Loïc BOUTET

Félix-Antoine DIAMOND

Xavier HAMEL

Ramy A. YAHIA

Modèles et langages de bases de données pour l'ingénierie

GLO-2005

Projet de session

Travail présenté à

Monsieur Richard KHOURY

Faculté de sciences et de génie

Département d'informatique et de génie logiciel

Université Laval

16 avril 2024

Table des matières

1. Énonciation du problème et des exigences

1.1 Contexte

À la suite des délibérations de l'équipe, il a été décidé que l'application créée dans le cadre du projet de session du cours GLO-2005 de l'Université Laval est une application de prise de notes. Elle peut être utilisée pour la prise de notes ou la rédaction de tout autre texte simple. Les notes en soit sont des chaînes de caractères écrites et affichées en respectant le standard MarkDown (.md) afin d'offrir la possibilité aux usagers de réaliser un minimum de formatage. Le nom de l'application produite est Notepad+++, un clin d'œil au légendaire logiciel de traitement de code Notepad++.

1.2 Utilisateurs cibles

Les utilisateurs cibles sont potentiellement toutes les personnes munies d'un ordinateur et d'un clavier, mais les utilisateurs clés («power users») qui utiliseront toutes les fonctionnalités offertes seront les gens possédant de bonnes connaissances en technologie, spécifiquement le langage MarkDown.

1.3 Avantages de notre solution

L'application développée offre plusieurs avantages par rapport au bon vieux bloc note de Windows. D'abord, les notes peuvent être regroupées en carnets de notes afin de respecter une hiérarchie conviviale. De plus, il est possible de partager une note avec d'autres utilisateurs, qui pourront eux aussi modifier cette note. Finalement, un historique des versions précédentes des notes est conservé pour consultation, au besoin.

1.4 Exigences

L'équipe a convenu des exigences suivantes :

- Un utilisateur peut créer un compte à l'aide de son adresse courriel, d'un mot de passe et de son nom;
- Un utilisateur peut se connecter à son compte à l'aide de son adresse courriel et de son mot de passe;
- Un utilisateur peut créer un nouveau carnet de note et le nommer. Il peut renommer ce carnet de note à tout moment. Il peut supprimer ce carnet de note, ce qui aura pour effet de supprimer toutes les notes contenues dans ce carnet;
- Un utilisateur peut créer une nouvelle note et la nommer (ce nom sera immuable par la suite). Il peut supprimer cette note : s'il est le seul utilisateur à y avoir accès (cas 1), elle est supprimée définitivement ainsi que toutes les versions archivées de cette note; si d'autres utilisateurs ont encore accès à cette note (cas 2), il en perd simplement l'accès;
- Un utilisateur (1) peut partager une note avec n'importe quel utilisateur (2) (excepté lui-même). L'utilisateur 2 obtient l'accès à cette note ainsi qu'à toutes les versions archivées.

Il peut désormais la modifier. Une modification effectuée par n'importe quel utilisateur ayant accès à cette note crée une nouvelle version de cette note qui est disponible à tous les utilisateurs y ayant accès;

- Un utilisateur peut consulter l'historique des versions d'une note auquel il a accès.

1.5 Bases architecturales

Dans cette application, le niveau client sert principalement à afficher une interface graphique conviviale pour l'utilisateur et récupérer ses actions et saisies de données (« inputs »). Du traitement très simple, comme la vérification du format des adresses courriels et mots de passe, est aussi effectué. Le framework Vue est utilisé. Cinq pages sont nécessaires : la page de connexion et création de compte, le profil utilisateur, la liste des notes, la page d'édition de notes et l'historique d'une note.

Le serveur d'application récupère les informations envoyées par le client, effectue des vérifications et envoie des requêtes à la base de données. Le serveur est réalisé avec la libraire Python Flask et consiste en un API JSON respectant la convention REST.

La base de données MySQL stocke toutes les informations nécessaires sur les utilisateurs et les notes.

