Université Jean Monnet — L3 Informatique

UE Développement Web 2

CollabDocs

Plateforme d'édition collaborative
de documents en temps réel

Rapport final de projet

Rachid Ghodbane

Anas Ben Abou

Yacine Addadi

Encadrants : Émilie Samuel, Hadi El Zein

Année universitaire 2025–2026

Table des matières

Introduction

Ce rapport présente **CollabDocs**, une application web d'édition collaborative de documents en temps réel développée dans le cadre de l'UE Développement Web 2. L'application répond à l'objectif central de l'énoncé : permettre à plusieurs utilisateurs, chacun depuis son propre navigateur, de visualiser et de modifier *simultanément* un même document partagé, les modifications de chacun se répercutant immédiatement chez les autres.

Le projet a été réalisé exclusivement avec les technologies imposées par l'UE : Java, servlets, JDBC sur une base MySQL/MariaDB, JSP/EL/JSTL, WebSockets et JavaScript, le tout structuré selon le patron Modèle–Vue–Contrôleur (MVC). Au-delà des fonctionnalités minimales attendues, CollabDocs propose huit types de documents éditables et un mécanisme d'historique avec restauration soumise à approbation. Ce document décrit l'application, son architecture, sa base de données, son fonctionnement collaboratif, sa gestion de la sécurité, les éléments extérieurs utilisés, ainsi que les limites identifiées et la répartition du travail au sein du groupe.

1 Présentation générale du projet

CollabDocs est une plateforme web sur laquelle un utilisateur peut créer des documents, les partager avec d'autres utilisateurs et les éditer à plusieurs en direct. Chaque document possède un *type* qui détermine l'éditeur spécialisé chargé pour le manipuler. L'application gère huit types de documents :

— **Code** — éditeur de code source avec coloration syntaxique développée par nos soins ;
— **Pixel Art** — grille de pixels dessinée sur un canevas HTML5 ;
— **Texte riche** — document mis en forme (gras, listes, titres) ;
— **Tableur** — feuille de calcul avec formules arithmétiques ;
— **Présentation** — diaporama composé de diapositives ;
— **Planning** — diagramme de Gantt avec tâches datées ;
— **Carte mentale** — arbre de nœuds reliés, en SVG ;
— **Diagramme** — graphe de nœuds et d'arêtes (organigramme).

L'énoncé n'imposait qu'un seul type de document collaboratif ; nous avons fait le choix d'en proposer plusieurs afin de couvrir des familles variées (texte, image matricielle, tableur, présentation, planning, graphe). Le cœur collaboratif — transport temps réel, présence, chat, historique, gestion des droits — est mutualisé : il est identique quel que soit le type de document, seul l'éditeur spécialisé change.

Le parcours type d'un utilisateur est le suivant : il s'inscrit ou se connecte, arrive sur sa page d'accueil qui liste ses documents et ceux qui lui sont partagés, crée un nouveau document en choisissant son type et son mode d'accès, puis ouvre l'éditeur correspondant. Toute personne disposant des droits requis peut alors rejoindre le même document et collaborer en direct.

2 Fonctionnalités de l'application

2.1 Fonctionnalités de base demandées

L'ensemble des fonctionnalités minimales de l'énoncé sont implémentées :

— **Visualisation simultanée** du même document par plusieurs utilisateurs, chacun via son navigateur ;
— **Modification en temps réel** : toute modification est propagée immédiatement aux autres clients connectés au même document ;
— **Droits différenciés** : sur l'application (visiteur, inscrit, administrateur) et sur les documents (aucun accès, lecture seule, lecture et écriture) ;
— **Modes d'accès** aux documents : public, par lien de partage, ou privé, avec en complément une protection facultative par mot de passe ;
— **Chargement et sauvegarde** : un document existant est rechargé depuis la base à l'ouverture, et son contenu peut être sauvegardé ;

— **Communication entre utilisateurs** : un chat persistant est associé à chaque document, ses messages étant stockés en base.

2.2 Fonctionnalités complémentaires

Plusieurs fonctionnalités ont été ajoutées au-delà du socle minimal :

— **Historique des versions** : chaque sauvegarde archive l'état du document ; l'historique est consultable depuis l'éditeur.

— **Restauration avec approbation** : le propriétaire peut restaurer directement une version ; un collaborateur en écriture, lui, soumet une *demande* de restauration que le propriétaire approuve ou rejette depuis une page de comparaison dédiée. La décision est notifiée en temps réel au demandeur.

— **Présence** : le nombre d'utilisateurs connectés et les notifications d'arrivée/départ sont affichés en direct.

— **Panneau d'administration** : gestion des comptes (changement de rôle, suppression), gestion des documents et statistiques globales.

— **Confort d'usage** : reconnexion WebSocket automatique avec délai d'attente croissant, sauvegarde au raccourci `Ctrl+S`, modification du titre en direct, défilement automatique du chat.

3 Architecture technique

3.1 Organisation Modèle–Vue–Contrôleur

L'application respecte le patron MVC. Le code Java est organisé en paquets fonctionnels clairement séparés :

— **Modèle** (`model`) — classes POJO représentant les entités métier : `User`, `Document`, `Message`, `Permission`.

