Rapport d’Itération n°3

1. Documentation utilisateur

1.1. Mode opératoire

- **Nom du livrable :** Projet JAVA modération
- **Version : 0.3**
- **Date de livraison : 24/03/2025**
- **Description :** Les propriétaires du site r3st0.fr souhaitent se doter d’une application Java qui sera installée sur les postes de travail des modérateurs, permettant de contrôler les avis déposés et de supprimer ou de bloquer la publication des avis ne respectant pas la charte d’utilisation du site ; chaque modérateur devra s’authentifier pour pouvoir utiliser l’application.

Installation :

- Pré-requis :

  - java <= 18

  -un IDE (netBeans, vs code)

  -wamp/xamp avec la BDD importée

- Procédure d'installation :
  1. Récupérer le projet depuis le dépôt gitLab (git clone https://gitlab.com/APeixoto/modo-modo.git)
  2. Importer la BDD fournit avec le projet dans wamp/xamp
  3. Le projet est prêt à être exécuté

Choix de la technologie d’accès aux données :

Nous avons choisi d'utiliser une **couche DAO** dans notre projet pour plusieurs raisons :

- **Simplicité et contrôle** : Le DAO nous permet de gérer directement les requêtes SQL et d’avoir un contrôle total sur l'accès aux données, sans complexité supplémentaire.
- **Architecture adaptée** : L’application étant locale et monolithique, une **API REST** serait inutilement lourde car il n’y a pas de communication entre services ou clients distants.
- **Alternative à JPA** : JPA apporte de la facilité pour des projets complexes, mais il impose un certain cadre et génère des surcharges inutiles pour un projet simple. Le DAO reste plus léger et adapté ici.

- **Bonne pratique** : Le DAO respecte la séparation des responsabilités et facilite la maintenance du code.

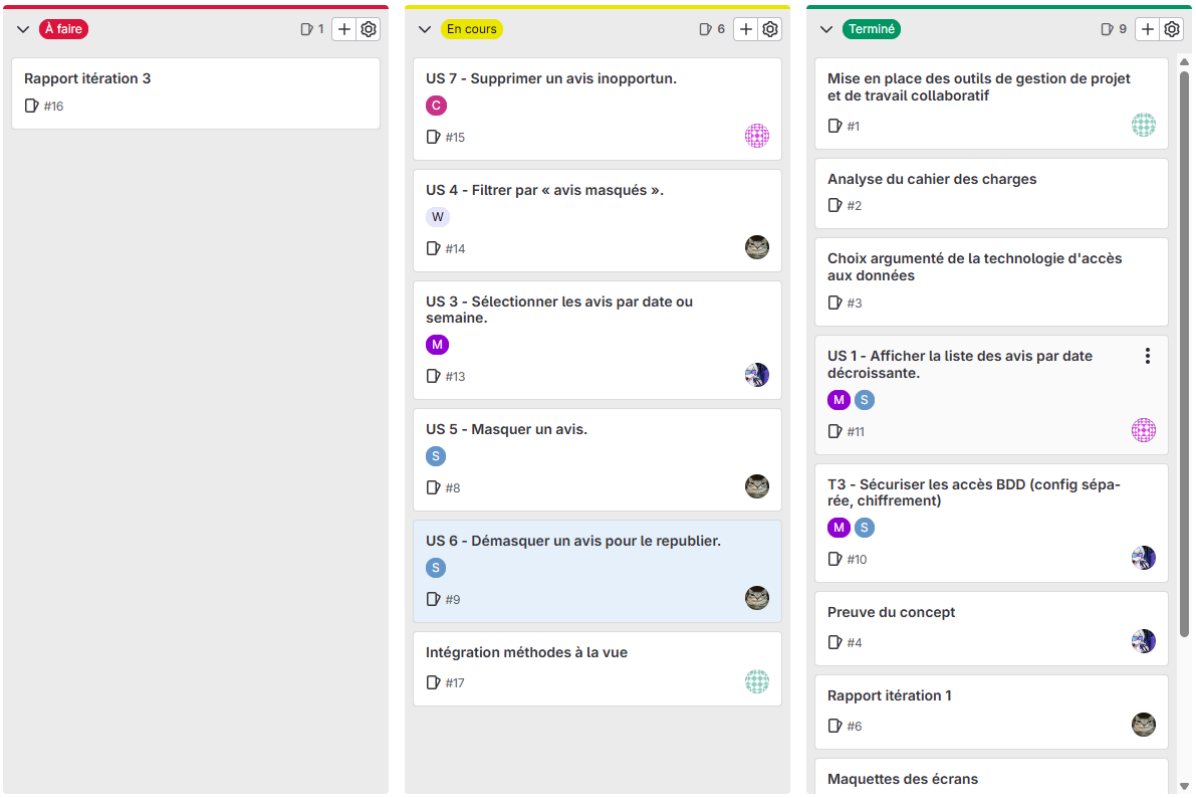
2. Documentation technique

2.1. Gestion de projet

Comptes-rendus des Daily Scrums :

