Faculté des sciences et de génie

UNIVERSITÉ LAVAL

Modèles et langages des bases de données pour l'ingénierie
GLO-2005
Hiver 2024
Baccalauréat en GLO

Rapport projet de session

Cédric Fontaine - 536983535

Melek Sebri - 536938082

Patrick Meunier - 536997115

Thierry Cantin-Demers - 536966799

Destinataire

Monsieur Richard Khoury

Date de remise : 16 avril 2024

Table des matières

Table des figures

1 Énonciation du problème et des exigences

Face aux défis rencontrés par de nombreux étudiants de l'Université Laval dans la gestion des projets d'équipe et la collaboration, en raison de l'absence d'une plateforme centralisée sur le portail universitaire, notre projet est né de cette lacune. En constatant que les étudiants doivent souvent recourir à des applications tierces pour la collaboration, ce qui peut être complexe et peu pratique, nous avons envisagé de créer une solution qui répondrait spécifiquement aux besoins de collaboration des étudiants et des responsables de cours. Notre plateforme collaborative universitaire est une réponse à la nécessité d'avoir une solution centralisée et adaptée aux besoins spécifiques de collaboration des étudiants de l'Université Laval, facilitant ainsi la gestion des projets d'équipe et améliorant l'efficacité de la collaboration au sein de l'université.

Pour répondre aux besoins spécifiques des étudiants dans notre application collaborative universitaire, nous devons tenir compte de plusieurs exigences essentielles :

Les étudiants ont besoin de créer et de gérer des équipes pour leurs projets académiques, de communiquer efficacement avec leurs coéquipiers, de consulter et de gérer les calendriers d'événements liés aux projets et aux cours, ainsi que d'accéder à des fonctionnalités de suivi des tâches et des progrès du projet.

Pour répondre à ces exigences, notre système doit offrir des fonctionnalités telles que la création et la gestion des équipes avec des outils de recherche et d'invitation des membres, des outils de communication intégrés comme la messagerie instantanée et les discussions de groupe, des calendriers d'événements partagés pour la planification, le suivi des tâches et des progrès du projet avec des listes de tâches et des tableaux de bord, ainsi qu'une authentification sécurisée et une gestion des comptes avec des niveaux d'autorisation adaptés.

Au niveau client, l'interface utilisateur est cruciale pour offrir une expérience fluide aux utilisateurs. Le client collecte et transmet les interactions utilisateur au serveur d'application pour traitement, et affiche les données de manière claire.

Le serveur d'application est le cœur de notre système, gérant la logique métier, la création des équipes, la communication entre les membres, etc. Il assure également l'authentification et l'accès aux fonctionnalités.

La base de données stocke et gère les données de l'application, garantissant leur intégrité et répondant aux requêtes du serveur pour une expérience utilisateur fiable.

