin d'éviter d'avoir deux fois le même hachage pour le même mot de passe.

2. Tasks

La table Tasks permet de stocker la liste des tâches. Chaque tâche va avoir une description, une assignation ainsi qu'une date limite de réalisation et un identifiant unique qui nous permet de faciliter les requêtes faites dans la base de données.
Bien sûr, la tâche sera liée à un kanban, ainsi qu'à une de ses colonnes. L'attribut id_kanban correspond au numéro du kanban, et est lié lors de la création de la tâche sans jamais pouvoir être modifié. L'attribut name_column correspond tant qu'à elle le nom de colonne dans laquelle la tâche aura été créée et pourra être modifiée après sa création.

3. Guests

Guests permet de gérer les invitations. Un gestionnaire peut inviter un utilisateur dans un de ses kanbans afin de lui permettre de s'attribuer et créer des tâches.

4. Kanbans

La table kanbans quant à elle va permettre de stocker l'ensemble des kanbans qui sont contenus dans l'application. Chaque kanban aura un gestionnaire, un nom, une description, des colonnes, et un attribut permettant de savoir s'il est public ou non. Nous avons aussi défini un identifiant unique qui nous permet de faciliter les requêtes faites dans la base de données.
Les colonnes sont stockées sous la forme d'une chaîne de caractères, séparée par une virgule. Les noms des colonnes ont donc échappé avant l'insertion dans la base de données afin qu'une colonne puisse contenir une virgule dans son nom

Le modèle DAO

Dans la partie précédente Le modèle MVC, nous avons vu que le côté serveur est la seule partie qui communique avec la base de données était la partie 2.1. Le modèle.

Afin de mieux structurer la partie modèle de notre MVC, nous allons utiliser un type de modèle : le modèle DAO (Data Access Object).

Ce modèle consiste à créer des objets ayant pour unique objectif de communiquer avec une partie spécifique de la base de données. Nous avons décidé de stocker l'ensemble des objets DAO dans le fichier :

/Java Resources/src/dao/

Les beans

Les Java Beans sont des classes Java qui permettent de manipuler et enregistrer l'état d'un objet. Elles contiennent des constructeurs pour instancier l'objet, des requêtes (getter) qui permettent d'encapsuler l'état de l'objet et des commandes (setter) pour changer les valeurs des attributs (propriétés) privées de l'objet.

Dans notre application nous avons utilisé trois Java Beans qui se trouve dans le paquetage *domain*:

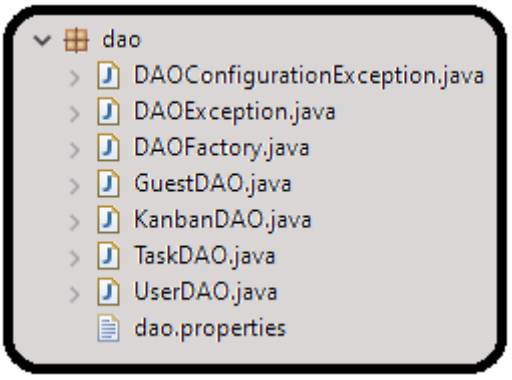
- *Kanban* : permet de manipuler/récupérer les données d'un kanban.
- *Task* : permet de manipuler/récupérer les données d'une tâche.
- *User* : permet de manipuler/récupérer les données d'un utilisateur.

Ces trois classes sont utilisées par les objets DAO, par les pages .jsp et par les contrôleurs pour changer/récupérer l'état d'un kanban/tâche/utilisateur.

Par exemple, si dans la base de données nous réalisons une requête pour récupérer un kanban nous devons mettre à jour le bean Kanban avec les informations récupérées. Ces informations vont être utilisées par les vues et les contrôleurs.

Réalisation

Afin de pouvoir déployer sans problème le site web sur n'importe quel serveur possédant une base de données, nous utilisons les objets DAO, ainsi que la page /WebContent/WEB-INF/views/configuration.jsp



Lors du déploiement du projet web, l'accès aux pages redirige automatiquement vers la page de configuration afin de configurer l'accès à la base de données. Une première vérification est faite afin de vérifier que l'accès donné est correct (scheme, address, username, password), et lors de la validation nous allons procéder à une vérification des tables.