Date	Participants	Avancement	Problèmes rencontrés	Prochaines actions
24/02/2025	Loric Anthony Titouan Stefen	Argumentation du choix de la technologie et P.O.C effectué	Problème de synchronisation avec le fichier restoJdbc dans le dossier « src » et « build »	Maquette à terminer, déterminer les prochains objectifs du projet
03/03/2025	Loric Anthony Titouan Stefen	Avancement sur la détermination de la priorisation des tâches	x	Commencer **US 1** et **T 3**
10/03/2025	Loric Anthony Titouan Stefen	Réorganisation des tâches prioritaires	x	Commencer **US 3**, **US 5**, **US 6, US 4** et **US 7**
17/03/2025	Loric Anthony Titouan Stefen	Organisation des tâches prioritaires	x	Commencer **US2**, **T 1**, **T 2** et **T 4**

Répartition des tâches :



Dépôt de code :

- Adresse du dépôt : [https://gitlab.com/APeixoto/modo-modo]
- Branche de travail de l'itération : [Main]

2.2. Code source

- **Branche Git associée :** [https://gitlab.com/APeixoto/modo-modo/-/tree/main/ModerationResto?ref_type=heads]

- **Évolutions du code :**

- L'objectif de l'**US4** était de filtrer une recherche par les «avis maqués» avec une nouvelle méthode, cette dernière a donc été ajoutée dans **CritiquerDAO** :
Cette méthode sélectionne donc les commentaires de la table avec la condition qu'ils soient à la valeur "0" qui correspond au fait qu'elles soient masquées.

```
public static ArrayList<String> getCommentaireMasques() throws SQLException, IOException, Exception {
    ArrayList<String> lesAvis = new ArrayList<>();
    Connection cnx = ConnexionBDD.getConnexion();
    PreparedStatement pstmt = cnx.prepareStatement("SELECT commentaire FROM critiquer WHERE date IS NOT NULL and visible = 0");
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        String unAvis = rs.getString("commentaire");
        lesAvis.add(unAvis);
    }
    return lesAvis;
}
```

- Création du test unitaire de la méthode **getCommentairesMasques** :

```
// Test 3 getCommentaireMasques
System.out.println("\n Test 3 : CritiquerDAO.getCommentaireMasques");
try {
    ArrayList<String> lesAvis = CritiquerDAO.getCommentaireMasques();
    for (int i = 0; i < lesAvis.size(); i++) {
        System.out.println(lesAvis.get(i));
    }
} catch (SQLException ex) {
    System.out.println("TestRestoDAO - échec getCommentaireMasques : " + ex.getMessage());
} catch (IOException ex) {
    System.out.println("TestRestoDAO - échec getCommentaireMasques : " + ex.getMessage());
}
```

- Ensuite la refonte des méthodes des **US5** et **US6** dans CritiquerDAO a dû être réalisée : Une méthode **setCommentaireVisible** a été ajoutée afin de mettre à jour un avis dans la table critiquer selon l'état que l'on souhaite lui attribuer, notamment grâce aux paramètres.

```
public static boolean setCommentaireVisible(int idR, int idU, int etat) throws SQLException, Exception {
Connection cnx = null;
PreparedStatement pstmt = null;
boolean updated = false;

try {
    cnx = ConnexionBDD.getConnexion();
    pstmt = cnx.prepareStatement("UPDATE critiquer SET visible = ? WHERE idR = ? AND idU = ?");
    pstmt.setInt(1, etat);
    pstmt.setInt(2, idR);
    pstmt.setInt(3, idU);

    int rowsUpdated = pstmt.executeUpdate();
    updated = rowsUpdated > 0; // Si au moins une ligne a été mise à jour

} catch (SQLException e) {
    e.printStackTrace(); // À remplacer par un bon log dans une vraie application
}
return updated;
}
```