2 Modélisation des données

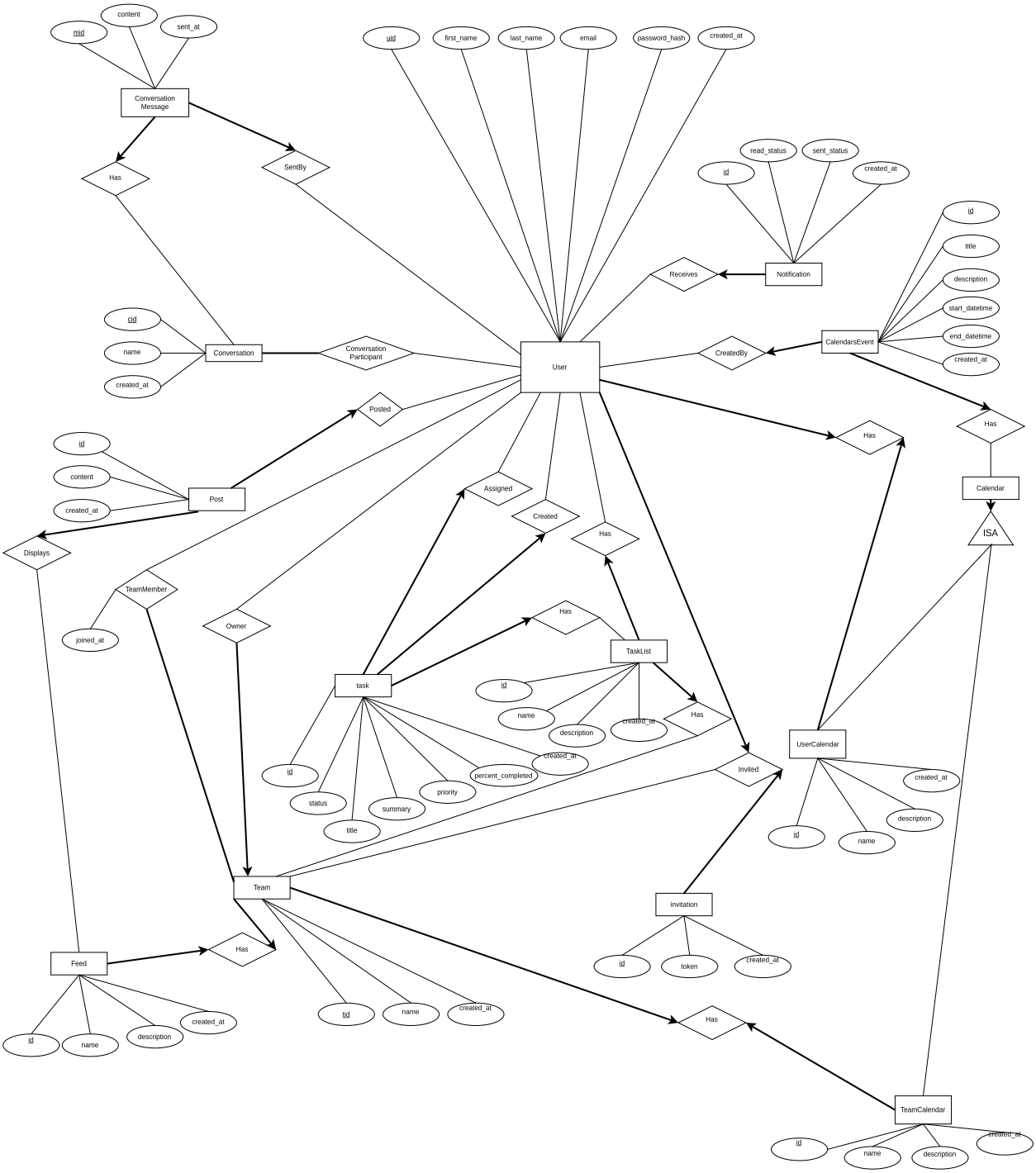

FIGURE 1 – Diagramme entité relation

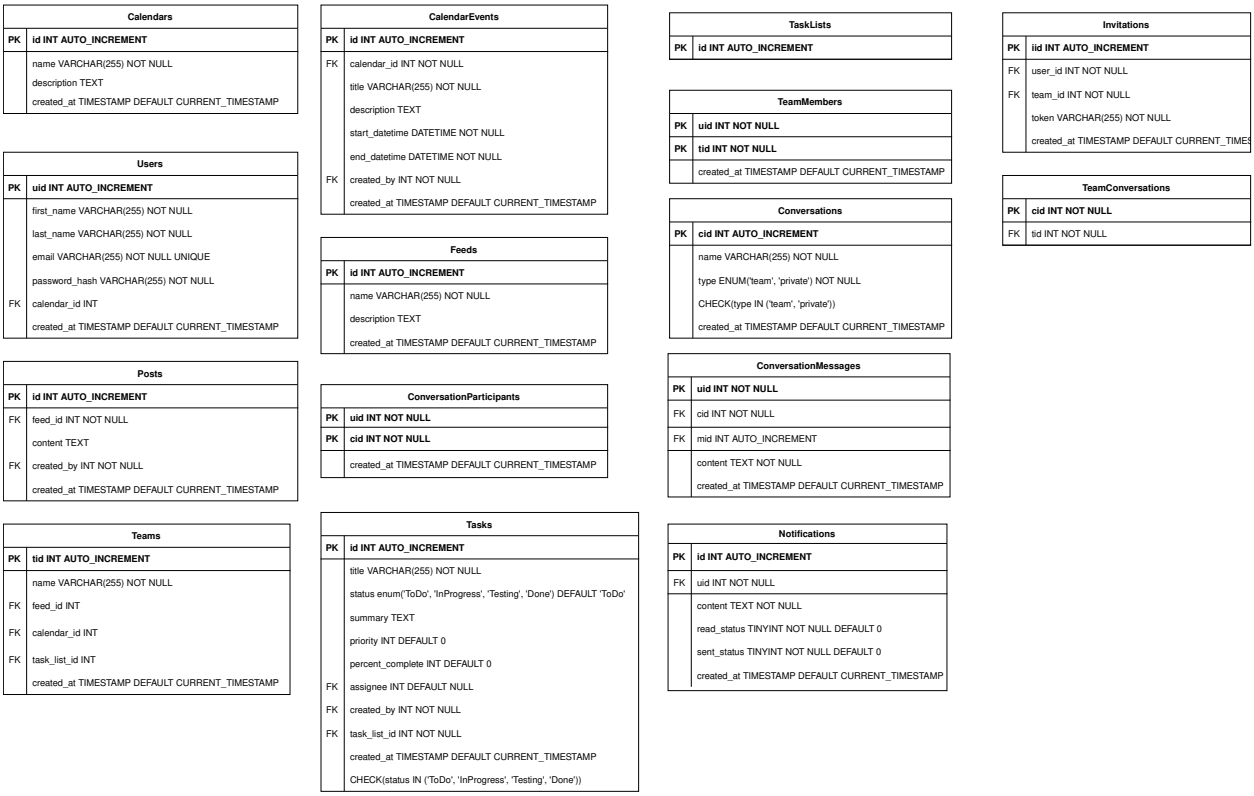

FIGURE 2 – Modèle relationnel

3 Création de la base de données

Notre base de données est constituée de 15 relations, et il est important de noter que chaque relation est conçue pour fonctionner de manière flexible, que ce soit avec 50 tuples ou 500. De plus, la plupart des relations incluent l'attribut created_at, qui enregistre automatiquement la date et l'heure de création de chaque tuple. Cet attribut est utile pour plusieurs fonctionnalités au-delà du projet, notamment les historiques des actions et les analyses temporelles.

Calendars :

Les attributs de cette relation Calendars comprennent un identifiant unique (`id`), un nom (`name`), une description (`description`), et une date de création (`created_at`). Ces attributs sont essentiels pour définir et organiser les calendriers dans la base de données. En ce qui concerne les contraintes d'intégrité, la clé primaire `id` doit être unique pour chaque calendrier. De plus, aucun champ `name` ou `created_at` ne peut avoir une valeur NULL, assurant que chaque calendrier a un nom défini et une date de création connue.

Users :

Les attributs de la relation Users incluent un identifiant unique (`uid`), un prénom (`first_name`),

un nom de famille (last_name), une adresse courriel unique (email), un hash de mot de passe (password_hash), une clé étrangère pour le calendrier (calendar_id), et une date de création (created_at). Ces attributs sont essentiels pour gérer les utilisateurs dans la base de données. En ce qui concerne les contraintes d'intégrité, la clé primaire uid doit être unique pour chaque utilisateur, (email) doit également être unique pour chaque utilisateur. De plus, les champs first_name, last_name, email, password_hash, et created_at ne peuvent pas avoir de valeur NULL. La clé étrangère calendar_id référence un calendrier avec des actions CASCADE sur DELETE et UPDATE.