— **Accès aux données** (`dao`) — couche d'accès JDBC : `UserDAO`, `DocumentDAO`, `PermissionDAO`, `MessageDAO`, `AccessTokenDAO`, `RestoreRequestDAO`, ainsi que `DBConnection` pour la connexion à la base.

— **Contrôleur** (`controller`) — servlets traitant les requêtes HTTP : `IndexServlet`, `LoginServlet`, `RegisterServlet`, `LogoutServlet`, `HomeServlet`, `DocumentServlet`, `EditorServlet`, `DocumentUnlockServlet`, `RestoreServlet`, `AdminServlet`.

— **Vue** (`WEB-INF/views`) — pages JSP utilisant EL et JSTL, sans *scriptlets* Java, organisées en gabarits réutilisables (en-tête, pied de page) et en fragments d'éditeurs spécialisés.

— **Temps réel** (`websocket`) — l'*endpoint* `DocumentWebSocket` et son configurateur `HttpSessionConfigurator`.

— **Transverse** — `filter/AuthFilter` (filtre d'authentification) et `util/PasswordUtil` (hachage des mots de passe).

Le flux d'une requête classique est le suivant : une servlet (Contrôleur) reçoit la requête HTTP, interroge la base via les DAO qui renvoient des objets du Modèle, place les données dans la requête, puis délègue l'affichage à une page JSP (Vue) qui ne contient aucune logique métier. La figure 1 résume cette organisation.

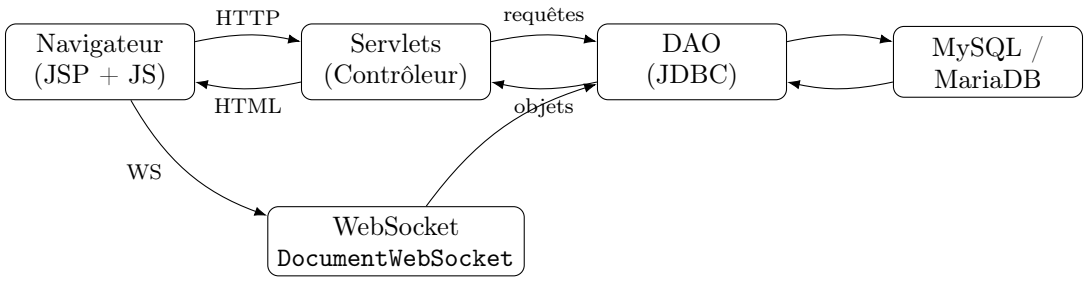

FIGURE 1 – Architecture générale et circulation des données.

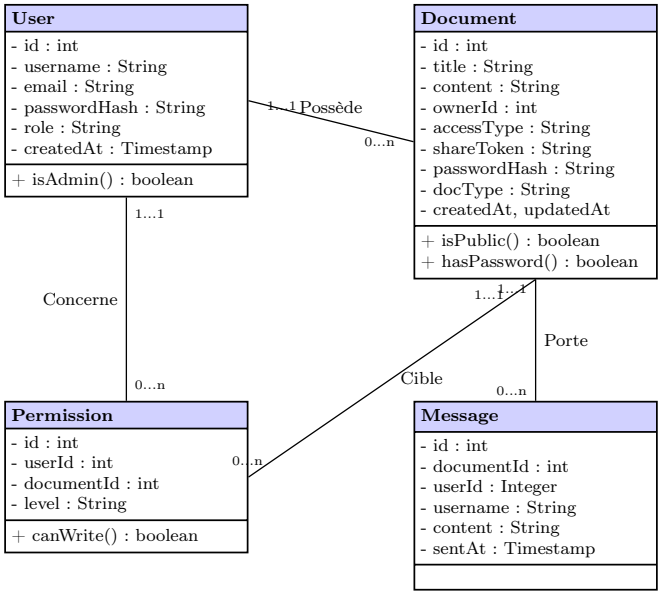

FIGURE 2 – Diagramme de classes de la couche Modèle (notation UML).

3.2 Diagramme de classes du modèle métier

La couche Modèle regroupe quatre classes Java qui représentent les entités manipulées par l'application. La figure 2 en présente le diagramme de classes. Ces classes sont de simples objets porteurs de données (POJO) : elles exposent des accesseurs et quelques méthodes utilitaires (`isAdmin()`, `hasPassword()`, `canWrite()`...), mais ne contiennent aucune logique d'accès à la base — celle-ci est entièrement déléguée aux DAO correspondants. Un document est rattaché à son propriétaire par le champ `ownerId` ; la classe `Permission` matérialise le partage *plusieurs-à-plusieurs* entre utilisateurs et documents en le qualifiant d'un niveau de droit ; chaque message de chat est rattaché à un document et, éventuellement, à son auteur.