Cette vérification est un peu particulière, elle est faite à l'aide de la classe DAOFactory, qui va être le nerf du modèle DAO. En effet, il s'agit de la classe permettant d'initialiser les différents objets DAO permettant de manipuler les requêtes vers les tables Users, Tasks, Guests et Kanbans.
Il s'agit également de la classe dans laquelle se trouvent toutes les déclarations permettant d'initialiser la base de données :

```
// Information des colonnes des tables
//      Chaque colonne est défini tel qu'un nom et un type
//      Si type == VARCHAR, alors il faut définir une taille

private static final String[][] TABLE_USER_ROWS   = {
        {"id",              "INT"},                     // Spécifique à la manipulation interne des données
        {"username",        "VARCHAR",      "32"},      // Pseudo des uitlisateurs
        {"passwd",          "VARCHAR",      "512"},
        {"name",            "VARCHAR",      "32"},
        {"surname",         "VARCHAR",      "32"}
};
private static final String[][] TABLE_KANBAN_ROWS = {
        {"id",              "INT"},                     // Spécifique à la manipulation interne des données
        {"id_owner",        "INT"},
        {"name_kanban",     "VARCHAR",      "32"},
        {"columns",         "VARCHAR",      "256"},
        {"is_public",       "BOOLEAN"},
        {"description",     "VARCHAR",      SIZE_TEXTAREA}
};
private static final String[][] TABLE_TASK_ROWS = {
        {"id",              "INT"},                     // Spécifique à la manipulation interne des données
        {"id_kanban",       "INT"},
        {"description",     "VARCHAR",      SIZE_TEXTAREA},
        {"id_assign",       "INT"},
        {"max_date",        "DATE"},
        {"name_column",     "VARCHAR",      "32"}
};
private static final String[][] TABLE_GUEST_ROWS = {
        {"id",              "INT"},
        {"id_kanban",       "INT"},
        {"id_user",         "INT"}
};
```

```java
// Contraintes des tables
private static final String[] CONS_USER_PK = {
        "ALTER TABLE users ADD CONSTRAINT PK_USER PRIMARY KEY(id, username)"
};
private static final String[] CONS_KANBAN_PK = {
        "ALTER TABLE kanbans ADD CONSTRAINT PK_KANBAN PRIMARY KEY(id, id_owner, name_kanban)",
        "ALTER TABLE kanbans ADD FOREIGN KEY FK_KANBAN_OWNER (id_owner) REFERENCES users(id)"
};
private static final String[] CONS_TASK_PK = {
        "ALTER TABLE tasks ADD CONSTRAINT PK_TASK PRIMARY KEY(id)",
        "ALTER TABLE tasks ADD FOREIGN KEY FK_TASK_KABAN (id_kanban) REFERENCES kanbans(id)",
        "ALTER TABLE tasks ADD FOREIGN KEY FK_TASK_OWNER (id_assign) REFERENCES users(id)"
};
private static final String[] CONS_GUEST_PK = {
        "ALTER TABLE guests ADD CONSTRAINT PK_GUEST PRIMARY KEY(id)",
        "ALTER TABLE guests ADD FOREIGN KEY FK_GUEST_KANBAN (id_kanban) REFERENCES kanbans(id)",
        "ALTER TABLE guests ADD FOREIGN KEY FK_GUEST_USER (id_user) REFERENCES users(id)"
};

// Liste des tables
//      Etant défini tel qu'un nom, les informations des colonnes, et des contraintes
private static final Object[][] LIST_TABLES       = {
        {TABLE_USER,    TABLE_USER_ROWS,    CONS_USER_PK},
        {TABLE_KANBAN,  TABLE_KANBAN_ROWS,  CONS_KANBAN_PK},
        {TABLE_TASK,    TABLE_TASK_ROWS,    CONS_TASK_PK},
        {TABLE_GUEST,   TABLE_GUEST_ROWS,   CONS_GUEST_PK}
};
```

Cette classe contient la déclaration de l'ensemble des tables, de leurs attributs ainsi que de leurs contraintes. A la sauvegarde de la configuration, DAOFactory va vérifier si l'ensemble des tables existent, et si l'une d'elles n'existe pas elle la crée, sinon elle vérifie que la table contient bien l'ensemble des bons attributs (même nom, même typage).

Cette configuration sera sauvegardée dans un fichier nommé "*dao.properties*" vérifiant une certaine syntaxe :

url = jdbc:mysql://*adress*:3306/*scheme*
driver = com.mysql.jdbc.Driver
user = *username*
password = *password*

Les noms soulignés correspondent aux entrées attendues lors de la configuration, et ce fichier sera utilisé à chaque création d'une instance de l'objet DAOFactory, afin de spécifier les paramètres de connexion à la base de données. Pour l'instant, le mot de passe n'est pas chiffré, mais il s'agit d'une amélioration possible de faire.