CalenderEvents :

Les attributs de la relation CalendarEvents comprennent un identifiant unique (id), une clé étrangère pour le calendrier (calendar_id), un titre (title), une description (description), des horaires de début (start_datetime) et de fin (end_datetime), un créateur (created_by), et une date de création (created_at). Ces attributs sont nécessaires pour gérer les événements liés aux calendriers. En termes de contraintes d'intégrité, la clé primaire id doit être unique pour chaque événement. Les champs calendar_id, title, start_datetime, end_datetime, et created_by ne peuvent pas avoir de valeur NULL. La clé étrangère created_by référence un utilisateur avec des actions CASCADE sur DELETE et UPDATE. La clé étrangère calendar_id référence un calendrier avec des actions CASCADE sur DELETE et UPDATE.

Feeds :

Les attributs de la relation Feeds comprennent un identifiant unique (id), un nom (name), une description (description), et une date de création (created_at). En termes de contraintes d'intégrité, la clé primaire id doit être unique pour chaque Feed. Le champ name ne peut pas avoir de valeur NULL, garantissant ainsi que chaque flux a un nom défini. La date de création (created_at) est automatiquement générée avec la valeur actuelle lors de la création du flux.

Posts :

Les attributs de la relation Posts comprennent un identifiant unique (id), une clé étrangère pour le flux (feed_id), un contenu (content), un créateur (created_by), et une date de création (created_at). Ces attributs sont nécessaires pour gérer les publications. En termes de contraintes d'intégrité, la clé primaire id doit être unique pour chaque publication. Les champs feed_id et created_by ne peuvent pas avoir de valeur NULL. La date de création (created_at) est automatiquement générée avec la valeur actuelle lors de la création de la publication. Les clés étrangères feed_id et created_by référencent respectivement les clés primaires de la relation Feeds et Users avec des actions CASCADE sur DELETE et UPDATE.