2. Modélisation des données

2.1 Modèle entité-relation

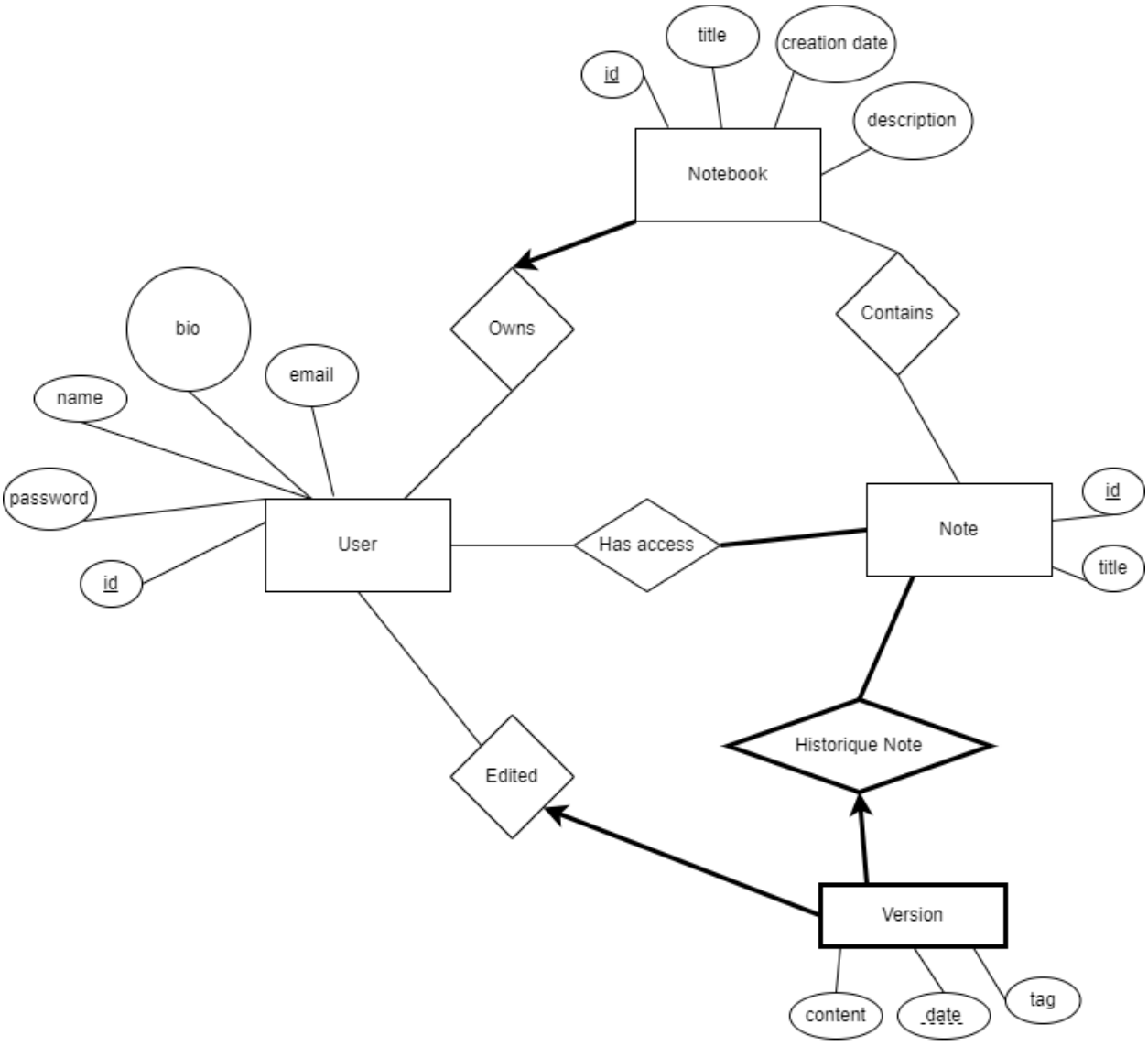

Figure 1. Modèle entité-relation de l'application

Le modèle entité-relation de la base de données contient quatre ensembles d'entités et cinq ensembles de relations.

Des utilisateurs (**User**) possèdent chacun un nom, un mot de passe (la version stockée est hachée), une bio (un court texte descriptif), un courriel et sont identifiés par un id, qui sert de clé primaire.

Des carnets de notes (**Notebook**) sont composés d'un titre, d'une date de création, d'une description et sont identifiés par un id, qui sert de clé primaire.