GuestDAO

Voici l'objet DAO permettant de faire les requêtes à la table Guest. Nous allons voir l'ensemble des requêtes faisables à la base de données à l'aide de cet objet :

```
public class GuestDAO {

    // ATTRIBUTS

    private static final String SQL_MAX_ID                  = String.format("SELECT max(id) FROM %s;"                          , DAOFactory.TABLE_GUEST);
    private static final String SQL_SELECT_GUEST            = String.format("SELECT id_user FROM %s WHERE id_kanban=?"         , DAOFactory.TABLE_GUEST);
    private static final String SQL_SELECT_KANBANS_GUEST    = String.format("SELECT id_kanban FROM %s WHERE id_user=?"         , DAOFactory.TABLE_GUEST);

    private static final String SQL_INSERT_GUEST            = String.format("INSERT INTO %s VALUES (?, ?, ?)"                 , DAOFactory.TABLE_GUEST);

    private static final String SQL_DELETE_KANBAN_GUEST     = String.format("DELETE FROM %s WHERE id_kanban=?"                 , DAOFactory.TABLE_GUEST);
    //private static final String SQL_DELETE_USER_GUEST       = String.format("DELETE FROM %s WHERE id_user=?"                   , DAOFactory.TABLE_GUEST);
    private static final String SQL_DELETE_USER_KANBAN_GUEST = String.format("DELETE FROM %s WHERE id_kanban=? AND id_user=?"  , DAOFactory.TABLE_GUEST);

    private DAOFactory factory;

    // CONSTRUCTEURS

    GuestDAO(DAOFactory factory) {
        this.factory = factory;
    }

    // REQUETES

    public List<User> getListGuests(Kanban kanban) throws DAOException {

    public List<String> getListGuestsUsername(Kanban kanban) throws DAOException {


    public List<Kanban> getListKanbanGuest(User user) throws DAOException {

    // COMMANDES

    public void addGuest(Kanban kanban, User user) throws DAOException {

    public void deleteKanbanGuest(Kanban kanban, User user) throws DAOException {

    public void deleteKanbanGuests(Kanban kanban) throws DAOException {

    // OUTILS

    private Long getNextId() throws DAOException{

}
```

Chaque objet DAO est constitué de la même structure, c'est-à-dire un ensemble de requêtes, et des commandes/requêtes permettant d'exécuter ses requêtes. Le constructeur de cet objet requérant la référence à un DAOFactory, il faut donc que la configuration de la connexion à la base de données ait eu lieu.

Nous analyserons uniquement cet objet car tous les autres seront de la même forme. Maintenant voyons une requête afin de voir comment elles sont faites :

```java
public List<User> getListGuests(Kanban kanban) throws DAOException {
    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    List<User> users = new ArrayList<User>();

    try {
        connection = factory.getConnection();
        preparedStatement = DAOFactory.initializePreparedRequest(
                connection,
                SQL_SELECT_GUEST,
                false,
                kanban.getId());

        resultSet = preparedStatement.executeQuery();

        UserDAO userDao = factory.getUtilisateurDao();
        while (resultSet.next()) {
            users.add(userDao.getUser(resultSet.getLong("id_user")));
        }
    } catch (SQLException e) {
        throw new DAOException(e);
    } finally {
        DAOFactory.close(resultSet, preparedStatement, connection);
    }
    return users;
}
```

Tout d'abord nous pouvons voir la requête qui permet de se connecter à la base de données. Par la suite nous initialisons la requête avec les arguments, et pour finir nous exécutons la requête.
Une fois la requête faite, nous analysons le résultat : dans ce cas, nous récupérons pour chaque ligne le numéro id_user, puis nous récupérons l'utilisateur correspond à l'aide d'une autre requête, et enfin nous ajoutons l'utilisateur à la liste pour au final renvoyer la liste une fois toutes les lignes analysées.

Avant de renvoyer le résultat, nous fermons bien les instances connectées à la base de données bien sûr.

KanbanDAO

Comme l'objet précédent, cette classe va permettre d'envoyer un ensemble de requêtes à la base de données, mais cette fois-ci pour récupérer des informations sur la table Kanban.

Nous pouvons par exemple récupérer la liste des kanbans publics, la liste des kanbans associées à un utilisateur, créer un kanban, ou en supprimer un.

Encore une fois, la construction de cet objet requiert une référence sur DAOFactory

TaskDAO

La classe TaskDAO permet de faire des requêtes dans la base de données afin de manipuler les informations liées aux tâches.

UserDAO

Pour finir, l'objet UserDAO va nous permettre de manipuler les données liées aux utilisateurs, cependant cette fois-ci la manipulation va être un peu différente. En effet, cette classe contient toujours les commandes/méthodes afin d'envoyer les requêtes à la base de données mais va être un peu différente.