TaskLists :

Les attributs de la relation TaskLists comprennent un identifiant unique (id). Cet attribut est nécessaire pour gérer les listes de tâches. En termes de contraintes d'intégrité, la clé primaire id doit

être unique pour chaque liste de tâches. Cela garantit qu'il n'y a pas de doublons dans les listes de tâches.

Tasks :

Les attributs de la relation Tasks comprennent un identifiant unique (`id`), un titre (`title`), un statut (enumérée comme 'ToDo', 'InProgress', 'Testing', 'Done' avec une valeur par défaut 'ToDo'), un résumé (`summary`), une priorité (`priority`), un pourcentage d'avancement (`percent_complete`), un assigné (`assignee`), un créateur (`created_by`), un identifiant de liste de tâches (`task_list_id`), et une date de création (`created_at`). En termes de contraintes d'intégrité, la clé primaire `id` doit être unique pour chaque tâche. Le champ `title` ne peut pas avoir de valeur NULL. Le champ `status` est une énumération avec des valeurs 'ToDo', 'InProgress', 'Testing', 'Done'. Le champ `assignee` peut être NULL par défaut, mais s'il est renseigné, il doit référencer un utilisateur existant grâce à la clé étrangère `assignee` qui référence la clé primaire de la relation Users avec des actions CASCADE sur DELETE et UPDATE. De même, les clés étrangères `created_by` et `task_list_id` référencent respectivement les clés primaires de la relation Users et TaskLists avec des actions CASCADE sur DELETE et UPDATE.

Teams :

Les attributs de la relation Teams comprennent un identifiant unique (`tid`), un nom (`name`), un identifiant de flux (`feed_id`), un identifiant de calendrier (`calendar_id`), un identifiant de liste de tâches (`task_list_id`), et une date de création (`created_at`). En termes de contraintes d'intégrité, la clé primaire `tid` doit être unique pour chaque équipe. Le champ `name` ne peut pas avoir de valeur NULL. Les clés étrangères `feed_id`, `calendar_id`, et `task_list_id` peuvent être NULL, mais si elles sont renseignées, elles doivent référencer des enregistrements existants dans les tables Feeds, Calendars, et TaskLists respectivement, grâce aux contraintes de clés étrangères. Ces contraintes assurent l'intégrité des données liées et définissent des actions CASCADE sur DELETE et UPDATE pour maintenir la cohérence des données en cas de modifications dans les tables référencées.

TeamMembers :

Les attributs de la relation TeamMembers comprennent un identifiant d'équipe (`tid`) et un identifiant d'utilisateur (`uid`), ainsi qu'une date de création (`created_at`). En termes de contraintes d'intégrité, les champs `tid` et `uid` ne peuvent pas avoir de valeur NULL et forment ensemble la clé primaire de la relation. Les clés étrangères `tid` et `uid` référencent respectivement les clés primaires de la relation Teams et Users avec des actions CASCADE sur DELETE et UPDATE.

Conversations :

Les attributs de la relation Conversations comprennent un identifiant unique auto-incrémenté (`cid`), un nom (`name`), un type (enumérée comme 'team' ou 'private'), et une date de création (`created_at`). En termes de contraintes d'intégrité, le champ `name` ne peut pas avoir de valeur NULL. Le champ

type est une énumération qui ne peut prendre que les valeurs 'team' ou 'private'. La contrainte check(type IN ('team', 'private')) assure que seules ces valeurs sont autorisées pour le champ type. La clé primaire cid garantit l'unicité des conversations.

ConversationParticipants :

Les attributs de la relation ConversationParticipants comprennent un identifiant de conversation (cid) et un identifiant d'utilisateur (uid), ainsi qu'une date de création (created_at). En termes de contraintes d'intégrité, les champs cid et uid ne peuvent pas avoir de valeur NULL et forment ensemble la clé primaire de la relation. Les clés étrangères cid et uid référencent respectivement les clés primaires des relations Conversations et Users avec des actions CASCADE sur DELETE et UPDATE.

