Gabriel St-Pierre (111 178 727)
Olivier Gingras (111 181 997)
Victor Mariton (111 246 554)

NRC : 15510

Projet de session :
Cooky

Travail présenté à
Richard Khoury
Pour le cours
Modèle et langages des bases
de données pour ingénieurs (GLO-2005)

Université Laval
Le mercredi 17 avril 2019

Table des matières

1 - Énonciation du problème

L'application qui est produite est un site de recette permettant la création et le partage de recettes, mais également l'achat des ingrédients requis pour ladite recette. Les ingrédients choisis sont ensuite mis dans un panier et l'utilisateur peut ensuite effectuer une commande, qui lui sera livrée à domicile. L'application agit également en tant que plateforme sociale en permettant de commenter, aimer et coter les recettes. Ces dernières sont constituées d'étapes, d'ingrédients ainsi que d'une description. Les ingrédients sont identifiés par un nom et une quantité leur est attribuée, ce qui en détermine le prix. L'utilisateur peut choisir les ingrédients qu'il désire mettre dans son panier. Le panier servira à procéder au paiement. L'utilisateur peut aussi créer ses propres recettes et ces nouvelles recettes sont disponibles pour tous les utilisateurs. L'utilisateur peut rechercher un ingrédient parmi tous les ingrédients disponibles. Dans le cas d'un changement d'adresse, l'utilisateur peut toujours mettre ses informations à jour. Les exigences de l'application seront donc les suivantes :

- Rechercher une recette
- Rechercher un ingrédient
- Afficher une fiche détaillée pour chaque recette
- Posséder un système de connexion
- Visualiser les recettes créées et aimées par l'utilisateur
- Avoir des instructions pour chaque recette
- Avoir une liste d'aliments pour chaque recette
- Affichage des caractéristiques de chaque produit (aliments)
- Noter les recettes
- Commenter les recettes
- Aimer une recette
- Confirmer un achat et créer une commande
- Livraison de la commande
- Modifier les informations de l'utilisateur

2 - Responsabilités des trois niveaux

2.1 - Fonctionnalités JavaScripts (Interface utilisateur)

La principale fonctionnalité JavaScript est la création de recettes. Celle-ci permet de créer une nouvelle recette qui contient un nom, une description, des étapes ainsi qu'une liste d'ingrédients. Par la suite, cette recette peut être supprimée par l'utilisateur qui la créez. Les autres utilisateurs peuvent aimer la recette afin qu'elle soit disponible dans leur section de recettes favorites, la commenter ainsi que lui attribuer une note.

Une autre fonctionnalité est l'ajout d'ingrédients dans le panier. Cette fonctionnalité est utile lorsque l'utilisateur désire faire cette recette, mais qu'il ne dispose pas des ingrédients nécessaires. Il pourra par la suite modifier les quantités ou supprimer des ingrédients selon ses besoins.

Une fois le panier désiré, l'utilisateur n'a qu'à confirmer et envoyer la commande au service. Le client pourra alors visualiser ses commandes en attente ainsi que celles déjà livrées. La date d'envoi de la commande ainsi que sa date d'arrivée et son coût seront affichés.

La recherche d'ingrédient par le nom est une méthode facile qui permet de trouver l'ingrédient recherché. Cette fonctionnalité est également disponible afin de permettre la recherche pour les noms de recettes.

2.2 - Fonctionnalités Python (serveur API REST)

Une des fonctionnalités primaires du serveur était de faire le lien entre la base de données et l'application. L'API agit en tant qu'interface aux données. Elle montre d'abord quelles actions sont permises (ajout, modification, obtenir, etc.) et sous quel contexte (recette, utilisateur, ingrédient, etc.). Elle s'assure ensuite de bien filtrer le format des données entrantes avant leur sauvegarde. L'interface, puisque facilement accessible à tous, ne serait pas assez sécuritaire pour effectuer une telle tâche. Finalement, le serveur retourne les informations de façon regroupée et sous un format compréhensible, clair et uniforme. Il agit donc réellement en tant que pont qui filtre les entrées et les sorties entre l'interface utilisateur et la base de données.