En effet, lorsqu'un utilisateur crée un compte, nous n'enregistrons pas directement son mot de passe, d'abord nous allons utiliser un chiffrement :

```java
public String encodePassword(String username, String passwordToHash) throws DAOException {
    String saltString = generateSalt(12);
    passwordToHash = saltString + passwordToHash;

    String generatedPassword = null;
    try {
        MessageDigest md = MessageDigest.getInstance(DIGEST_ALGO);
        md.update(passwordToHash.getBytes());
        byte[] bytes = md.digest();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i< bytes.length; i++)
            sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
        generatedPassword = sb.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new DAOException("Instance of '" + DIGEST_ALGO + "' does not exist for MessageDigest");
    }
    return saltString + generatedPassword;
}

public String verifyPassword(String username, String passwordToHash) throws DAOException  {
    String storedPassword = null;
    try {
        storedPassword = getUserPassword(username);
    } catch (DAOException e) {
        throw e;
    }
    String salt = storedPassword.substring(0, 12);
    passwordToHash = salt + passwordToHash;

    String generatedPassword = null;
    try {
        MessageDigest md = MessageDigest.getInstance(DIGEST_ALGO);
        md.update(passwordToHash.getBytes());
        byte[] bytes = md.digest();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i< bytes.length; i++)
            sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
        generatedPassword = sb.toString();
    } catch (NoSuchAlgorithmException e) {
        throw new DAOException("Instance of '" + DIGEST_ALGO + "' does not exist for MessageDigest");
    }
    if (!(salt + generatedPassword).equals(storedPassword)) {
        return null;
    }
    return storedPassword;
}
```

Lors du traitement de l'envoi du formulaire d'inscription par l'utilisateur, la première chose qui est faite est de chiffrer le mot de passe en SHA-256 en ajoutant du sel. Ensuite nous vérifions si le nom d'utilisateur est déjà pris, et seulement s'il n'est pas déjà pris nous enregistrons son compte avec le mot de passe haché.

Lorsqu'un utilisateur essaye de se connecter à son compte, nous allons de nouveau hasher le mot de passe avec le sel présent dans le hachage du mot de passe de l'utilisateur correspondant, et nous vérifions si le hachage final est le même.

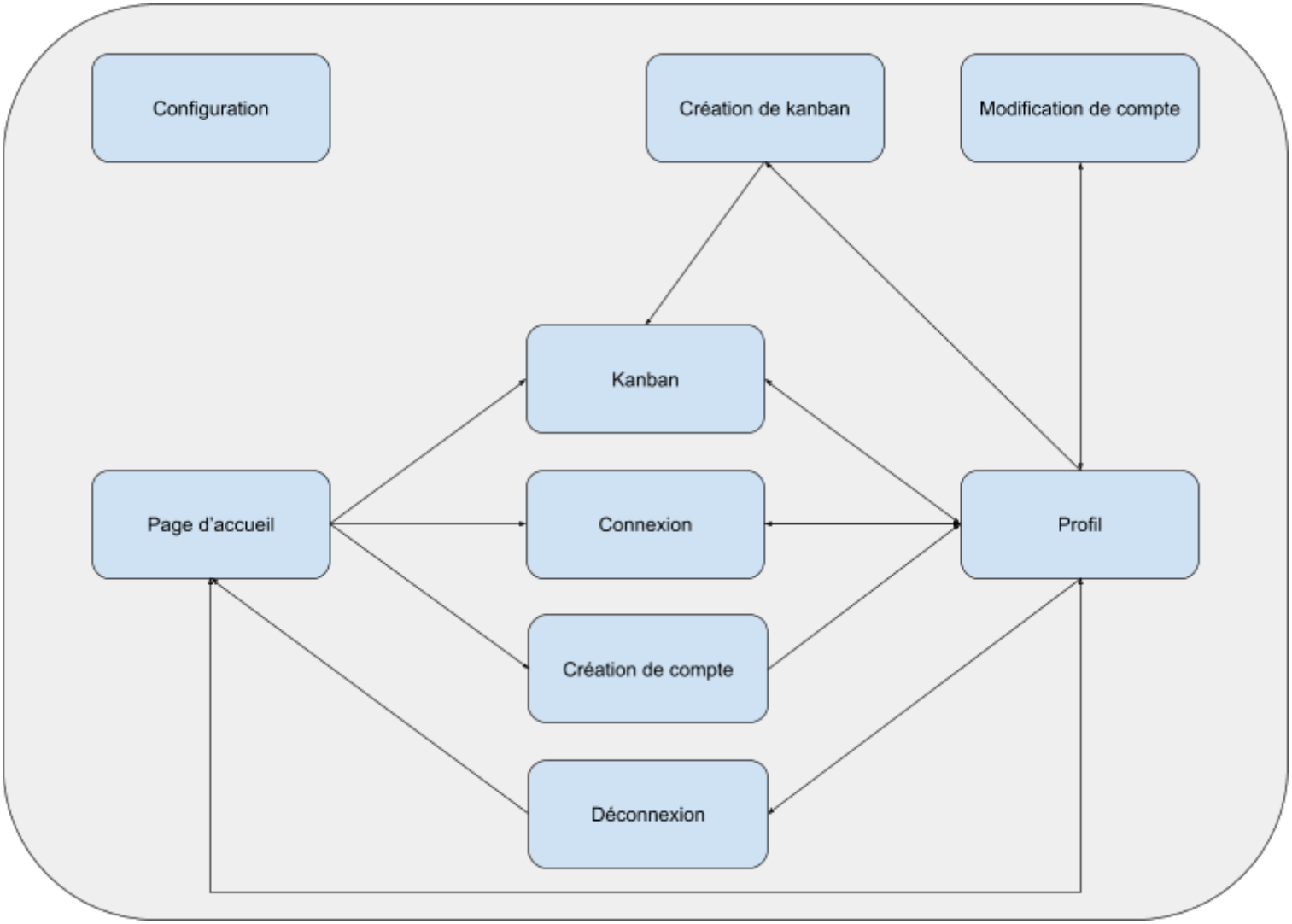
Prenons un exemple pour que ce soit plus concret :