- Afin de tester efficacement cette méthode, une seconde a été créée : **getVisibiliteCommentaire** dans le but de savoir l'état de l'avis, s'il est visible ou non.

```
public static int getVisibiliteCommentaire(int idR, int idU) throws SQLException, Exception {
Connection cnx = ConnexionBDD.getConnexion();
PreparedStatement pstmt = cnx.prepareStatement("SELECT visible FROM critiquer WHERE idR = ? AND idU = ?");
pstmt.setInt(1, idR);
pstmt.setInt(2, idU);

ResultSet rs = pstmt.executeQuery();
int visibilite = -1; // Valeur par défaut si aucun avis trouvé

if (rs.next()) {
    visibilite = rs.getInt("visible");
}
return visibilite;
}
```

- Ces deux nouvelles méthodes ont donc été accompagnées par la création de leur test unitaire respectif dans lequel il est récupéré et affiché l'état actuel de l'avis pour ensuite le modifier dans l'état opposé, pour afficher son nouvel état :

```java
// Test 4 setCommentaireVisible
System.out.println("\n Test 4 : CritiquerDAO.setCommentaireVisible");
try {
    int idR = 1;
    int idU = 3;

    // Vérifier l'état actuel du commentaire
    int etatActuel = CritiquerDAO.getVisibiliteCommentaire(idR, idU);
    System.out.println("État actuel du commentaire (idR=" + idR + ", idU=" + idU + ") : " + etatActuel);

    // Changer la visibilité (si visibilité est 0 alors mettre 1 et inversement)
    int nouvelEtat = (etatActuel == 1) ? 0 : 1;
    boolean updated = CritiquerDAO.setCommentaireVisible(idR, idU, nouvelEtat);

    if (updated) {
        System.out.println("✅ Visibilité mise à jour avec succès !");
    } else {
        System.out.println("❌ Échec de la mise à jour de la visibilité.");
    }

    // Vérifier après mise à jour
    int etatApres = CritiquerDAO.getVisibiliteCommentaire(idR, idU);
    System.out.println("Nouvel état du commentaire : " + etatApres);

} catch (SQLException | IOException ex) {
    System.out.println("TestRestoDAO - échec setCommentaireVisible : " + ex.getMessage());
}
```

- L'objectif de l'**US7** était de pouvoir supprimer un avis avec une nouvelle méthode, cette dernière a donc été ajoutée dans **CritiquerDAO** :
Cette méthode supprime un avis selon les paramètres donnés lors de l'appel de cette dernière.

```java
/**
 * Méthode permettant de supprimer un avis en ayant l'idResto et l'idUtilisateur liés à l'avis.
 * @param idResto
 * @param idUtilisateur
 * @throws SQLException
 * @throws IOException
 * @throws Exception
 */
public static void supprimerAvis(int idResto, int idUtilisateur) throws SQLException, IOException, Exception {
    Connection cnx = ConnexionBDD.getConnexion();

    // Requête SQL pour supprimer un avis spécifique
    String sql = "DELETE FROM critiquer WHERE idR = ? AND idU = ?";

    PreparedStatement pstmt = cnx.prepareStatement(sql);
    pstmt.setInt(1, idResto);
    pstmt.setInt(2, idUtilisateur);

    int rowsAffected = pstmt.executeUpdate();

    if (rowsAffected > 0) {
        System.out.println("La suppression a bien été effectuée.");
    } else {
        System.out.println("Aucun avis correspondant trouvé.");
    }

}
```

- Cette méthode a été accompagnée par l'ajout d'une méthode qui insère un avis dans la table pour aider lors des tests :

```java
public static void ajouterAvis(int idResto, int idUtilisateur, int note, String commentaire, Date date) throws SQLException, IOException, Exception {
Connection cnx = ConnexionBDD.getConnexion();

// Requête SQL pour insérer un nouvel avis
String sql = "INSERT INTO critiquer (idR, idU, note, commentaire, date, visible) VALUES (?, ?, ?, ?, ?, ?)";

PreparedStatement pstmt = cnx.prepareStatement(sql);
pstmt.setInt(1, idResto);
pstmt.setInt(2, idUtilisateur);
pstmt.setInt(3, note);
pstmt.setString(4, commentaire);
pstmt.setDate(5, (java.sql.Date) date);
pstmt.setBoolean(6, true); // L'avis est visible par défaut

int rowsAffected = pstmt.executeUpdate();

if (rowsAffected > 0) {
    System.out.println("L'avis a bien été ajouté.");
} else {
    System.out.println("Échec de l'ajout de l'avis.");
}

}
```