ConversationMessages :

Les attributs de la relation ConversationMessages comprennent un identifiant unique auto-incrémenté (mid), un identifiant de conversation (cid), un identifiant d'utilisateur (uid), le contenu du message (content), et une date de création (created_at). En termes de contraintes d'intégrité, le champ cid, uid, et content ne peuvent pas avoir de valeur NULL. La clé primaire mid garantit l'unicité des messages. Les clés étrangères cid et uid référencent respectivement les clés primaires des relations Conversations et Users avec des actions CASCADE sur DELETE et UPDATE.

TeamConversations :

Les attributs de la relation TeamConversations comprennent un identifiant de conversation (cid) et un identifiant d'équipe (tid). En termes de contraintes d'intégrité, les champs cid et tid ne peuvent pas avoir de valeur NULL et forment ensemble la clé primaire de la relation, garantissant ainsi qu'une conversation ne peut être associée qu'à une seule équipe. Les clés étrangères tid et cid référencent respectivement les clés primaires des relations Teams et Conversations avec des actions CASCADE sur DELETE et UPDATE.

Notifications :

Les attributs de la relation Notifications comprennent un identifiant unique auto-incrémenté (id), un identifiant d'utilisateur (uid), le contenu de la notification (content), le statut de lecture (read_status), le statut d'envoi (sent_status), et une date de création (created_at). En termes de contraintes d'intégrité, les champs uid, content, read_status, et sent_status ne peuvent pas avoir de valeur NULL. Le champ uid est une clé étrangère référençant la clé primaire de la relation Users avec des actions CASCADE sur DELETE et UPDATE.

Invitations :

Les attributs de la relation Invitations comprennent un identifiant unique auto-incrémenté (iid), un identifiant d'utilisateur (user_id), un identifiant d'équipe (team_id), un token (token) et

une date de création (`created_at`). En termes de contraintes d'intégrité, les champs `user_id`, `team_id`, et `token` ne peuvent pas avoir de valeur NULL. La clé étrangère `team_id` référence la clé primaire de la relation Teams avec des actions CASCADE sur DELETE et UPDATE. De même, la clé étrangère `user_id` référence la clé primaire de la relation Users avec des actions CASCADE sur DELETE et UPDATE.

Génération des données :

Toutes les tables de la base de données sont populées avec des données générées avec le site web Mockaroo et Github Copilot. Nous avons générés 100 utilisateurs avec des prénoms, noms de famille et courriels aléatoires à l'aide de Mockaroo (tous ces utilisateurs ont le même mot de passe 'password'). Nous avons ensuite créé des fonctions Python pour générer des requêtes MySQL qui créent des conversations privées avec des utilisateurs aléatoires de ceux existants. Avec l'aide de Github Copilot, nous avons finalement générés des équipes (dans lesquels des utilisateurs existants aléatoires sont ajoutés), des messages dans les conversations, des publications (« posts ») dans les flux d'activités (« feeds ») des équipes, des événements dans les calendriers des équipes et des utilisateurs et des tâches dans les tableaux de tâches (tableaux kanban) des équipes.

La procédure « PopulateDatabase » de notre fichier « /database/procedures.sql » permet de donc d'utiliser ces données générées pour populer les tables de notre base de données.

4 Création des requêtes et des routines

Gâchettes : Nous utilisons 12 gâchettes dans notre projet. Certaines sont utilisées comme contraintes d'intégrités où que les mots clés SQL ne repondait pas à nos besoins. En autre, nous avons des gâchettes qui associent un calendrier aux nouveaux utilisateurs et équipes, qui associe à une nouvelle équipe un flux d'activité (*feed*), une conversation et une liste de tâches, qui rajoute un nouveau équipier à toutes les conversations de l'équipe, qui rajoute les équipiers d'une équipe à une nouvelle conversation et qui fait le nettoyage lorsqu'un membre d'une équipe part de l'équipe ou que l'équipe est dissoute. Nous avons aussi des gâchettes qui créer des notifications pour chaques nouveaux messages, invitations et membres d'équipe. Notre serveur fait de l'attente active (*polling*) sur ces notifications et les envoie aux clients connectés en ligne.

Procédure : Nous avons une seule procédure dans notre application. Cette procédure appellée *CreatePrivateConversationWithUsers* regarde si un utilisateur avec un certain courriel exist, s'il existe, il crée une conversation privée avec les deux utilisateurs.

Fonctions : Nous avons quatre fonctions utilitaires. Deux d'entre eux aide à la création en créant une équipe pour un utilisateur en particulier, ainsi que de créer une conversation pour une équipe. Les deux autres regardent si un utilisateur fait partie d'une certaine équipe ou s'il fait partie d'une certaine conversation.