- L'utilisateur crée un compte avec comme nom 'David' et comme mot de passe 'madrillet'
- Nous vérifions que l'utilisateur David n'est pas déjà présent
- Nous générons un sel de longueur 12 : ebc3301ee28a
- Nous hachons le mot 'ebc3301ee28amadrillet' : 8171587FE94AAAF5D91AD0B8D9D300187D675C99549BA2AC8368B9690C618D81
- Nous stockons l'utilisateur avec comme informations :
  - user : David
  - passwd : ebc3301ee28a8171587FE94AAAF5D91AD0B8D9D300187D675C99549BA2AC8368B9690C618D81

- Lorsque nous essayons de se connecter en tant que David, nous récupérons le sel : ebc3301ee28a
- Nous hachons le mot de passe qu'il a donné concaténé au sel : ebc3301ee28amadrillet
- Nous obtenons bien le même hachage : 8171587FE94AAAF5D91AD0B8D9D300187D675C99549BA2AC8368B9690C618D81
- Il s'agit bien du bon utilisateur

Les pages utilisateurs (Vue)

Les utilisateurs se connectant auront accès à différentes pages, ayant chacun une fonctionnalité différente. Voici un schéma des ensembles interactions qu'il est possible d'avoir entre les différentes pages :



Les noms sont accès explicite, mais nous allons tout de même expliquer les différentes pages ainsi que les relations.

Page d'accueil

Correspond à la page principale. N'importe quel utilisateur connecté ou non y a accès, et permet d'accéder aux kanbans publics.

Un utilisateur non connecté pourra se connecter, créer un compte ou accéder à un kanban public.

Un utilisateur connecté quant à lui aura toujours accès aux kanbans public, mais également à une page profil, et pourra également se déconnecter.

Kanban

La page kanban ne pourra pas être accédée manuellement par un utilisateur. En effet, la page est uniquement chargée en fonction du kanban sélectionné sur les autres pages.

Connexion

Cette page permet à n'importe qui de se connecter. Une fois connecté, l'utilisateur sera automatiquement redirigé vers sa page profil.

Création de compte

Cette page permet à n'importe qui de créer un compte utilisateur. Chaque utilisateur doit avoir un nom différent. Une fois son compte créé, l'utilisateur sera redirigé vers sa page profil.

Déconnexion

Il s'agit d'une page "tampon". Elle permet uniquement de se déconnecter, puis de rediriger vers la page d'accueil

Profil

La page profil est la page permettant à un utilisateur connecté d'accéder à ses différentes informations. A partir d'ici, il pourra accéder aux kanbans dont il est le gestionnaire et aux kanbans sur lesquels il est invité.

Il pourra également accéder à la page permettant de créer un nouveau kanban, et à la page permettant de modifier ses informations personnelles (nom, mot de passe, etc…)

L'utilisateur pourra également décider de se déconnecter et être redirigé vers la page d'accueil, ou uniquement aller sur la page d'accueil sans se déconnecter.

Modification de compte

Cette page permet aux utilisateurs connectés de modifier leurs informations personnelles. Si l'utilisateur n'est pas connecté, alors il sera redirigé vers la page de connexion.

Création de Kanban

Cette page porte bien son nom, et permet à n'importe quel utilisateur connecté de créer un nouveau kanban. Si l'utilisateur voulant accéder n'est pas connecté, alors il sera automatiquement dirigé vers la page de connexion.

Configuration

Cette page permet de configurer la connexion à la base de données. L'accès à cette page se fait automatiquement lorsque la connexion n'a pas encore été faite, en redirigeant toutes les pages vers celle-ci. Cela se fait à l'aide d'une méthode dans l'objet DAOFactory dans chaque méthode de chaque contrôleur :

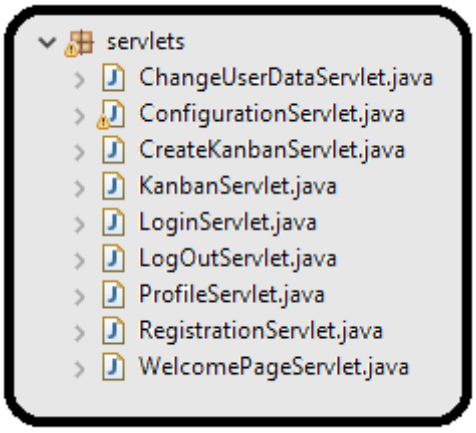
```
if (!DAOFactory.dbIsValidate()) {
response.sendRedirect(request.getContextPath() + "/configuration");
}
```