3.3 Technologies et déploiement

L'application est packagée en archive `war` par Maven et se déploie sur un conteneur de servlets Apache Tomcat 10 (espace de noms Jakarta EE). Les dépendances déclarées dans le `pom.xml` sont l'API Servlet, l'API WebSocket, JSTL, le pilote JDBC MySQL et une bibliothèque de manipulation JSON. Le routage des servlets et le filtre d'authentification sont configurés dans `web.xml`. La couche cliente repose sur du JavaScript standard (ES5/ES6) sans framework, organisé en un script commun (`editor.js`, gestion du WebSocket, du chat et de la barre d'outils) et un script par type d'éditeur, communiquant entre eux par une petite interface de fonctions globales (`EDITOR_setContent`, `EDITOR_getContent`, `EDITOR_send`).

4 Base de données

4.1 Modèle relationnel

La persistance repose sur une base relationnelle MySQL/MariaDB (encodage `utf8mb4`) comportant sept tables. Le schéma relationnel est le suivant (les clés primaires sont soulignées, les clés étrangères en italique) :

- **users**(id, username, email, password_hash, role, created_at)
- **documents**(id, title, content, *owner_id*, access_type, share_token, has_password, password_hash, doc_type, created_at, updated_at)
- **permissions**(id, *user_id*, *document_id*, level)
- **messages**(id, *document_id*, *user_id*, username, content, sent_at)
- **document_history**(id, *document_id*, content, *saved_by*, saved_at)
- **document_access_tokens**(id, session_id, *document_id*, granted_at, expires_at)

— **restore_requests**(id, *document_id*, *requested_by*, *history_id*, status, requested_at, resolved_at)

4.2 Relations et intégrité

Les principales associations sont synthétisées dans le tableau 1, et la figure 3 schématise le modèle relationnel avec ses clés étrangères. L'intégrité référentielle est garantie par des clés étrangères. La suppression d'un utilisateur ou d'un document est propagée en cascade (`ON DELETE CASCADE`) sur les données dépendantes ; l'auteur d'un message ou d'une version est en revanche mis à `NULL` (`ON DELETE SET NULL`) afin de conserver l'élément même si le compte est supprimé. Une contrainte d'unicité sur le couple (`user_id`, `document_id`) de la table `permissions` empêche les doublons et permet une mise à jour idempotente du niveau de droit.

Entité A	Entité B	Relation
User	Document	Un utilisateur possède plusieurs documents (1–N).
User	Document	Partage via `permissions` : N–N qualifié par un niveau.
Document	Message	Un document porte plusieurs messages de chat (1–N).
Document	Historique	Un document a plusieurs versions archivées (1–N).
Document	Restauration	Une demande de restauration cible une version (N–1).
Document	Jeton d'accès	Jetons temporaires de déverrouillage (1–N).

Table 1 – Synthèse des associations entre entités.

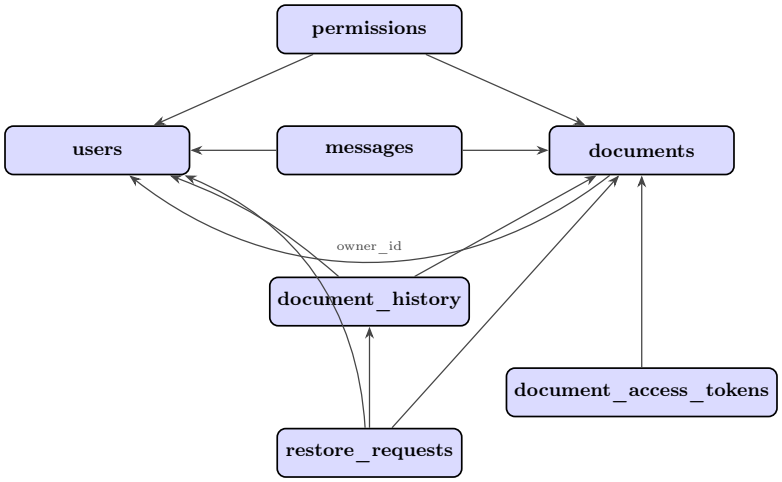

Figure 3 – Modèle relationnel : les sept tables et leurs clés étrangères (chaque flèche pointe de la clé étrangère vers la table référencée).

Le contenu des documents (`content`) est stocké sous forme de texte JSON dans une colonne `LONGTEXT`. Ce choix permet d'utiliser une structure unique pour les huit types de documents : chaque éditeur sérialise son état propre (grille de pixels, liste de diapositives, nœuds et arêtes, etc.) et le type est mémorisé dans la colonne `doc_type`. Comme l'illustre la figure 3, la table `documents` occupe une position centrale : elle est référencée par la plupart des autres tables, tandis que la table `users` est reliée à toutes les tables associées aux contributions des utilisateurs.

5 Fonctionnement collaboratif en temps réel

5.1 Transport par WebSocket

La collaboration en direct repose sur les WebSockets. Un *endpoint* unique, `DocumentWebSocket`, est exposé à l'adresse `/ws/doc/{docId}`. À l'ouverture d'un document, le client établit une connexion WebSocket dont l'identifiant du document figure dans l'URL. Côté serveur, les sessions connectées à un même document sont regroupées dans une « salle » (structure `ConcurrentHashMap` associant un

identifiant de document à l'ensemble de ses sessions). Le configurateur `HttpSessionConfigurator` transfère l'utilisateur de la session HTTP vers la session WebSocket, ce qui permet d'identifier l'auteur de chaque message.