- La création de ces deux méthodes a également été suivi par la création du test unitaire : Le test procède d'abord à une suppression du commentaire à ajouter pour être certain du fonctionnement de la suppression qui suit pour ensuite le supprimer.

```java
//Test 3 supprimerAvis et ajouterAvis
System.out.println("\n Test 3 : CritiquerDAO.supprimerAvis, CritiquerDAO.ajouterAvis");

// Définition des valeurs de test
int idResto = 4; // Remplace par un ID existant
int idUtilisateur = 2; // Remplace par un ID utilisateur existant
int note = 4;
String commentaire = "Test de création et suppression d'un avis";
java.sql.Date date = java.sql.Date.valueOf("2024-03-17");

try {
    // Supprime l'avis existant avant d'en insérer un nouveau
    CritiquerDAO.supprimerAvis(idResto, idUtilisateur);

    // Ajouter un avis
    CritiquerDAO.ajouterAvis(idResto, idUtilisateur, note, commentaire, date);

    // Supprimer l'avis
    CritiquerDAO.supprimerAvis(idResto, idUtilisateur);
} catch (Exception e) {
    System.err.println("Erreur lors du test : " + e.getMessage());
    e.printStackTrace();
}
```

- L'objectif de l'**US3** était de sélectionner des avis par date ou semaine avec des nouvelles méthodes, ces dernières ont donc été ajoutées dans **CritiquerDAO** :
Cette méthode recherche les avis selon la date spécifiée lors de son appel grâce au paramètre.

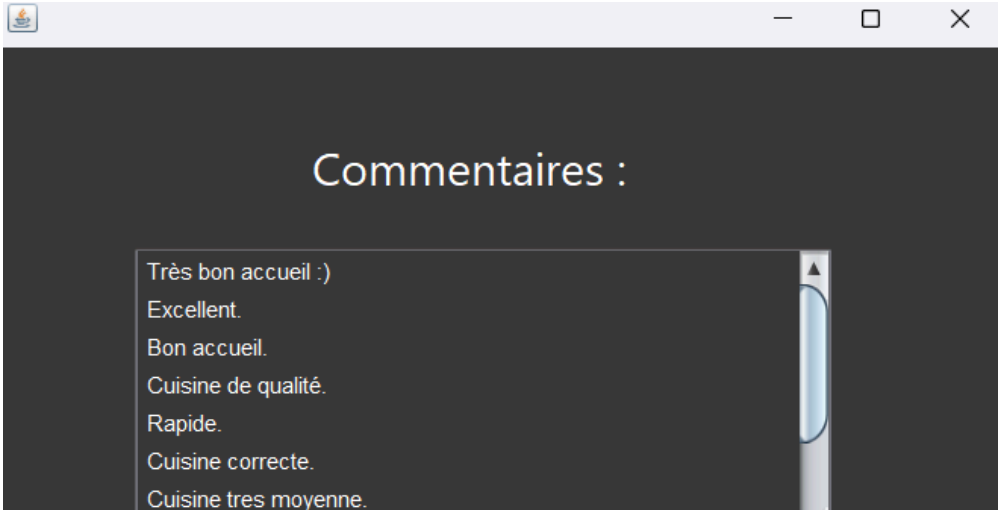
```java
public static ArrayList<Critiquer> getCommentairesByDate(String dateStr) throws SQLException, IOException, Exception {
    ArrayList<Critiquer> lesAvis = new ArrayList<>();
    Connection cnx = ConnexionBDD.getConnexion();
    PreparedStatement pstmt = cnx.prepareStatement("SELECT * FROM critiquer WHERE DATE(date) = ?");
    pstmt.setDate(1, Date.valueOf(dateStr));
    ResultSet rs = pstmt.executeQuery();

    while (rs.next()) {
        int idR = rs.getInt("idR");
        int idU = rs.getInt("idU");

        Resto unResto = RestoDAO.getOneById(idR);
        Utilisateur unUtilisateur = UtilisateurDAO.getOneById(idU);

        Critiquer uneCritique = new Critiquer(
            unResto,
            rs.getInt("note"),
            rs.getString("commentaire"),
            unUtilisateur,
            rs.getDate("date"),
            rs.getInt("visible")
        );

        lesAvis.add(uneCritique);


    }

    return lesAvis;
}
```