Requêtes : Nos requêtes sont appelés avec le connecteur MySQL Python dans les fichiers contenus

dans le dossier /server/database/tables. Certaines fonctions font des requêtes sur place (*inline*), tandis que d'autre fonction utilise les requêtes définies dans le fichier /server/database/queries.py. Certaines requêtes notables sont : *GET_CALENDAR_EVENT_TEAM_ID_QUERY* qui retrouve les événements associés aux calendrier d'une équipe en fesant une double jointure, *update_task* qui mets à jour une tâche particulière en permettant n'importe quel sous-ensemble de champs à mettre à jour et *update_event* qui fait la même chose, mais pour un événement de calendrier.

5 Indexation et optimisation du système

Pour optimiser l'exécution des requêtes dans notre système, nous avons pris en compte les requêtes les plus fréquemment utilisées par les utilisateurs et anticipé la charge de travail attendue. Après avoir analysé ces requêtes, nous avons identifié quatre index à créer afin d'optimiser les performances de la base de données.

Tout d'abord, pour la table des utilisateurs, nous avons créé un index sur la colonne de l'adresse courriel (CREATE INDEX users_index ON Users (email)). Cela permet d'accélérer les requêtes de recherche d'utilisateurs par leur adresse e-mail, ce qui est une opération courante dans notre système.

En ce qui concerne les notifications, nous avons créé un index composite sur les colonnes uid, read_status et sent_status (CREATE INDEX notifications_index ON Notifications (uid, read_status, sent_status)). Cet index permet d'optimiser les requêtes de récupération des notifications en fonction de l'utilisateur concerné, de leur état de lecture et de leur état d'envoi.

Enfin, pour les invitations, un index a été créé sur les colonnes team_id et user_id (CREATE INDEX invitations_index ON Invitations (team_id, user_id)). Cela permet d'améliorer les performances lors de la recherche d'invitations associées à une équipe spécifique et à un utilisateur donné.

Il convient de mentionner que la plupart des requêtes utilisent des clés primaires ou des clés secondaires pour la recherche. Comme MySQL crée automatiquement des index sur ces clés par défaut, nous n'avons pas eu besoin de créer plusieurs index supplémentaires.

6 Normalisation des relations

Les relations de la base de données ont été normalisées pour garantir leur cohérence et leur intégrité. Chaque relation, y compris "Calendars", "Users", "CalendarEvents", "Feeds", "Posts", "TaskLists", "Tasks", "Teams", "TeamMembers", "Conversations", "ConversationParticipants", "ConversationMessages", "TeamConversations", "Notifications", et "Invitations", a été structurée selon les normes de la base de données. Elles sont toutes en 1NF, 2NF, 3NF et BCNF, ce qui signifie qu'elles ne contiennent pas de dépendances fonctionnelles non triviales et qu'elles sont exemptes de redon-

dances et d'anomalies.

7 Sécurité de la base de données

Dans le cadre de la sécurisation de notre système, nous avons identifié les risques potentiels de perte ou de vol des données et mis en place des mesures appropriées pour garantir sa sécurité. Tout d'abord, nous avons veillé à ce que les mots de passe des utilisateurs soient stockés de manière sécurisée. Pour ce faire, nous avons utilisé des techniques de hachage pour transformer les mots de passe en une forme irréversible. Cela garantit que même en cas de violation, les mots de passe des utilisateurs restent protégés.

De plus, pour la fonctionnalité d'invitation d'utilisateurs, nous avons créé un token haché qui stocke des informations sur l'utilisateur invité. L'utilisation de ce token garantit que lorsque l'utilisateur accepte une invitation, son identité est vérifiée. Cela nous assure que seuls les utilisateurs authentifiés peuvent rejoindre des groupes ou des équipes.

Cependant, un potentiel problème de sécurité concerne le stockage des messages entre les utilisateurs et les groupes. Actuellement, ces messages sont stockés sans être hachés. Pour remédier à cela, nous envisageons de mettre en œuvre le hachage des messages textes avant de les stocker. Cette mesure ajouterait une couche supplémentaire de sécurité, garantissant que les messages restent confidentiels même en cas de violation.