Un utilisateur possède (**Owns**) un nombre arbitraire de carnets de notes. Il peut en créer et en supprimer à sa guise. Cependant, un carnet de notes doit appartenir à un seul utilisateur.

Une note (**Note**) est identifiée par un id, sa clé primaire; ainsi que d'un titre. Une note ne possède pas de contenu en soit, celui-ci est enregistré en tant que version.

Un carnet de notes peut contenir (**Contains**) un nombre indéterminé de notes ou être vide. Une note peut flotter librement dans la collection d'un utilisateur, elle n'est pas nécessairement dans un carnet de notes. Une note peut être dans plus d'un carnet de notes.

Un utilisateur a accès (**Has access**) à un nombre indéterminé de notes. Une note doit obligatoirement être accessible à un ou plusieurs utilisateurs, sans quoi elle est supprimée car inutilisable. Il est à noter qu'une note peut encore exister malgré sa suppression par son créateur si elle avait préalablement été partagée à un autre utilisateur.

La version d'une note (**Version**) représente l'état du contenu textuel d'une note enregistré à un moment précis. Par défaut, le contenu d'une note est celui de sa plus récente version, mais il est possible en tout temps de récupérer d'anciennes versions. Une version est composée d'une date (date et heure précise, clé discriminante), d'un contenu textuel et d'un tag. Puisqu'une version ne peut exister sans appartenir à une note (**Historique Note**), c'est un ensemble d'entité faible. Sa clé primaire est une concaténation de l'id de la note et de la date de la version. Il est à noter qu'une note possède au minimum une version, soit son état à sa création.

Une nouvelle version d'une note a été rédigée (**Edited**) par un utilisateur, qui peut être le créateur de la note ou tout autre utilisateur disposant de l'accès à la suite du partage de la note.

2.2 Modèle relationnel

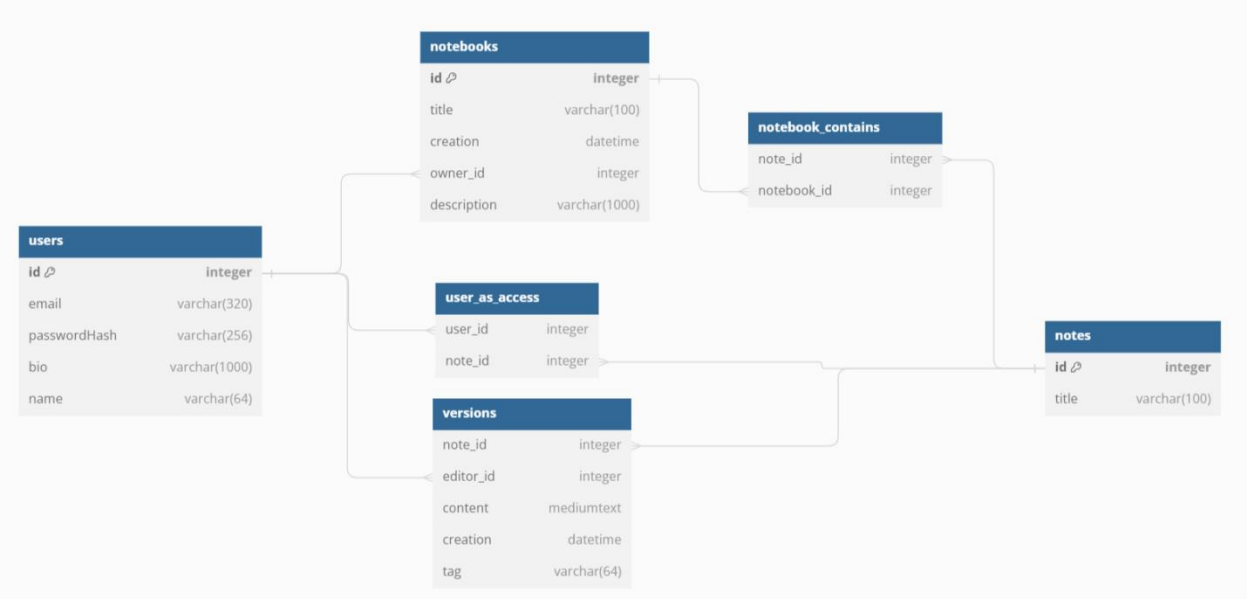

Figure 2. Modèle relationnel de l'application

Le modèle relationnel est composé de six relations.