- Un nouveau test a été ajouté dans le même but que les précédents, voir si cette méthode fonctionne correctement :
Le test utilise donc la méthode avec un paramètre, une date, afin de recherche l'avis qui la possède.

```java
// Test 3 getCommentairesByDate
System.out.println("\nTest 3 : CritiquerDAO.getCommentairesByDate");
try {
    ArrayList<Critiquer> lesAvis = CritiquerDAO.getCommentairesByDate("2024-03-01");
    for (Critiquer avis : lesAvis) {
        System.out.println(avis); // Affiche les détails de l'avis
    }
} catch (SQLException ex) {
    System.out.println("TestCritiquerDAO - échec getCommentairesByDate : " + ex.getMessage());
} catch (IOException ex) {
    System.out.println("TestCritiquerDAO - échec getCommentairesByDate : " + ex.getMessage());
}
```

- Création des interfaces graphiques :

L'interface graphique de base ressemble à ceci pour le moment :



```
    }

    private void formWindowOpened(java.awt.event.WindowEvent evt) {
        getContentPane().setBackground(new Color(59, 56, 55));


        try {
                List<String> commentaires = CritiquerDAO.getCommentaireByDateNoMasq();

                // Création d'un modèle de liste
                DefaultListModel<String> model = new DefaultListModel<>();

                // Ajout des commentaires au modèle
                for (String commentaire : commentaires) {
                    model.addElement(commentaire);
                }

                // Application du modèle à la JList
                jListResto.setModel(model);

            } catch (IOException ex) {
                Logger.getLogger(Moderateur.class.getName()).log(Level.SEVERE, null, ex);
            } catch (Exception ex) {
                Logger.getLogger(Moderateur.class.getName()).log(Level.SEVERE, null, ex);
            }
    }
```

Comme on peut le voir, la fenêtre récupère via le DAO la liste de tous les commentaires y compris ceux qui ne sont pas masqués vu qu'on est dans la vue du modérateur. Si nous cliquons sur un des éléments de la liste nous lançons cet évènement :

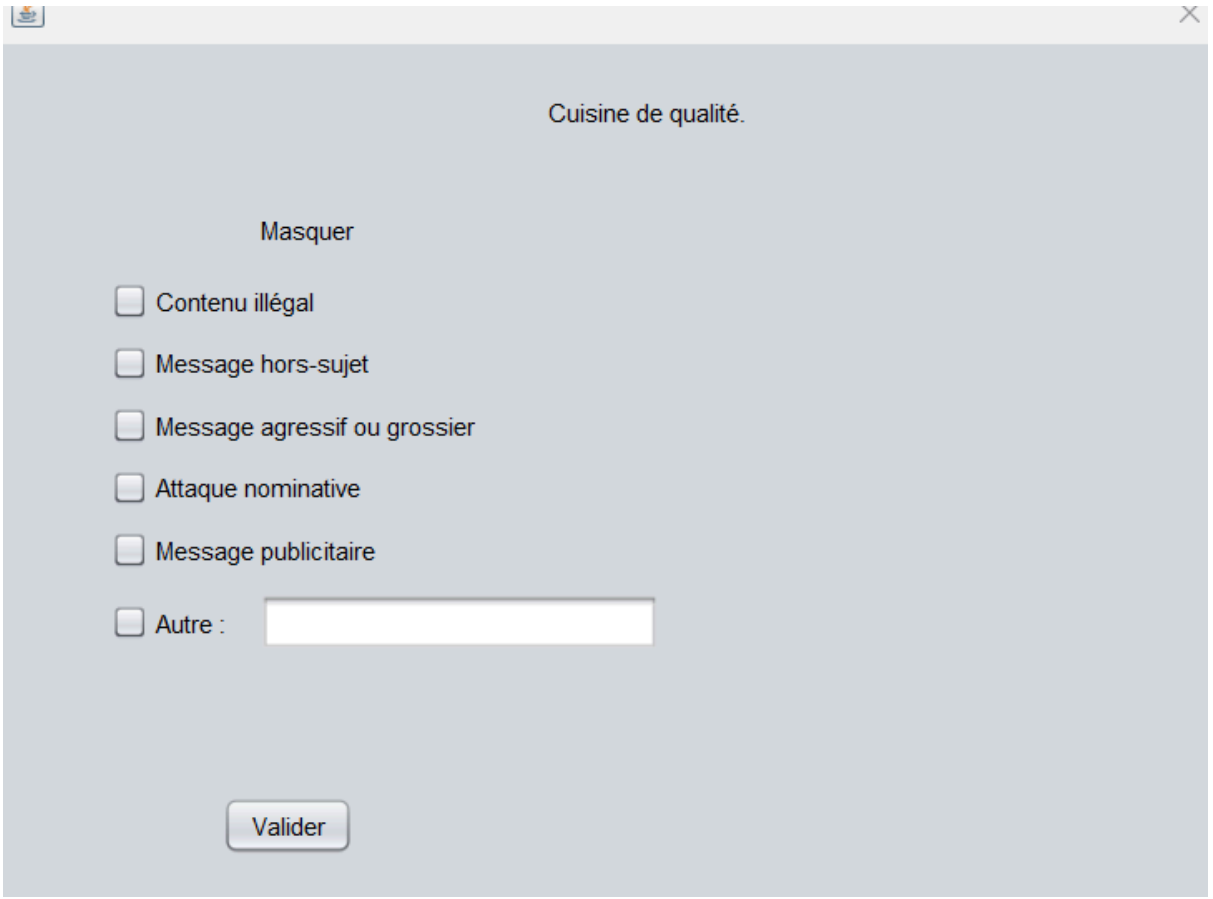
```
private void jListRestoMouseClicked(java.awt.event.MouseEvent evt) {
CommentaireSettings unCommentaireSettings;
    try {
        unCommentaireSettings = new CommentaireSettings(this, true, CritiquerDAO.getAll().get(jListResto.getSelectedIndex()));
        unCommentaireSettings.setVisible(true);
    } catch (IOException ex) {
        Logger.getLogger(Moderateur.class.getName()).log(Level.SEVERE, null, ex);
    } catch (Exception ex) {
        Logger.getLogger(Moderateur.class.getName()).log(Level.SEVERE, null, ex);
    }

}
```