Pour renforcer la sécurité de notre base de données, nous utilisons des requêtes paramétrées avec Python et MySQL. Cette méthode empêche toute injection de code SQL malveillant, assurant ainsi la sécurité de notre système contre les attaques par injection SQL. Voici un example d'une telle requête :

```
cursor.execute("SELECT * FROM CalendarEvents WHERE id = %(id)s",
dict(id=event_id))
```

8 Implémentation de la logique d'affaire

Dans le cadre de notre projet, nous avons développé les fonctions Python et les routes Flask nécessaires pour mettre en œuvre la logique métier. Cette couche joue un rôle crucial en assurant la communication entre la base de données et l'interface utilisateur, avec plusieurs composants distincts.

Tout d'abord, l'API expose les endpoints utilisés par l'interface utilisateur, avec des routes dédiées à chaque fonctionnalité de l'application. Ces routes reçoivent les requêtes HTTP et les dirigent vers les services correspondants pour un traitement ultérieur.

Les services représentent le cœur de la logique métier. Chaque fonctionnalité a son propre service,

tel que TeamService pour la gestion des équipes. Ces services traitent les données de la base de données et peuvent interagir avec des services externes si nécessaire.

Les modèles, quant à eux, représentent les données de la base sous forme d'objets Python. Chaque table de la base a son modèle correspondant, comme TeamModel pour la table Team. Ces modèles permettent des opérations de lecture, écriture et mise à jour sur les données de la base de manière abstraite.

La gestion des erreurs est essentielle pour assurer la fiabilité de l'application. Des gestionnaires d'exceptions sont mis en place pour capturer les erreurs inattendues et les gérer de manière appropriée, en affichant des messages significatifs à l'utilisateur ou en les enregistrant dans les journaux de l'application.

L'intégration de SocketIO permet la mise en place d'un serveur en temps réel pour la gestion des messages instantanés. Cela facilite la communication bidirectionnelle entre le serveur et le client, essentielle pour les fonctionnalités de chat en temps réel.

Enfin, la partie de connexion à la base de données gère la connexion et les opérations sur la base. Des méthodes sont définies pour effectuer des opérations telles que la création, la lecture, la mise à jour et la suppression de données, utilisant des requêtes SQL pour une interaction efficace et sécurisée avec la base.

En plus, l'ajout des types de variables dans Python a été un élément crucial pour contrôler et valider les données qui circulent entre les différents niveaux de l'application. En définissant des types de données précis pour les paramètres des fonctions, les attributs des modèles et les valeurs de retour, nous avons pu spécifier clairement ce qui est attendu de la base de données comme résultat et ce qui est attendu du client comme données d'entrée. Cela nous a permis de détecter les éventuelles incohérences ou erreurs de données dès la phase de développement, en fournissant une validation statique des types.

9 Implémentation de l'interface utilisateur

Pour implémenter l'interface utilisateur de notre projet de collaboration, nous avons opté pour l'utilisation de React. L'interface utilisateur est conçue de manière à offrir une expérience fluide et intuitive à l'utilisateur. Voici comment nous avons organisé l'interface :

- Authentification et Enregistrement : L'utilisateur est d'abord dirigé vers une page où il peut se connecter s'il possède déjà un compte ou s'enregistrer s'il n'en a pas.

- Dashboard Utilisateur : Une fois connecté, l'utilisateur est redirigé vers son tableau de bord personnel. Ce tableau de bord contient un sidebar avec des éléments de navigation permettant d'accéder aux fonctionnalités principales de l'application.

- Éléments du Sidebar : Le sidebar contient des liens vers différentes sections de l'application, telles

que les équipes, les messages, le whiteboard, les invitations et l'agenda personnel de l'utilisateur.

- Navigation dans les Onglets : Chaque section accessible depuis le sidebar présente des informations pertinentes pour l'utilisateur. Par exemple, dans la section "Équipes", l'utilisateur peut consulter la liste de toutes les équipes auxquelles il appartient. Pour chaque équipe, il peut accéder aux publications, aux membres, aux invitations, aux messages et aux tâches associées.

Des mécanismes de validation sont mis en place pour garantir que seules les données valides sont envoyées au backend. Cela permet d'éviter les erreurs de saisie et les attaques délibérées. Par exemple, il y a la validation des adresses e-mail et des noms d'équipe. Cette validation côté client garantit que seules les données conformes aux critères définis sont transmises au serveur.

En plus de valider les informations côté client, nous avons mis en place une gestion des erreurs et des notifications de statut. Lorsque l'utilisateur effectue une action, comme créer une équipe ou envoyer un message, des "toasts" s'affichent pour lui fournir un retour d'information instantané sur le succès ou l'échec de son action. Par exemple, un toast informe l'utilisateur du succès de la création d'une équipe, tandis qu'un "toast" approprié lui signale tout problème rencontré.