Une deuxième fonctionnalité très importante du serveur est d'assurer la sécurité quant aux actions permises. Premièrement, l'accès à l'application est protégé. Toute personne désirant y accéder doit posséder un compte et s'identifier. Ce processus d'authentification permet

d'étiqueter chaque utilisateur et ainsi pouvoir y effectuer certaines vérifications. L'une d'elles est justement la deuxième fonctionnalité de sécurité, soit l'autorisation. Puisque l'application étiquette chaque utilisateur, elle peut empêcher certaines actions. Par exemple, un utilisateur ne peut pas créer une recette sous le nom de quelqu'un d'autre. Ou encore, il ne peut commander un panier qui ne lui appartient pas.

2.3 - Fonctionnalités SQL (base de données)

Une des fonctionnalités de la base de données SQL est de mettre à jour automatiquement le panier courant de l'utilisateur. De cette façon, l'utilisateur n'a pas besoin de s'occuper de la gestion des paniers. Lors de la création de l'utilisateur, un nouveau panier lui est assigné. De plus, lors de la confirmation de la commande, un nouveau panier lui est assigné afin de permettre l'achat d'autres produits, malgré la commande en cours.

Une fonctionnalité qui est pratique pour empêcher les problèmes de livraison est le fait de ne pas pouvoir modifier le panier une fois commandé. De cette façon, le risque d'erreur lors de la livraison est minimisé en ne permettant pas les changements de dernière minute. Cela permet également de sécuriser le panier en y empêchant la modification du prix. Ainsi, une commande de 100$ sera assurée de ne pas comprendre plus de 100$ de produits après achat.

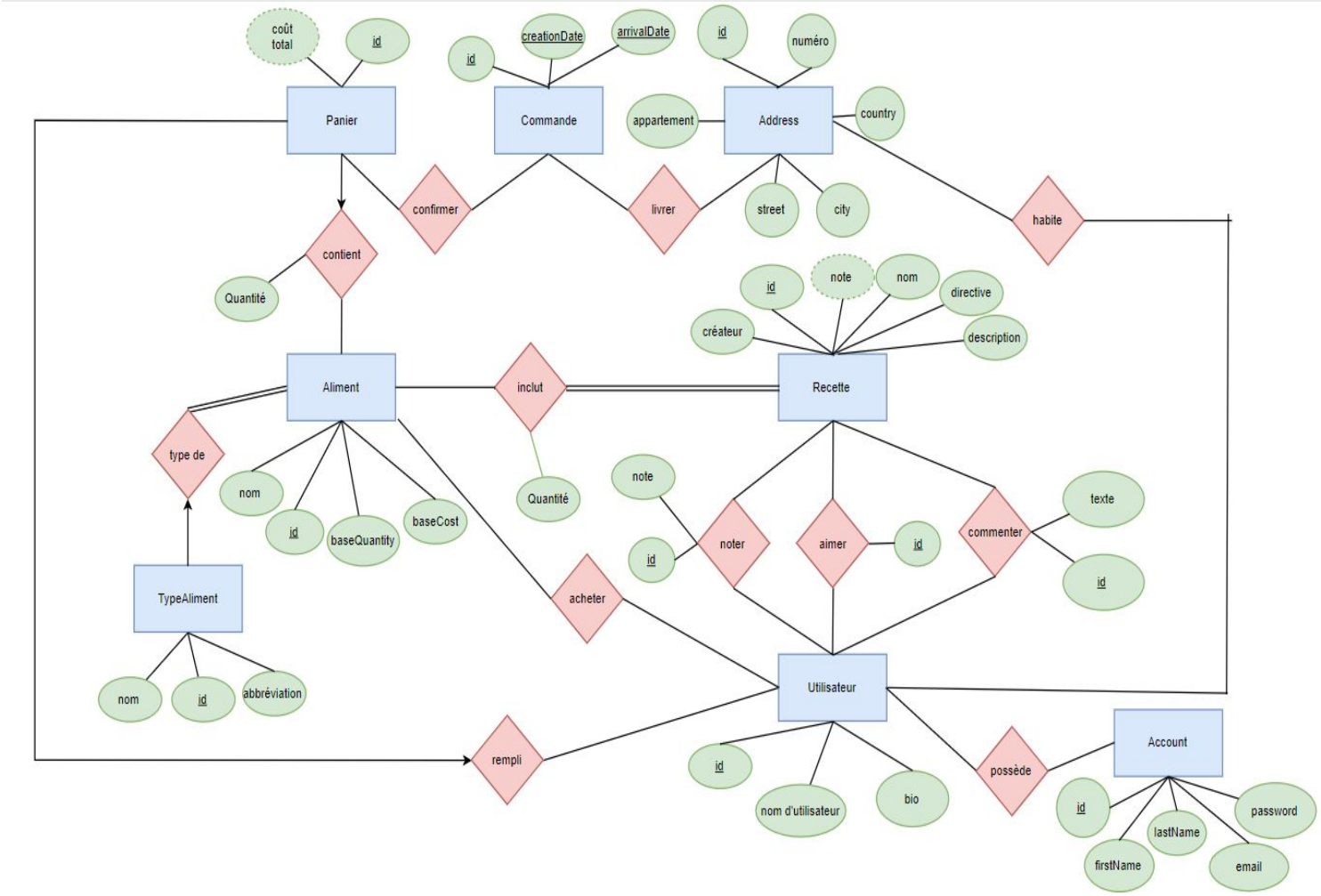
Troisièmement, la couche de persistance possède également des gâchettes qui s'occupent de calculer automatiquement les sommes et les moyennes de différentes valeurs. Par exemple, les calculs des prix du panier (total et par produit) se font à chaque modification relative au panier. De cette façon, l'application s'assure qu'aucune erreur de calcul n'est produite et que le calcul est basé sur les données réelles et actuelles. De plus, lorsque l'utilisateur note une recette, une gâchette est activée et recalcule la moyenne totale des cotes. De cette façon, la note moyenne des recettes reste toujours à jour.