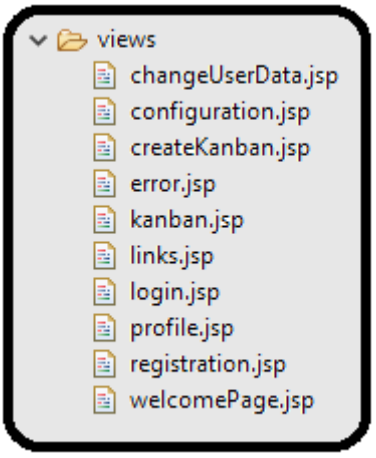
A l'aide de cette condition, aucune page ne pourra être accédée tant que la configuration n'aura pas été faite.

Liens serveur/client (Contrôleurs)

La manipulation côté serveur

Comme précisé dans la partie Le modèle MVC, le serveur envoie les données au client à l'aide de la vue, et le client récupère les données du client à l'aide des contrôleurs. Voici l'ensemble des pages utilisés par la vue, ainsi que l'ensemble des contrôleurs :





Le principe des contrôleurs va être d'être associé à une vue, afin d'exécuter certaines choses en fonction des actions de l'utilisateur, et comme expliqué dans la partie 2.1. Les contrôleurs, les servlets vont nous servir de contrôleur.

Une servlet va être une classe Java qui va permettre de créer dynamiquement des données, à partir d'une page jsp donnée. Chaque servlet est composé de multiples méthodes qui vont être appelées en fonction de la requête HTTP envoyé à la page associé à la requête. Par exemple, si une requête GET est envoyée sur la page associée à la servlet, alors la méthode doGet sera appelée. Voici un exemple de servlet associé à la page *pageServlet* et ayant défini les 2 méthodes étant appelées lors du GET et du POST :

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

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

        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

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

}
```

Dans notre projet, chaque page .jsp associée à une vue, sera également associée à une servlet permettant d'exécuter une tâche différente. Prenons en exemple la servlet associé à la page d'accueil, ayant pour vue *welcomePage.jsp*, *WelcomePageServlet.java* :

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

    // ATTRIBUTS

    private static final long serialVersionUID = 1L;

    private static final String VUE         = "/WEB-INF/views/welcomePage.jsp";
    private static final String VUE_KANBAN  = "/WEB-INF/views/kanban.jsp";

    private KanbanDAO kanbanDao;
    private TaskDAO taskDao;

    // COMMANDES

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        if (!DAOFactory.dbIsValidate()) {
            response.sendRedirect(request.getContextPath() + "/configuration");
        } else {
            request.setCharacterEncoding("UTF-8");
            Kanban kanban = kanbanDao.getKanban(Long.valueOf(request.getParameter("kanban-value")));
            request.getSession().setAttribute("kanban", kanban);
            List<Task> tasks = taskDao.getKanbanTasks(kanban);
            request.setAttribute("tasks", tasks);
            this.getServletContext().getRequestDispatcher(VUE_KANBAN).forward(request, response);
        }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        if (!DAOFactory.dbIsValidate()) {
            response.sendRedirect(request.getContextPath() + "/configuration");
        } else {
            this.kanbanDao = DAOFactory.getInstance().getKanbanDao();
            request.setAttribute("infoTest", kanbanDao);
            List<Kanban> kanbans = kanbanDao.getPublicKanbans();
            request.setAttribute("publicKanbans", kanbans);
            this.getServletContext().getRequestDispatcher(VUE).forward(request, response);
        }
    }

}
```

Sur cette servlet, nous n'avons aucune condition permettant de faire un affichage différent selon si l'utilisateur est connecté ou non, mais nous verrons plus tard comment est gérée la connexion.

La méthode doGet est utilisée lorsque l'utilisateur envoie une requête au serveur afin de récupérer la page à l'url */welcomePage*. Tout ce qui est fait est de vérifier si la configuration de la connexion à la base de données a bien été faite. Si elle est bien faite, alors nous récupérons la liste des kanbans publics de la base de données à l'aide de l'objet DAO permettant de récupérer les informations sur les kanbans.

La méthode doPost est utilisée quant à elle lorsqu'un utilisateur envoie une page au serveur, à l'aide par exemple d'un formulaire. Dans ce cas, elle est utilisée lorsque l'utilisateur sélectionne un kanban public à afficher. En choisissant un kanban, le numéro du kanban choisi est envoyé à travers un formulaire, ce numéro est récupéré, puis une information sur le kanban sélectionné et ses tâches associées sont stockées dans la variable de sessions et les attributs de requête.