Cela créer un jDialog nommé CommentaireSettings et qui lui envoie l'objet critique choisi par le click via le selectedIndex. Faut savoir qu'un jDialog de base ne propose pas d'envoyer un objet en tant que paramètre, pour cela, je créer une surcharge de constructeur et un nouvel attribut pour qu'on puisse se le permettre. Ce qui ressemble à ceci :

```
public class CommentaireSettings extends javax.swing.JDialog {

    private Critiquer uneCritique;
    /**
     * Creates new form CommentaireSettings
     */
    public CommentaireSettings(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }
    public CommentaireSettings(java.awt.Frame parent, boolean modal, Critiquer uneCritique) {
        super(parent, modal);
        initComponents();
        this.uneCritique = uneCritique;
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
```

On peut voir que le second constructeur prend en charge maintenant un objet Critiquer.

Ainsi, quand on clique sur un item de la jList nous arrivons sur ce jDialog ci-dessous :



Celui-ci récupère l'affichage du commentaire ciblé et propose la possibilité (pour l'instant) de masquer celui-ci ainsi que sélectionner la raison. Pour l'instant, la fonctionnalité n'est pas implémentée mais sera faite une fois que les codes seront fusionnés à la prochaine séance. On ajoutera également un nouveau champ "Raison" dans la base de données dans la table Critiquer.

2.4. Tests unitaires et d'intégration

Jeux d'essai :

- Test de la méthode **getCommentaireMasques** issue de la classe **CritiquerDAO** :
Pour les deux prochains tests, voici l'état de la table critiquer :
Quelques avis ont expressément été modifiés en visible "0".

	idR	note	commentaire	idU	date	visible
Éditer Copier Supprimer	1	3	moyen	2	2024-03-01	1
Éditer Copier Supprimer	1	3	Très bonne entrecote, les frites sont maisons et d...	3	2024-03-02	0
Éditer Copier Supprimer	1	4	Très bon accueil.	5	2024-03-03	1
Éditer Copier Supprimer	1	4	5/5 parce que j'aime les entrecotes	6	2024-03-04	1
Éditer Copier Supprimer	1	5	*NULL*	7	*NULL*	1
Éditer Copier Supprimer	2	2	bof.	2	2024-03-06	1
Éditer Copier Supprimer	2	1	À éviter...	3	2024-03-07	1
Éditer Copier Supprimer	2	1	Cuisine tres moyenne.	5	2024-03-08	1
Éditer Copier Supprimer	2	5	*NULL*	6	*NULL*	1
Éditer Copier Supprimer	4	5	*NULL*	3	*NULL*	1
Éditer Copier Supprimer	4	5	Rapide.	5	2024-03-11	1
Éditer Copier Supprimer	5	3	Cuisine correcte.	5	2024-03-09	1
Éditer Copier Supprimer	6	4	Cuisine de qualité.	5	2024-03-13	1
Éditer Copier Supprimer	7	4	Bon accueil.	1	2024-03-14	1
Éditer Copier Supprimer	7	*NULL*	*NULL*	3	*NULL*	1
Éditer Copier Supprimer	7	5	Excellent.	5	2024-03-15	0
Éditer Copier Supprimer	8	1	*NULL*	6	*NULL*	1
Éditer Copier Supprimer	8	4	*NULL*	7	*NULL*	1
Éditer Copier Supprimer	9	4	Très bon accueil :)	3	2024-03-18	0