3 - Modélisation

Afin de bien visualiser les différentes responsabilités et relations entre les entités, il est important de bien modéliser les données. Pour ce faire, deux types de diagrammes ont été construits afin de représenter les données sous différents niveaux d'abstraction.

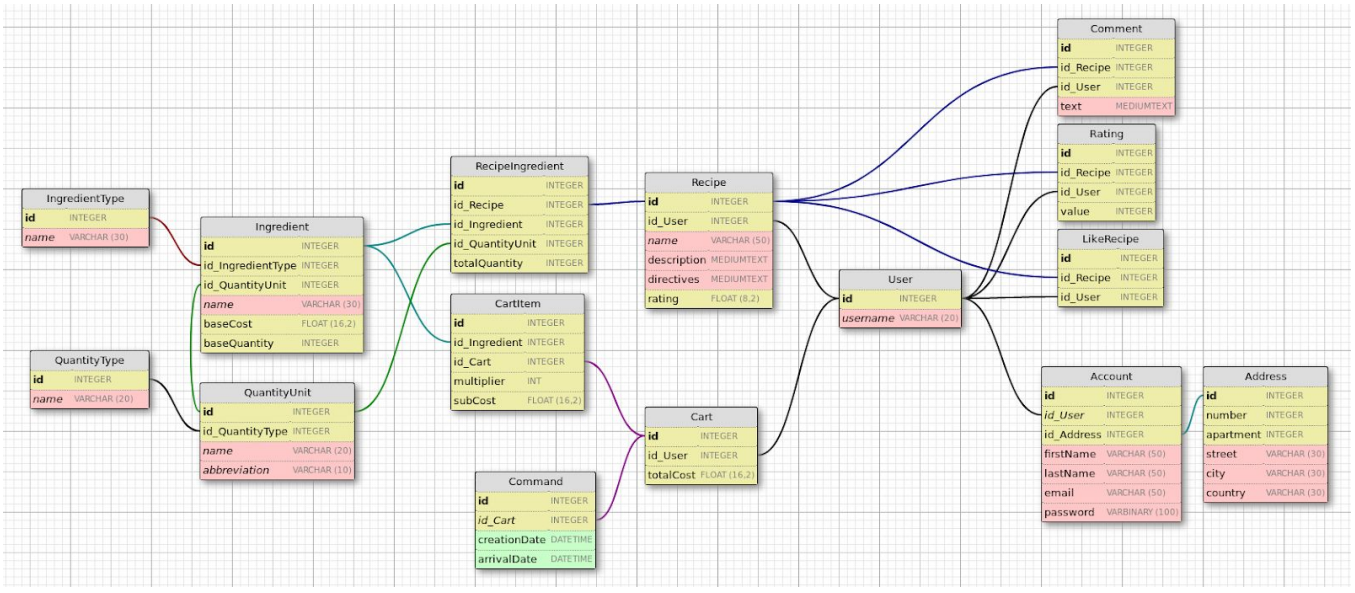
3.1 - Modèle entité-relation

Le modèle entité-relation définit de façon large et conceptuelle les différents types de données (entités, objets) qui définissent les principales fonctionnalités de l'application. De plus, on y notera les attributs ainsi que leur type (clé primaire, composée, etc.).