5.2 Protocole de messages

Les échanges utilisent des messages JSON typés. Les principaux types sont :

- `init` — envoi du contenu courant et du nombre de connectés à l'arrivée d'un client ;
- `content_update` — diffusion d'une modification de contenu à tous les autres clients de la salle ;
- `pixel_update`, `slide_change`, `cursor_cell` — messages fins propres à certains éditeurs (un pixel modifié, changement de diapositive, position de curseur dans le tableur) ;
- `chat_message` — message de chat, enregistré en base puis diffusé à tous ;
- `save_request` — sauvegarde du document : mise à jour du contenu et archivage d'une version dans l'historique ;
- `title_update` — modification du titre ;
- `user_joined` / `user_left` — notifications de présence ;
- `force_restore`, `restore_request`, `restore_approved`, `restore_rejected` — messages liés au workflow de restauration.

Lorsqu'un utilisateur modifie un document, son éditeur envoie un message de mise à jour ; le serveur le rediffuse à tous les autres participants de la salle, qui appliquent le nouvel état. Les messages de chat et les confirmations de sauvegarde sont, eux, diffusés à l'ensemble des participants, expéditeur compris. Pour l'éditeur Pixel Art, une optimisation a été mise en place : seul le pixel modifié est transmis pendant l'édition, le contenu complet n'étant sérialisé qu'à la sauvegarde.

5.3 Robustesse côté client

Le client gère la perte de connexion : en cas de coupure, une reconnexion automatique est tentée avec un délai croissant (*backoff* exponentiel plafonné). L'envoi des modifications de contenu est temporisé (*debounce*) afin de limiter le nombre de messages lors d'une frappe rapide.

6 Gestion des utilisateurs, des droits et de la sécurité

6.1 Authentification et rôles

L'application distingue trois rôles : `admin`, `registered` (inscrit) et `visitor`. L'inscription valide le format des champs (longueur du nom, format de l'adresse e-mail, longueur et confirmation du mot de passe) et vérifie l'unicité du nom d'utilisateur et de l'e-mail. La connexion ouvre une session HTTP dans laquelle l'utilisateur est mémorisé. Le filtre `AuthFilter`, appliqué aux URL sensibles (`/home`, `/editor/*`, `/document/*`, `/admin/*`), redirige vers la page de connexion tout visiteur non authentifié et réserve les routes d'administration aux seuls administrateurs.

6.2 Droits sur les documents

Chaque document a un *propriétaire*, qui dispose de tous les droits. Les autres utilisateurs peuvent recevoir une permission de *lecture seule* ou de *lecture et écriture* via la page de partage. Trois modes d'accès sont combinés à ce système : `public` (visible par tous), `link` (accessible à toute personne connaissant le lien de partage) et `private` (réservé au propriétaire et aux utilisateurs explicitement autorisés). La servlet `EditorServlet` centralise le contrôle d'accès en vérifiant, avant tout affichage, que l'utilisateur a le droit de voir le document, et calcule s'il peut l'éditer.

6.3 Protection par mot de passe et jetons d'accès

Un document peut être protégé par un mot de passe. Plutôt que de faire transiter ce mot de passe dans l'URL, l'application utilise un mécanisme de *jetons d'accès* : lorsqu'un utilisateur saisit le bon mot de passe, la servlet `DocumentUnlockServlet` enregistre un jeton temporaire en base (table `document_access_tokens`, valable huit heures) associé à sa session. Les accès suivants sont autorisés tant que ce jeton est valide, et les jetons expirés sont purgés automatiquement.

6.4 Mots de passe

Les mots de passe ne sont jamais stockés en clair : ils sont hachés en SHA-256 (empreinte hexadécimale de 64 caractères) par la classe utilitaire `PasswordUtil` avant insertion en base. Toutes les requêtes SQL sont construites avec des `PreparedStatement`, ce qui protège l'application contre les injections SQL. Les paramètres de connexion à la base sont isolés dans la classe `DBConnection` et doivent être adaptés à l'environnement de déploiement.

7 Éléments extérieurs utilisés

Conformément à l'énoncé, les éléments externes employés sont énumérés ci-dessous, avec leur rôle et leur emplacement dans le projet. Les parties que nous n'avons pas écrites nous-mêmes y sont clairement identifiées.

Technologies vues en cours. Les éléments suivants relèvent des technologies imposées par l'UE et étudiées en cours et en TP ; leur emploi faisait partie du cadre attendu du projet :

- **Apache Tomcat 10** — conteneur de servlets et de WebSockets utilisé pour le déploiement.
- **JDBC et le pilote MySQL Connector/J** — connexion à la base de données ; le pilote est déclaré comme dépendance Maven.
- **JSTL** — bibliothèque de balises standard, employée dans toutes les pages JSP pour éviter les *scriptlets*.
- **Bibliothèque JSON** — analyse et construction des messages JSON échangés par le WebSocket, côté serveur.