La servlet “principale”

Chaque servlet va avoir une utilitée différente et utile, mais une en particulier va être au coeur de l’application : kanbanServlet, la servlet associée à la page /kanban.
En effet, c’est sur cette page où toutes les données liées aux kanbans vont être traitées. Nous avons choisi de mettre beaucoup de manipulation sur la même servlet afin d’éviter à l’utilisateur un changement et/ou un rafraîchissement du navigateur à chaque manipulation.

Sur cette page, l’utilisateur à la possibilité de créer/supprimer/modifier une tâche et inviter/supprimer un invité au kanban quand il en est le gestionnaire.
Chacune de ses fonctionnalités va être faite à l’aide d’une popup permettant à l’utilisateur de ne pas rafraîchir le navigateur pour une simple modification, mais également des popups javascript de demande de confirmation lors d’une suppression.

Le fait de mettre toutes ses fonctionnalités a pour conséquence d’obtenir une servlet qui est beaucoup plus grande que le reste des autres (400 au lieu d’environ 100 d’habitude). Afin de régler ce problème, une grande condition est faite afin de séparer chacun des cas.

Les sessions et attributs

Pour pouvoir bien gérer notre application, nous avons utilisé deux variables de sessions, une pour stocker l'utilisateur connecté et une pour stocker un kanban.
Nous initialisons la variable de session user dans les servlets LoginServlet et RegistrationServlet, donc au moment où l'utilisateur se connecte ou quand il crée un compte. Nous mettons à jour cette variable dans la servlet ChangeUserDataServlet, donc quand l'utilisateur décide de changer ses données, et nous détruisons la variable de session dans la servlet LogOutServlet, donc quand l'utilisateur se déconnecte.

Dans la variable de session kanban nous stockons le kanban sur lequel nous souhaitons avoir accès pour afficher les informations dans la page /kanban. Les kanbans publics sont stockés dans des attributs. Nous initialisons la variable de session dans la servlet CreateKanbanServlet, donc quand l’utilisateur crée un kanban. Comme ça nous pouvons avoir accès aux kanbans d’un utilisateur.

Nous utilisons aussi des attributs qui sont définis avec la méthode request.setAttribute(nom, valeur). Ces attributs sont utilisés pour enregistrer des variables dans le même cycle de requêtes et elles sont accessibles par la méthode request.getAttribute(nom). Nous utilisons des attributs pour stocker les tâches d’un kanban, pour stocker la liste des kanbans où l'utilisateur est gestionnaire, pour stocker les kanbans où l’utilisateur est invité et aussi pour stocker les informations qui vont permettre de voir si une action est réalisée avec succès ou pas.

La différence entre les attributs et les variables de session est que les variables de session sont accessibles dans toutes les pages, alors que les attributs ne sont accessibles que par les pages .jsp qui correspondent aux servlets où les attributs sont définis.

La manipulation côté client

Les manipulations côté client vont permettre de faire un affichage dynamique à l'utilisateur, mais nous permettent également de faire des manipulations cachées à l'utilisateur. Sans rentrer dans les détails, nous utilisons du javascript afin de permettre à l'utilisateur de sélectionner un affichage des informations sur sa page profile, mais également afin de créer des "popups" permettant de mettre à jour des tâches.

L'affichage de la popup se met à jour en fonction de la tâche que veut modifier l'utilisateur, et donc nous utilisons le javascript afin de ne pas charger une nouvelle page, mais plutôt modifier les informations de la popup qui va être affichée.

Difficultées

Nous avons eu des problèmes pour connecter l'application avec la base de données. Pour cela nous avons fait beaucoup de recherches pour bien comprendre comment fonctionne l'accès à une base de données depuis le langage Java. Nous avons compris que l'utilisation de l'objet DAO pour avoir accès à la base de données, est important et du coup nous avons implémenté ce principe avec les classes DAO et les Java Beans.

Nous avons aussi rencontré des problèmes avec le fichier de configuration, vu que ce fichier doit permettre l'accès à la base de données et son initialisation avec les tables nécessaires. Le fichier de configuration a également créé des erreurs côté serveur mais en faisant des tests nous avons réussi à faire fonctionner notre application sans avoir des erreurs côté serveur ou côté configuration.

Un autre problème, pas pour la difficulté mais pour le temps passé pour le gérer, était la gestion des droits d'accès pour les utilisateurs (supprimer un kanban, ajouter un kanban, supprimer, modifier, ajouter une tâche, accès à un kanban public etc). Nous avons bien défini dès le début les tables dans la base de données et les liaisons entre ces tables donc nous avons réussi à bien gérer les droits d'accès.