10 Tests du système

L'assurance de la fonctionnalité et de la fiabilité du système dans son ensemble a été une priorité tout au long du processus de développement. Nous avons effectué une série de tests pour garantir que les trois niveaux du système sont intégrés de manière harmonieuse et que la communication entre eux est efficace, même dans des situations d'utilisation normales et potentielles d'erreurs.

Un aspect important des tests a été la validation des entrées utilisateur. Toutes les formes d'entrée utilisateur ont été contrôlées pour s'assurer qu'elles correspondent aux types de données attendus. Par exemple, si un champ de la page web devait recevoir un nombre, nous nous sommes assurés qu'un message d'erreur approprié serait affiché sous le champ si l'utilisateur entre une chaîne de caractères ou une valeur invalide.

En plus de cela, toutes les interactions entre la partie client et la partie serveur ont été rigoureusement testées pour garantir leur bon fonctionnement. Nous avons vérifié que les requêtes et les réponses étaient correctement traitées et que les données étaient échangées de manière sécurisée et cohérente.

Pour améliorer la convivialité de l'application, nous avons également intégré des messages instantanés pour les appels d'API [1]. Ces messages contextuels apparaissent à l'écran pour indiquer à l'utilisateur si une action a été effectuée avec succès ou s'il y a eu un échec. Cela permet à l'utilisateur de comprendre rapidement le statut de ses actions et de résoudre les problèmes éventuels.

1. interface de programmation d'application (Application Programming Interface)

11 Accessibilité du système

Notre application a été conçue en gardant à l'esprit la diversité des utilisateurs qui pourraient l'utiliser, y compris ceux ayant des besoins spécifiques ou des contraintes techniques. En tant que plateforme de collaboration destinée à être utilisée simultanément par plusieurs étudiants en temps réel, nous avons accordé une attention particulière à son accessibilité.

L'une des caractéristiques essentielles de notre application est sa capacité à permettre une communication en temps réel entre les utilisateurs. Cette fonctionnalité permet à plusieurs utilisateurs d'interagir et de collaborer simultanément, renforçant ainsi son utilité dans le contexte de la collaboration d'équipe au sein de l'Université Laval. En ce qui concerne l'accessibilité visuelle, nous avons intégré des icônes dans toute l'application afin d'aider les utilisateurs ayant des troubles de vision à naviguer plus facilement et à comprendre le contenu. Cela comprend l'utilisation de contrastes élevés et la mise en évidence des éléments importants pour améliorer la lisibilité. Une amélioration que nous envisageons est l'utilisation de balises sémantiques pour structurer le contenu de l'application. Cela aidera non seulement les utilisateurs ayant des difficultés de vision, mais également ceux qui utilisent des technologies d'assistance pour naviguer sur le web.

12 Gestion de l'équipe et organisation du travail

La gestion de l'équipe et l'organisation du travail ont été des éléments cruciaux pour mener à bien ce projet collaboratif. Nous avons pris le temps nécessaire pour évaluer différentes idées de projet, et finalement, nous avons choisi celle d'une plateforme collaborative. Face à la complexité du projet, nous avons ensuite veillé à maintenir une coordination efficace et une communication transparente au sein de l'équipe tout au long du processus de développement.

Des réunions régulières ont été planifiées et tenues tout au long du projet. Ces rencontres hebdomadaires ont permis à chaque membre de l'équipe de partager les progrès réalisés, les défis rencontrés et les solutions proposées. Cela a favorisé une compréhension commune du statut du projet et a permis d'identifier rapidement les éventuels obstacles à surmonter.

Nous avons mis en place des processus clairs pour assurer une intégration harmonieuse des composants individuels. Par exemple, l'utilisation de 'GitHub' [2] pour gérer les 'issues' [3] et les 'pull requests' [4] a facilité la collaboration et la révision du code entre les membres de l'équipe.

Il convient également de noter que différentes tâches ont été attribuées en fonction des spécialités et des intérêts de chaque membre de l'équipe. Certaines tâches se concentraient davantage sur la partie base de données, d'autres sur l'interface utilisateur, tandis que d'autres encore implémentaient des fonctionnalités touchant toutes les couches du projet.

2. plateforme open source de gestion de versions et de collaboration
3. des éléments créés dans un référentiel pour planifier, discuter et suivre le travail
4. demande de fusion d'une série de modifications d'une branche à une autre