- Résultat du test :
Il est retourné seulement les avis masqués.

```
 Test 3 : CritiquerDAO.getCommentaireMasques
Très bonne entrecote, les frites sont maisons et delicieuses.
Excellent.
Très bon accueil :)
```

- Test de la méthode **setCommentaireVisible** issue de la classe **CritiquerDAO** :
L'état initial de cet avis était "0" et a été modifié en "1".

```
 Test 4 : CritiquerDAO.setCommentaireVisible
État actuel du commentaire (idR=1, idU=3) : 0
✅ Visibilité mise à jour avec succès !
Nouvel état du commentaire : 1
```

- Voici l'état de la table après l'opération :
L'avis de la seconde ligne a subi un changement.

	idR	note	commentaire	idU	date	visible
Éditer Copier Supprimer	1	3	moyen	2	2024-03-01	1
Éditer Copier Supprimer	1	3	Très bonne entrecote, les frites sont maisons et d...	3	2024-03-02	1
Éditer Copier Supprimer	1	4	Très bon accueil.	5	2024-03-03	1
Éditer Copier Supprimer	1	4	5/5 parce que j'aime les entrecotes	6	2024-03-04	1
Éditer Copier Supprimer	1	5	*NULL*	7	*NULL*	1
Éditer Copier Supprimer	2	2	bof.	2	2024-03-06	1
Éditer Copier Supprimer	2	1	À éviter...	3	2024-03-07	1
Éditer Copier Supprimer	2	1	Cuisine tres moyenne.	5	2024-03-08	1
Éditer Copier Supprimer	2	5	*NULL*	6	*NULL*	1
Éditer Copier Supprimer	4	5	*NULL*	3	*NULL*	1
Éditer Copier Supprimer	4	5	Rapide.	5	2024-03-11	1
Éditer Copier Supprimer	5	3	Cuisine correcte.	5	2024-03-09	1
Éditer Copier Supprimer	6	4	Cuisine de qualité.	5	2024-03-13	1
Éditer Copier Supprimer	7	4	Bon accueil.	1	2024-03-14	1
Éditer Copier Supprimer	7	*NULL*	*NULL*	3	*NULL*	1
Éditer Copier Supprimer	7	5	Excellent.	5	2024-03-15	0
Éditer Copier Supprimer	8	1	*NULL*	6	*NULL*	1
Éditer Copier Supprimer	8	4	*NULL*	7	*NULL*	1
Éditer Copier Supprimer	9	4	Très bon accueil :)	3	2024-03-18	0

- Test de la méthode **ajouterAvis** issue de la classe **CritiquerDAO** :
Comme énoncé précédemment, l'avis est d'abord supprimé, s'il existe, pour être certain de sa création qui suit pour le supprimer correctement.

```
 Test 3 : CritiquerDAO.supprimerAvis, CritiquerDAO.ajouterAvis
Aucun avis correspondant trouvé.
L'avis a bien été ajouté.
La suppression a bien été effectuée.
```

- Test de la méthode **getCommentairesByDate** issue de la classe **CritiquerDAO** :
Le test retourne le ou les avis qui correspondent à la date donnée lors de l'appel de la méthode.

```
Test 3 : CritiquerDAO.getCommentairesByDate
Critiquer{idR=l'entrepote, note=3, commentaire='moyen', idU=drskott, date=2024-03-01, visible=l}
```

2.4. Priorisation des tâches

Estimation des complexités manquantes :

ID		Description	Criticité	Complexité (estimée)