Éléments externes complémentaires. Les deux éléments suivants n'ont pas été vus en cours et ont été ajoutés par le groupe :

- **Bulma (CSS, via CDN)** — framework CSS utilisé pour la mise en forme générale (barre de navigation, boutons, formulaires). Le design n'étant pas un critère d'évaluation, ce choix vise uniquement une présentation correcte.
- **Quill.js (via CDN)** — éditeur de texte riche réutilisé pour le *seul* type de document « Texte riche ». Nous signalons en toute transparence que cet éditeur n'a pas été développé par nos soins : il n'est utilisé que pour ce type particulier.

Travail personnel. À l'exception du type « Texte riche », les sept autres éditeurs (Code avec coloration syntaxique, Pixel Art, Tableur, Présentation, Planning, Carte mentale, Diagramme) ont été *intégralement développés par le groupe* en JavaScript, sans bibliothèque d'édition externe. L'ensemble de la couche serveur (servlets, DAO, WebSocket, filtre, modèle) est également un travail personnel.

8 Difficultés rencontrées et limites

Le projet répond aux attentes de l'énoncé, mais certains points méritent d'être signalés honnêtement.

- **Fusion des modifications concurrentes.** La synchronisation diffuse l'état du document (message `content_update`) : en cas de frappe réellement simultanée de deux utilisateurs sur un même document texte, c'est la dernière mise à jour reçue qui l'emporte. Nous n'avons pas implémenté d'algorithme de fusion de type *Operational Transformation* ou CRDT, qui aurait dépassé le cadre du projet. L'éditeur Pixel Art atténue ce problème par des mises à jour à la granularité du pixel.
- **Connexion à la base.** La classe `DBConnection` gère une connexion JDBC unique partagée (avec reconnexion automatique). Un véritable pool de connexions serait préférable pour un usage fortement concurrent.
- **Hachage des mots de passe.** Les mots de passe sont hachés en SHA-256 sans sel. Une fonction dédiée et salée (de type *bcrypt*) offrirait une meilleure résistance, mais sortait du périmètre des notions vues en TP.

— **Contrôle d'accès au niveau du WebSocket.** Le contrôle des droits de visualisation et d'écriture est principalement assuré par les servlets et l'interface ; le *endpoint* WebSocket pourrait revérifier systématiquement, côté serveur, les permissions de chaque message reçu.

Aucune fonctionnalité de base de l'énoncé n'a été laissée de côté ; les limites ci-dessus concernent des améliorations avancées et facultatives.

Difficultés concrètes rencontrées. Au-delà de ces limites de conception, le développement a soulevé plusieurs difficultés pratiques :

— **Mise en place de l'environnement.** Le passage à Apache Tomcat 10 a imposé l'espace de noms `jakarta.*` (et non `javax.*`) : plusieurs erreurs de déploiement provenaient d'imports ou de dépendances incompatibles. La configuration commune du projet Maven et de la base de données sur les postes de chaque membre a également demandé une phase de mise au point.

— **Débogage du WebSocket.** La transmission de la session HTTP vers la session WebSocket, nécessaire pour identifier l'auteur des messages, a été délicate à mettre en place et a nécessité l'écriture du configurateur `HttpSessionConfigurator`. La gestion des déconnexions et de la reconnexion automatique a aussi exigé plusieurs ajustements.

— **Synchronisation des éditeurs.** Faire fonctionner les huit éditeurs sur une même couche collaborative a demandé de définir une interface commune (`EDITOR_setContent`, `EDITOR_getContent`, `EDITOR_send`). Sans temporisation des envois, une frappe rapide saturait le canal ; la mise en place du *debounce* a résolu ce problème.

— **Gestion de la concurrence.** La diffusion simultanée de modifications à plusieurs clients a révélé des cas de retour d'écho et d'écrasement de contenu, qui ont conduit à distinguer les messages rediffusés « aux autres » de ceux rediffusés « à tous ».

Ces difficultés ont été résolues progressivement et nous ont permis de mieux comprendre le fonctionnement réel d'une application web temps réel.

9 Répartition du travail

Le projet a été mené par un groupe de trois étudiants. Le tableau 2 récapitule la répartition indicative des tâches.

Membre	Principales contributions	Temps
Rachid Ghodbane	Conception de la base de données, couche d'accès aux données (DAO), classes du modèle, authentification et gestion des droits (`AuthFilter`, `PasswordUtil`).	≈ 55 h
Anas Ben Abou	Servlets et contrôleurs, *endpoint* WebSocket et protocole temps réel, chat, historique et workflow de restauration.	≈ 55 h
Yacine Addadi	Pages JSP et gabarits, éditeurs JavaScript spécialisés, intégration de l'interface et de la couche collaborative côté client.	≈ 55 h
Total		$\approx$ **165 h**

Table 2 – Répartition du travail au sein du groupe.

La répartition s'est appuyée sur les trois couches du patron MVC, de manière à confier à chaque membre un ensemble cohérent : les données pour Rachid, le traitement et le temps réel pour Anas, l'interface pour Yacine. Le travail a toutefois été mené en étroite collaboration — l'interface commune entre couche serveur et éditeurs, ainsi que le protocole WebSocket, ayant été définis ensemble — et les phases de tests, de débogage et de rédaction du rapport ont été partagées de façon équitable entre les trois membres.