La relation **users** contient les informations sur les utilisateurs de l'application. Ils sont identifiés par un id unique (clé primaire). Cet id est un entier auto incrémental défini par MySQL.

La relation **notebooks** contient les informations sur les carnets de notes, identifiés par un id unique (clé primaire). Un carnet de notes peut être relié à son propriétaire grâce au champ owner_id.

La relation **notes** contient les informations sur les notes, identifiées par un id unique (clé primaire).

La relation **versions** contient les informations sur toutes les versions historiques des notes existantes. Le champ note_id permet de relier une note à ses versions. Le champ editor_id associe la version à l'utilisateur qui l'a éditée. Le champ content est probablement le plus important de la base de données : il s'agit du contenu d'une version précise d'une note. Ce champ utilise le type de données MySQL mediumtext, permettant de stocker des chaînes de caractères d'une taille allant jusqu'à 16 MB.

La relation **notebook_contains** permet de savoir quelles notes sont à l'intérieur d'un carnet de notes.

La relation **user_has_access** permet de savoir quelles notes sont accessibles à un utilisateur précis.

3. Création de la base de données

La base de données peut être créée à l'aide du script setup_bd.py. Celui-ci récupère d'abord les variables d'environnements nécessaires à la connexion (hôte, port, nom d'utilisateur, mot de passe et nom de la base de données), puis exécute cinq fichiers SQL : DestroyDB.sql, InitializeDB.sql, addTriggers.sql, createNote.sql et populate.sql, dans cet ordre.

DestroyDB.sql supprime tout le contenu de la base de données si existante afin de la réinitialiser pour commencer à l'utiliser ou faire des tests. Toutes les tables, procédures et triggers sont supprimés.

InitializeDB.sql vient à son tour créer toutes les relations et index nécessaires. Toutes les contraintes mentionnées dans la section *2 : Modélisation des données* y sont définies. Les clés étrangères et leurs contraintes de référence y sont aussi définies :

- owner_id dans notebooks fait référence à id dans users. Lorsqu'un utilisateur est supprimé, on supprime aussi tous ses carnets de notes.
- note_id dans versions fait référence à id dans notes. Lorsqu'une note est supprimée, on supprime aussi tout l'historique de ses versions.
- editor_id dans versions fait référence à id dans users. Lorsqu'un utilisateur est supprimé, on ne veut pas effacer son historique de contribution afin que d'autres

utilisateurs puissent encore voir l'historique complet d'une note partagée. On attribue donc la valeur null à editor_id dans ce cas.

- user_id dans user_has_access fait référence à id dans users. Lorsqu'un utilisateur est supprimé, on supprime aussi ses accès aux notes.
- note_id dans user_has_access fait référence à id dans notes. Lorsqu'une note est supprimée, plus personne ne devrait y avoir accès donc les accès sont aussi supprimés pour cette note.
- note_id dans notebook_contains fait référence à id dans notes. Lorsqu'une note est supprimée, elle ne devrait plus être présente dans aucun carnet de note.
- notebook_id dans notebook_contains fait référence à id dans notebooks. Lorsqu'un carnet de notes est supprimé, celui-ci ne devrait plus contenir de notes.

populate.sql vient ensuite peupler la base de données avec des commandes INSERT INTO.

addTriggers.sql et createNote.sql servent respectivement à ajouter les gâchettes à la base de données et à créer la procédure d'ajout de notes dans la base de données. Ces fichiers sont décrits plus en détail à la section *4 : Création des requêtes et des routines*.

4. Création des requêtes et des routines

Toutes les requêtes de base effectuées sur la base de données peuvent être trouvées dans le fichier backend/database_operations.py. Les gâchettes et procédures quant à elles sont définies dans les fichiers addTriggers.sql et createNote.sql.

La procédure **create_note** permet de créer une nouvelle note. Lorsqu'une note est créée par un utilisateur, cet utilisateur doit par défaut y avoir accès, sans quoi la note serait inutilisable. Une version initiale doit aussi être créée pour cette note. Cette procédure permet donc de réaliser les trois insertions nécessaires à la création d'une nouvelle note.