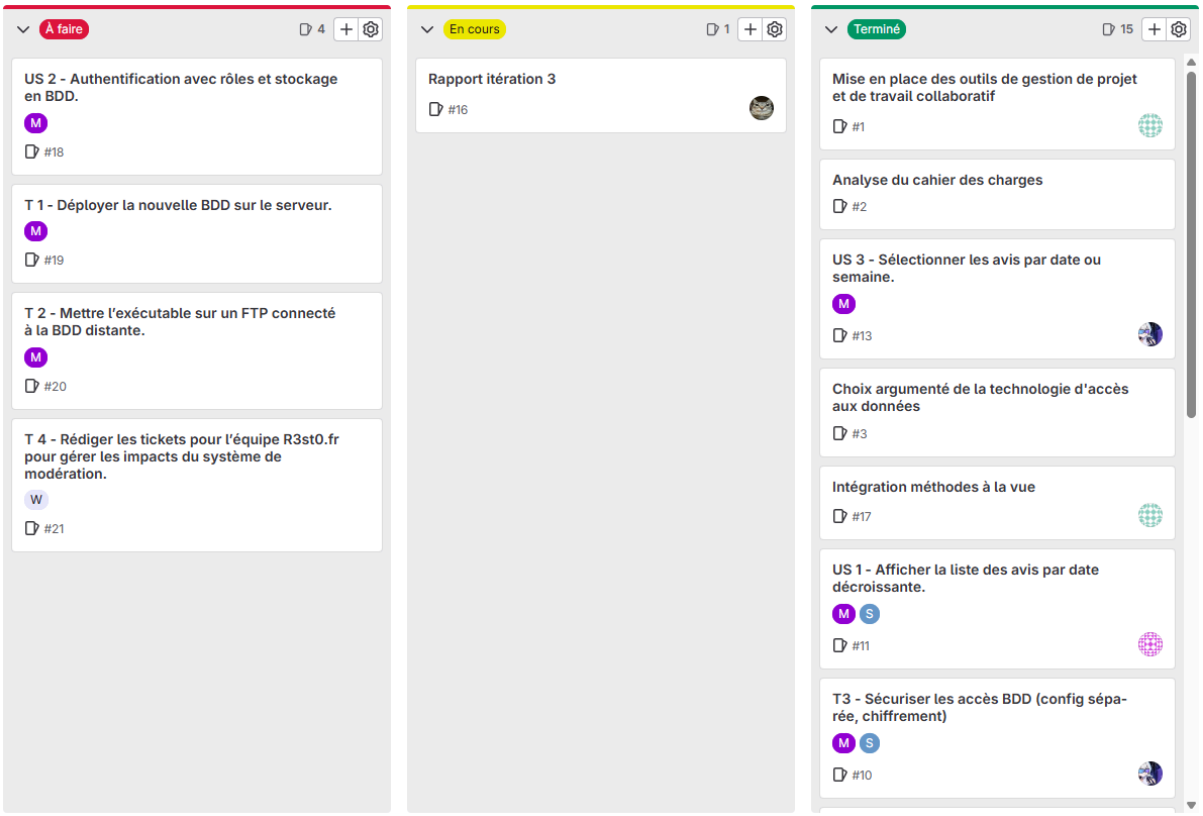
US 1		Afficher la liste des avis par date décroissante.	M	S
US 2		Authentification avec rôles et stockage en BDD.	M	L
US 3		Sélectionner les avis par date ou semaine.	M	M
US 4		Filtrer par « avis masqués ».	W	S
US 5		Masquer un avis.	S	S
US 6		Démasquer un avis pour le republier.	S	S
US 7		Supprimer un avis inopportun.	C	S
T 1		Déployer la nouvelle BDD sur le serveur.	M	S
T 2		Mettre l’exécutable sur un FTP connecté à la BDD distante.	M	M
T 3		Sécuriser les accès BDD (config séparée, chiffrement).	S	M
T 4		Rédiger les tickets pour l’équipe R3st0.fr pour gérer les impacts du système de modération.	W	M

A faire :

- **Itération 3** : Enchaîner avec **US2**, **T 1**, **T 2** et **T 4**.

2.5. Bilan de l’itération

- **Travail réalisé :**



- Les tâches **US3**, **US4** et **US7**ont été réalisées avec succès. Quant aux **US5** et **US6**, elles ont bien été retravaillées pour correspondre à ce qui était attendu.
Les tâches qui devaient absolument être complétées sur cette itération sont faites et nous avons pu même en faire une qui était destinée à l’itération suivante.

CAPTURE TABLEAUX TICKETS FIN ITÉRATION

- **Travail restant à faire :**

- Le travail de cette itération est fait, il nous faut donc poursuivre sur la prochaine.

- **Difficultés rencontrées :**

- Aucun problème pertinent retenu durant cette itération.