Conclusion

CollabDocs constitue une plateforme fonctionnelle d'édition collaborative de documents en temps réel, conforme aux exigences de l'UE Développement Web 2. L'ensemble des fonctionnalités

minimales demandées a été implémenté : visualisation et modification simultanées, propagation immédiate des changements, gestion des droits et des modes d'accès, chargement et sauvegarde, et communication entre utilisateurs. Le projet va au-delà du socle en proposant huit types d'éditeurs, un historique de versions et un mécanisme de restauration soumis à approbation.

Sur le plan technique, l'application met en œuvre de manière cohérente les technologies imposées — Java, servlets, JDBC, JSP/EL/JSTL, WebSockets et JavaScript — dans une architecture MVC clairement structurée. Les limites identifiées, principalement l'absence de fusion fine des modifications concurrentes, ouvrent des pistes d'amélioration. Ce projet nous a permis de consolider notre maîtrise du développement web côté serveur et de la communication temps réel, à travers une application complète menée en équipe, de la base de données jusqu'à l'interface.

Annexes

Cette annexe regroupe les schémas de conception détaillés du projet : le modèle relationnel de la base de données et les diagrammes de classes UML des couches Modèle, DAO et Contrôleur. Ils complètent les versions synthétiques présentées dans le corps du rapport.

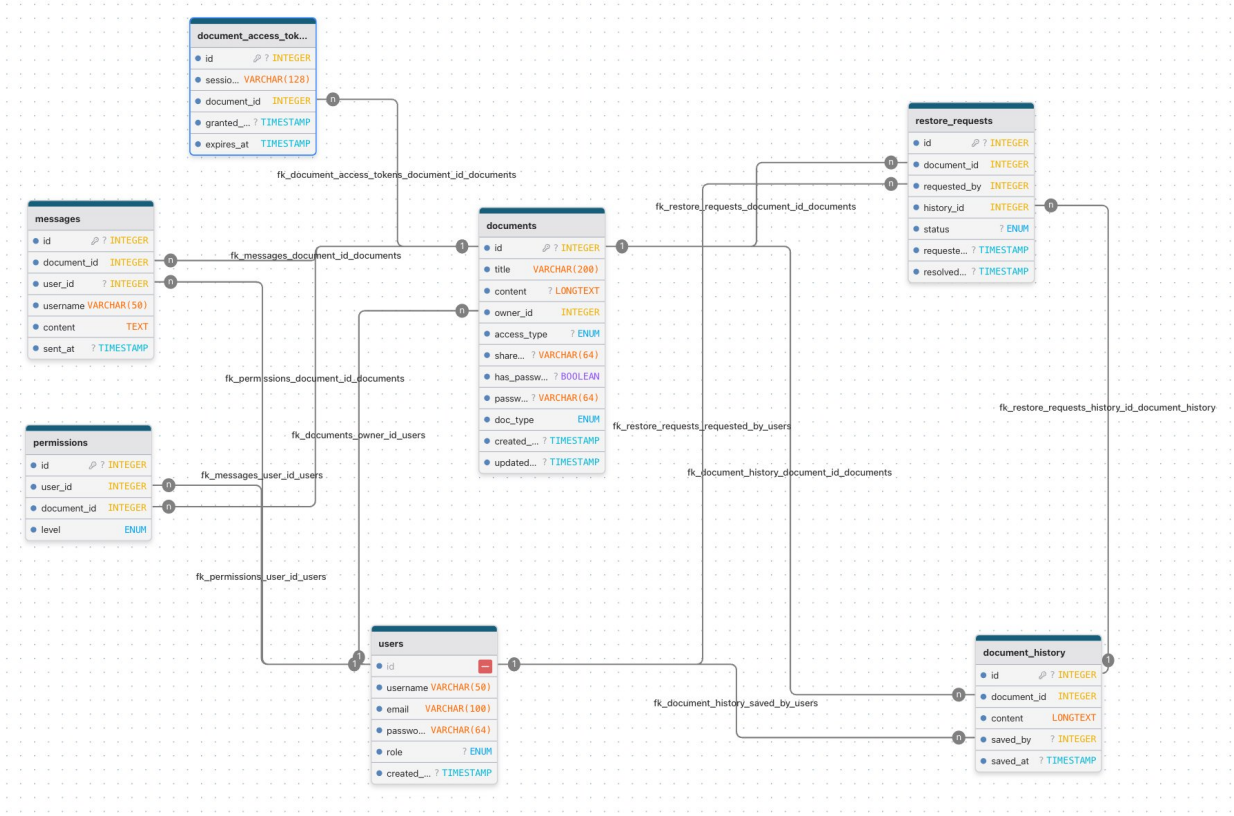

FIGURE 4 – Modèle relationnel complet de la base de données CollabDocs (sept tables et leurs clés étrangères).