La gâchette **delete_orphan_notes_when_access_deleted** permet de s'assurer qu'aucune note ne se retrouve perdue et inutilisable dans la base de données. En effet, une note peut toujours exister malgré sa suppression par son créateur si elle avait préalablement été partagée à d'autres utilisateurs. Cette gâchette vérifie, lorsqu'un accès à une note est supprimé, qu'au moins un utilisateur a encore accès à cette note, sans quoi la note est supprimée de la base de données.

La gâchette **delete_orphan_notes_when_parent_deleted** remplit la même fonction, mais lorsqu'un utilisateur est supprimé. Elle vérifie, à l'aide d'une triple requête imbriquée, quelles sont les notes où le seul accesseur est l'utilisateur supprimé.

5. Indexation et optimisation du système

Seulement deux index sont nécessaires pour optimiser les performances de la base de données puisque beaucoup de requêtes sont faites en utilisant les index déjà créés par défaut par MySQL (en arbre B groupé pour les clés primaires de relations).

L'index versions_index sur la relation versions a comme clé de recherche <note_id, creation, editor_id>. Il permet donc les requêtes index seulement pour récupérer les tuples (creation, editor_id) associés à une note précise. Cela est utile puisque l'utilisateur souhaitera souvent obtenir la liste des versions précédentes, sans pour autant obtenir leur contenu. Cet index est d'autant plus utile que la relation versions contient l'attribut content, qui peut contenir des chaînes de caractères d'une taille allant jusqu'à 16MB. Pouvoir obtenir la liste des versions sans être encombré par leur contenu lors de la recherche améliore ainsi les performances de manière significative.

L'index notebooks_index sur la relation notebooks a comme clé de recherche <owner_id>. Presque toutes les requêtes effectuées sur la relation notebooks sont des requêtes d'égalité sur owner_id, il est donc utile d'avoir un index en arbre B non-groupé sur cet attribut. Un index groupé aurait été plus pertinent, mais MySQL ne le permet pas.

6. Normalisation des relations

En analysant notre domaine d'application et en y établissant les dépendances fonctionnelles qui reflètent les contraintes du monde réel, nous avons réussi à normaliser toutes les relations de notre base de données à une forme normale de Boyce Codd. Par conséquent, elles respectent également les formes normales inférieures. Cela a permis d'éliminer toute redondance dans les données stockées, réduisant ainsi les risques d'anomalies lors de l'insertion, de la mise à jour ou de la suppression de données.

Les relations user_has_access et notebook_contains sont deux exemples de normalisation des relations: elles servent à faire le lien entre deux ensembles d'entités en évitant les répétitions.

7. Sécurité de la BD

La sécurité de l'application a été une priorité majeure lors de sa conception et son implémentation. Plusieurs mesures ont été prises afin d'assurer un projet fiable et résilient. Premièrement, les mots de passe des utilisateurs sont toujours hachés à l'aide de l'algorithme SHA-256 de la librairie Python Werkzeug avant insertion dans la base de données. Cela garantit que même en cas de fuite de données, les mots de passes resteront protégés et indéchiffrables. Nous sommes également conscients des risques posés par les injections SQL, alors nous avons employé l'utilisation de requêtes paramétriques avec PyMySql pour toutes les interactions avec la base de données. Cette

approche assure que les données entrantes sont traitées comme de simples données, et non comme du code SQL exécutable.

Il est important de noter que notre application fonctionne actuellement en HTTP, mais il serait nécessaire d'utiliser le standard HTTPS pour sécuriser les communications entre le client et le serveur advenant un déploiement en production.

8. Implémentation de la logique d'affaire

L'application Notepad+++ a été construite sur une base architecturale REST. Cette décision a été prise afin d'assurer des interactions claires et structurées entre le client et le serveur. Chaque fonctionnalité de l'application est encapsulée dans des fonctions Flask distinctes. Toutes ont des noms descriptifs et des messages de retour explicites, facilitant ainsi la lisibilité et la compréhension de notre API. Les fonctions sont responsables de valider les données d'entrée dans les cas où c'est nécessaire. Par exemple, la fonction register qui s'occupe d'enregistrer de nouveaux utilisateurs valide non seulement le format du courriel entré, mais valide aussi que le courriel et le nom d'utilisateur ne sont pas déjà utilisés par un autre compte. Pour assurer la robustesse de l'application, des blocs try-except ont été intégrés dans toutes les méthodes Flask, évitant ainsi les comportements inattendus. Ces pratiques assurent que notre logique d'affaires reste en tout temps efficace et sécurisée.