3.2 - Modèle relationnel

Le diagramme relationnel, plus abstrait et plus représentatif de la base de données, permet de facilement visualiser les types des données stockées ainsi que leurs relations. Il devient donc facile de connaître le chemin à suivre afin d'obtenir certaines informations.



4 - Implémentation de la base de données

Afin de créer la base de données, une série d'étapes a été effectuée. Celle-ci passe notamment par l'initialisation des tables, l'ajout de contraintes, la création de gâchettes, la création de données statiques et finalement, le remplissage de données aléatoires.

4.1 - Création des tables

La création des tables a été effectuée exactement selon le diagramme relationnel. Certains attributs ont été marqués comme étant uniques afin de prévenir certaines incohérences. Par exemple, un utilisateur ne pourrait avoir deux « account » différents, car cela résulterait en deux mots de passe différents. De plus, deux ingrédients ne peuvent logiquement avoir le même nom, surtout si c'est la seule façon de les identifier.

4.2 - Ajout de contraintes

Certaines contraintes de base ont été définies directement dans les tables, comme les contraintes uniques des attributs ainsi que les clés primaires et étrangères. Par contre, les contraintes de groupes d'attributs doivent être effectuées à part. Celles-ci sont particulièrement importantes au niveau des tables d'association (« RecipeIngredient », « RecipeLike », « Comment », etc.). Pour la plupart d'entres-elles, il faut empêcher les duplicatas au niveau des paires associées. Par exemple, un utilisateur ne pourrait pas aimer une même recette deux fois. Par contre, il aurait le droit de la commenter plusieurs fois. Ce ne sont donc pas toutes les tables d'association qui possèdent cette contrainte.

4.3 - Création de gâchettes

Les principales gâchettes créées sont décrites à la section 2.3 du présent rapport. Celles-ci ont dû être définies autant lors de l'insertion de tuples que lors de la modification ou suppression. Par contre, puisque ces gâchettes sont relativement simples, elles n'ont pas été exportées dans des procédures à part. De plus, les gâchettes au niveau de la suppression ont été sécurisées à l'aide de la commande « IFNULL » au cas où plus aucun tuple ne serait trouvé.

4.4 - Remplissage des tables statiques

Un point important de l'application est qu'elle doit contenir certaines informations existantes du monde réel qui ne doivent pas être modifiées. Ces informations forment des ensembles finis et peuvent donc être remplies à la main. Par exemple, il existe seulement un certain nombre de types de mesures (millilitres, grammes, tasses, etc.) et seulement certains groupes d'aliments (viandes, légumes, produits laitiers, etc.). La seule exception a été le remplissage des ingrédients, qui a été généré aléatoirement en raison de la grandeur de l'ensemble.

4.5 - Remplissage des autres données

Le reste des données appartenant aux tables dites « dynamiques » (au contenu changeant) a été généré aléatoirement à l'aide d'un script.

5 - Indexation, normalisation et optimisation

Afin d'améliorer la cohérence et les performances de la base de données, certains indices, normalisations et optimisations ont été ajoutés.

5.1 - Indexation

En ce qui concerne les indices, les tables utilisent les indices uniques en raison des contraintes uniques sur les attributs définis en SQL. Les index uniques du projet sont nécessaires afin d'éviter les duplications (voir les sections 4.1 et 4.2). De plus, la mise en place d'IDs uniques pour chaque table est un exemple de « non-clustered index ». De cette façon, il est facile d'accéder à chacune des tables selon la référence de l'ID de l'autre table.

5.2 - Normalisation

L'ensemble des tables de la base de données sont au moins en quatrième forme normale. Les dépendances multi valeurs ne sont pas permises. La création de tables avec peu d'attributs permet d'arriver à cette situation. Comme il s'agit de la quatrième forme normale, il s'agit donc aussi de la forme normale de Boyce-Codd. En effet, les tables SQL sont faites de façon à ce qu'il existe toujours une super clé pour la table.