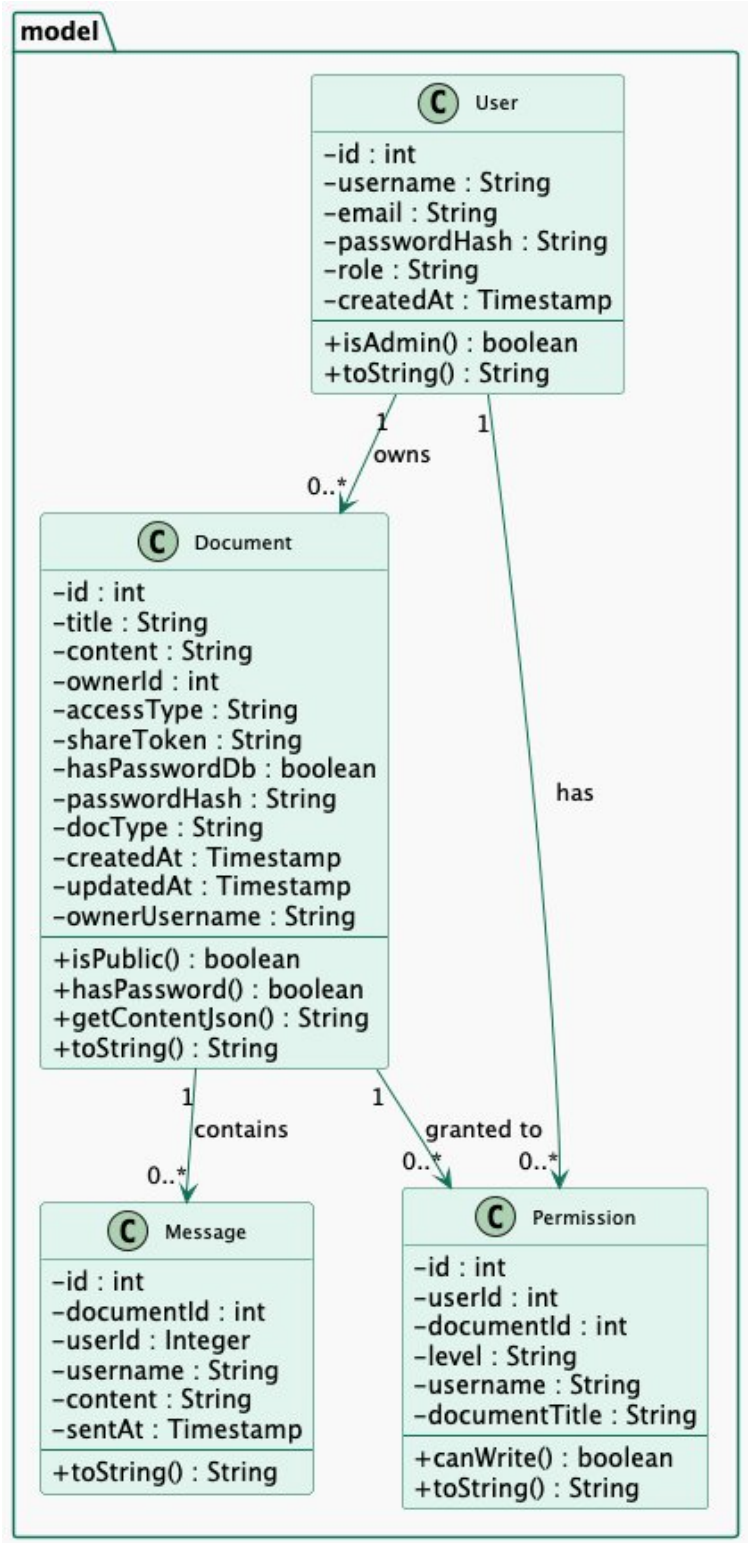

FIGURE 5 – Diagramme de classes UML de la couche Modèle (paquetage `model`).

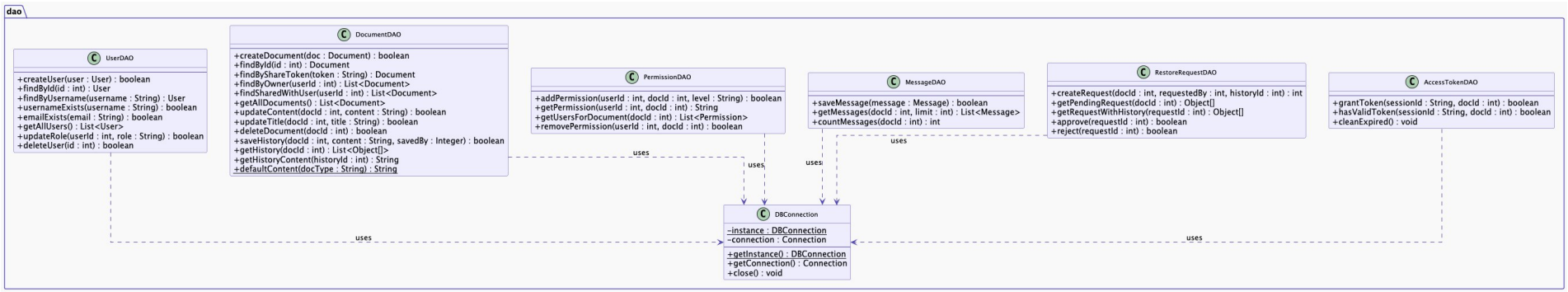

FIGURE 6 – Diagramme de classes UML de la couche d'accès aux données (paquetage `dao`). *Figure à consulter en zoomant dans le document.*