9. Implémentation de l'interface utilisateur

L'interface utilisateur est bâtie avec le plus de retours visuels possibles lorsqu'un utilisateur interagit avec l'application. L'objectif est d'indiquer clairement à l'utilisateur si le résultat de ses actions a été une réussite ou un échec.

L'application adopte un style de navigation assez linéaire. L'utilisateur commence par se connecter à son compte avant de pouvoir accéder au menu principal. À partir de cet endroit, il peut soit accéder aux notes récentes ou suivre la route standard et accéder à ses carnets de notes. Par la suite, il pourra choisir un notebook et finalement il pourra choisir une note à modifier. Un menu dans le haut de l'écran facilite la navigation et permet de retourner à la page précédemment visitée.

Le système de note utilise la librairie tipTap, une librairie très utilisée pour les applications de prise de notes. Elle gère l'affichage formatté du texte MarkDown. Son style minimaliste et intuitif rend l'application particulièrement facile d'accès et rapide à comprendre.

10. Tests du système

Plusieurs tests manuels de l'application ont été faits pour assurer l'intégration correcte entre l'interface utilisateur, la logique d'affaire et la base de données. Cette approche garantit que les modifications réalisées par l'utilisateur au niveau de l'interface sont correctement traitées par la couche intermédiaire avant d'être enregistrées dans la base de données. Ces changements sont ensuite retransmis à la couche intermédiaire et affichés à la couche utilisateur. La gestion des erreurs est également traitée par le système. Des erreurs sous la forme de pop-ups et de messages d'erreur ont été implémentés afin de guider l'utilisateur vers une utilisation appropriée de Notepad+++. Par exemple, saisir un courriel avec un format invalide dans la page de login ou d'inscription affiche une erreur avec un message clair, invitant l'utilisateur à saisir un courriel de format valide. Ensuite, si le courriel ou le mot de passe est invalide, un pop-up est présent pour informer l'utilisateur du problème et l'inviter à réessayer.

11. Accessibilité du système

Grâce à la gestion des sessions par l'application, un utilisateur peut se connecter sur différents appareils.

L'application peut être utilisée malgré une connexion internet lente ou instable. Les communications entre le client et le serveur s'en tiennent au minimum : une fois l'application chargée, seules les données nécessaires sont échangées en format JSON.

La création d'un compte ne nécessite pas d'authentification externe, seule une adresse courriel valide est nécessaire.

L'application fournit des rétroactions à l'utilisateur lorsqu'il interagit avec l'application. Il reste continuellement informé du résultat de ses actions. L'utilisateur peut donc facilement voir si ses actions ont été une réussite ou un échec et adapter ses actions suivantes en conséquence.

Les couleurs de l'application s'en tiennent à une palette monochrome forte en contraste, permettant aux utilisateurs ayant des troubles visuels d'utiliser l'application sans problème.

12. Gestion de l'équipe et organisation du travail

La communication et la coordination ont été efficaces tout au long du projet. Une longue rencontre préliminaire a permis de s'entendre sur le fonctionnement de l'équipe et d'assurer la division des tâches. Quelques échanges ont eu lieu au cours du projet, mais seulement une autre rencontre officielle a été nécessaire pour mettre en commun les différentes parties du projet.

Un répertoire GitHub a été configuré dès le début du projet afin d'éviter les conflits lors de la modification du code par plusieurs membres de l'équipe en parallèle.

La séparation du travail n'était peut-être pas optimale d'un point de vue académique : une personne s'est chargée presque entièrement de l'interface utilisateur, une autre, du serveur et deux autres du rapport et de la base de données de manière conjointe. Or, cela n'a pas été un problème puisque chaque membre de l'équipe a assuré un suivi constant sur les autres parties même s'il n'en était pas l'auteur. Avant de fusionner des changements dans la branche master du répertoire GitHub, deux personnes devaient approuver ces changements. Chaque membre de l'équipe a ainsi contribué à l'ensemble du projet. De plus, l'architecture s'est décidée en équipe, il n'y a donc pas eu de conflits à ce niveau.