Fonctionnalités

Implémentation

L'application web que nous avons développée contient un ensemble de fonctionnalités. Afin que ce soit plus simple de visualiser les possibilitées d'utilisation, voici une explication détaillée de chaque fonctionnalité séparé en différents groupes, avec comme code couleur vert lorsqu'il s'agit d'une fonctionnalité demandée dans le sujet, et bleu lorsqu'il s'agit d'une fonctionnalité supplémentaire :

Informations générales:

- Gestion des permissions d'accès aux pages :
  - toutes → si la configuration a été faite
  - profile → uniquement lorsqu'un utilisateur est connecté
  - kanban → uniquement lorsqu'un utilisateur choisit un kanban
  - ChangeUserData → uniquement lorsqu'un utilisateur est connecté
- Gestions des permissions aux kanbans :
  - Invités uniquement par le gestionnaire
  - Ajout de tâches possibles que si invité ou gestionnaire
  - Modification des tâches assignées (gestionnaire accès à toutes les tâches)
  - Les visiteurs peuvent uniquement voir les kanbans publics, sans aucune interaction (juste le visuel)
- Gestion des erreurs
  - Gestion de la configuration d'accès à la base de données : redirection vers la page de configuration tant qu'elle n'as pas été correctement configurée
  - Entrée utilisateur incorrecte : message d'information à l'utilisateur (utilisateur inconnu, mauvais mot de passe, enregistrement impossible, etc…)
- Utilisations des objets vus en cours :
  - jsp
  - servlet
  - java
- Interaction et manipulation côté serveur et client, à l'aide des Servlets et du Javascript

Les comptes utilisateurs :

- Création d'un compte
- Connexion au compte
- Une page de profil

- Stockage des données privées sécurisé (hachage des mots de passe)
- Modification des données personnelles
- Affichage des kanbans ayant un lien avec l'utilisateur:
    - La liste des kanbans dont il est gestionnaire (triée par ordre lexicographique)
    - La liste des kanbans dont il est invité (triée par ordre lexicographique)
    - La liste des tâches dont-il est assigné (triée en fonction des ID)
    - La liste des tâches pour un kanban donné (triée en fonction des dates)

Les kanbans :

- Création d'un kanban
    - Description
    - nom
    - nombre de colonnes variables (entre 2 et 7)
- Suppression d'un kanban
- Ajout et suppression d'un invité au kanban
    - La suppression d'un invité retire l'ensemble de ses assignations dans les tâches du kanban

Les tâches :

- Création de tâche avec de multiples informations :
    - Description
    - Assignation
    - Date limite de réalisation
- Suppression d'une tâche
- Description avec mise en forme (texte **gras** et texte souligné)

Amélioration

Même si l'application actuelle est stable, plusieurs améliorations sont possibles afin d'avoir une application qui fonctionne parfaitement et qui possède plus de fonctionnalités.

Actuellement la configuration de la connexion à la base de données fonctionne correctement mais n'est pas totalement sécurisée. En effet, le premier point est de hasher le mot de passe, et par la suite faire une vérification avant de valider la configuration. La page de configuration comporte déjà une double confirmation permettant d'implémenter rapidement ce module (ayant déjà commencé à mettre en place ce système), il suffit donc de finir ce processus.

Nous pouvons améliorer la base de données. Avec les tables que nous avons décidé de construire, l'application va marcher correctement si le nombre d'utilisateurs n'est pas très grand mais si le nombre d'utilisateurs est assez important, les requêtes vont prendre plus de temps pour être réalisées, donc il va y avoir une latence.

Le déplacement d'une tâche est réalisé dans la popup de la modification d'une tâche. Nous pouvons améliorer cette partie en faisant glisser les tâches d'une colonne à une autre en utilisant JavaScript et la propriété *sortable* et en regardant quelle colonne a été choisie par l'utilisateur pour pouvoir mettre à jour la base de données des tâches.

Le tri des kanbans et des tâches est fait automatiquement, nous ne donnons pas à l'utilisateur la possibilité de choisir s'il veut trier les informations ou non et quel type de tri il préfère. Nous pouvons réaliser ça avec des boutons pour chaque type de tri et en utilisant JavaScript pour savoir quel bouton a été choisi par l'utilisateur et dans quelle partie pour pouvoir afficher le bon résultat.

Et pour finir, une autre amélioration est de sécuriser les injections SQL en échappant toutes les entrées utilisateurs. Vu que nous n'avons pas encore vu en cours cette partie, nous avons du mal à voir comment la mettre en place.

Compléments

Il est possible d'améliorer notre application en ajoutant plus de fonctionnalités. Nous pouvons ajouter la possibilité :

- à un utilisateur d'avoir plus de données (photo, adresse, mail etc).
- de mettre à jour un kanban, changer son nom, sa description etc (même principe comme modification d'une tâche avec une popup).
- d'envoyer un lien pour inviter des utilisateurs à rejoindre un kanban (principe de google drive).
- à un utilisateur de supprimer son compte.
- d'avoir plus d'options de mise en forme pour la description d'une tâche (couleur, italique etc).