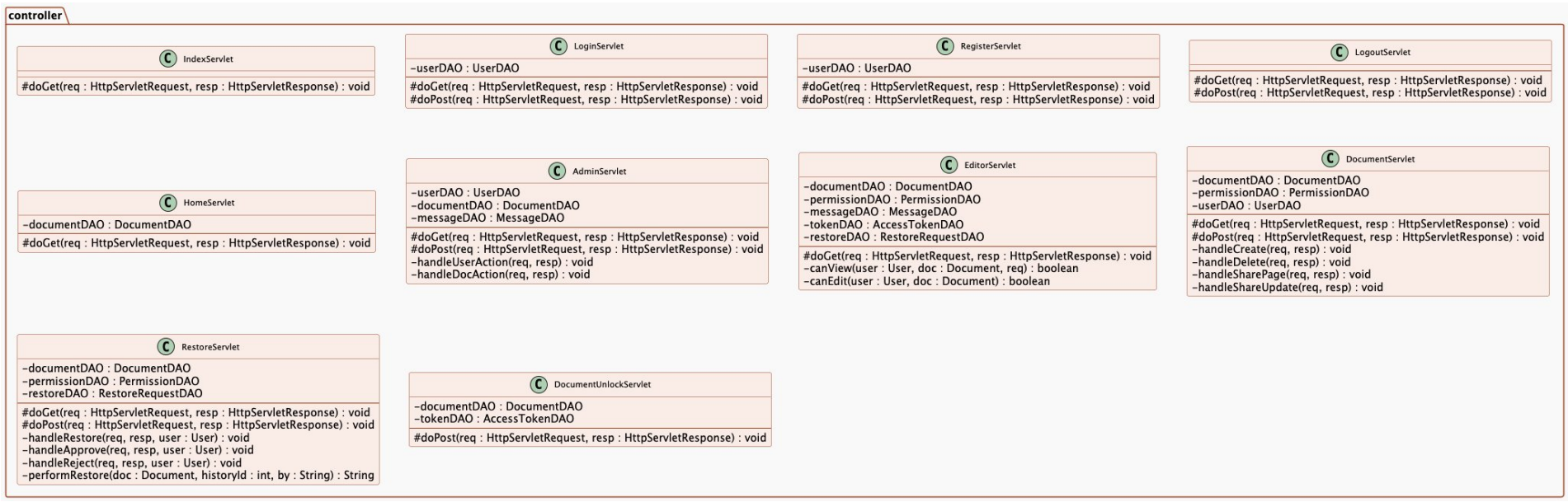

FIGURE 7 – Diagramme de classes UML de la couche Contrôleur (paquetage `controller`).

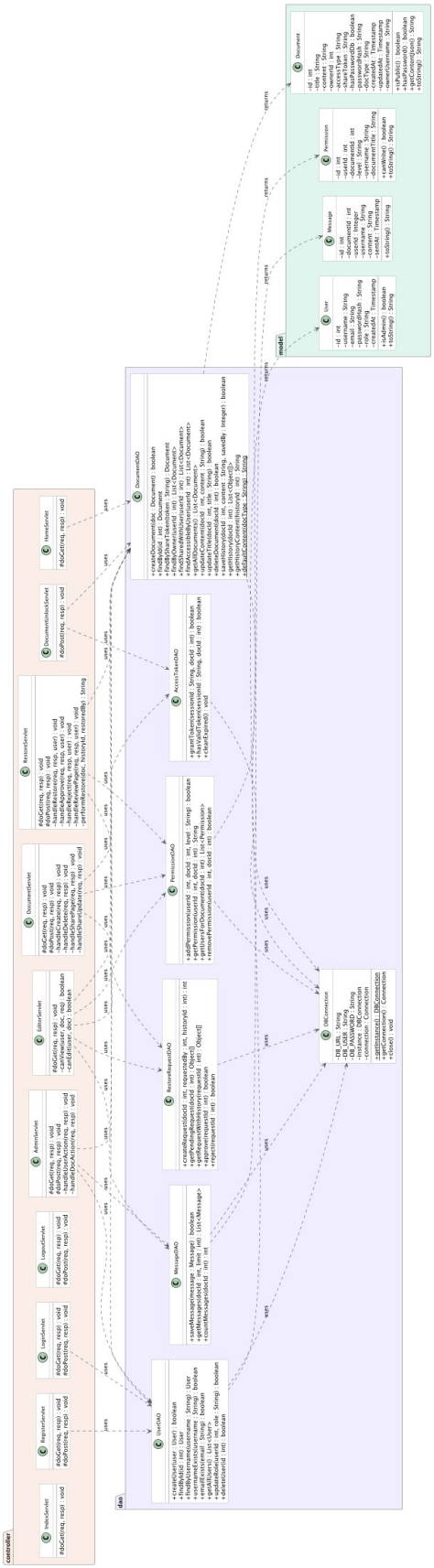

FIGURE 8 – Vue d'ensemble : interactions entre les couches Contrôleur, DAO et Modèle de l'application.