5.3 - Optimisation

Les requêtes d'insertions et de modifications étant peu utilisées, il n'était pas nécessaire de les optimiser. Par contre, les requêtes de balayage et d'égalité demeurent les éléments centraux de l'application. Les noms de recettes et d'ingrédients étant uniques, les tables sont indexées selon un arbre B+ non-groupé, ce qui permet une recherche plus rapide. Pour les autres tables, puisque les accès se font plus directement par les IDs, aucune optimisation n'était vraiment nécessaire.

Finalement, l'ensemble des requêtes restent plutôt simples, ce qui ne nécessite pas vraiment d'effort au niveau de l'optimisation. Celle-ci a plutôt été implémentée au niveau de l'interface à l'aide de requêtes parallèles et asynchrones.

6 - Implémentation de la logique d'affaires

La logique d'affaires a été implémentée à l'aide de la librairie Flask. Il a été décidé d'en faire un API plutôt qu'une application rendue côté serveur afin de permettre de nombreux avantages. Premièrement, l'API permet des requêtes en parallèle à différentes ressources, et ces appels se font à des URL précis (selon le standard REST). Ainsi, les connexions se font de façons claires et efficaces, et sont groupées par ressources plutôt que par pages de présentation.

De plus, afin de permettre les appels en parallèle, le système de serveur Gunicorn a été utilisé afin d'englober Flask. Ainsi, les requêtes sont mises en file d'attente afin d'éviter une surcharge.

Troisièmement, le serveur utilise un système de logique à trois couches. La première est formée des contrôleurs, qui s'occupent d'acheminer l'information entre l'interface et le serveur, et de gérer les codes de retour. La deuxième est formée des DAO (« Data Access Objects »), qui s'occupent de communiquer avec la base de données. Ce sont ces derniers qui définissent les requêtes SQL. La troisième est formée des modèles, qui permettent de facilement représenter les données (tables) sous forme de classe python. Ces objets sont donc facilement transportables et transformables entre chacune des couches.

Voici la liste des routes permettant d'effectuer certaines actions prédéfinies :

- GET /users retourne (retourne tout les users)
- GET /users/<id>/account (retourne l'account du user id)
- GET /users/<id> (retourne l'user par son id)
- GET /users/<name> (retourne l'user par son nom)
- GET /users/<id>/recipes (retourne les recettes du user id)
- GET /users/<id>/likes (retourne les recettes aimées du user id)
- GET /users/<id>/address (retourne les adresses du user id)
- GET /users/<id>/ratings (retourne la note du user)
- GET /users/<id>/cart (retourne le panier courant de l'user)
- GET /users/<id>/commands (retourne les commandes du user)
- GET /recipes (retourne toutes les recettes ou la recette selon le nom)
- GET /recipes/<id> (retourne la recette selon l'id)
- GET /recipes/<id>/ingredients (retourne la liste des ingrédients de la recette)

- GET /recipes/<id>/comments (retourne les commentaires de la recette)
- GET /ingredients (retourne tous les ingrédients ou l'ingrédient selon le nom)
- GET /ingredients/<id>/mesures (retourne la mesure selon l'id de l'ingrédient)
- GET /cart (retourne tous les paniers)
- GET /cart/<id> (retourne le panier de l'id)
- GET /cart/<id>/items (retourne les items du panier selon l'id)
- POST /users (création d'un nouveau user)
- POST /users/<id>/account (création d'un compte pour l'user id)
- POST /cart/<id>/command (création d'une commande pour le panier id)
- POST /cart/<id>/items (création d'un item au panier courant)
- POST /address (création de l'adresse de l'utilisateur)
- POST /recipes (création d'une nouvelle recette)
- POST /recipes/<id>/like (aime la recette id)
- POST /recipes/<id>/ratings (ajout d'une note pour la recette
- POST /recipes/<id>/comment (ajout d'un nouveau commentaire à la recette)
- PUT /users/<id>/email (modifie l'email du user)
- PUT /users/<id>/password (modifie le password du user)
- PUT /users/<id>/city (modifie la ville du user)
- PUT /users/<id>/street (modifie le nom de rue du user)
- PUT /users/<id>/apartment (modifie le numéro d'appartement)
- PUT /users/<id>/doorNumber (modifie le numéro de porte)
- PUT /recipes<id>/name (modifie le nom de la recette)
- PUT /recipes<id>/directives (modifie les directives de la recette)
- PUT /recipes<id>/ingredientQuantity (modifie la quantité d'ingrédients)
- PUT /recipes/<id>/ratings (modifie la note à la recette)
- PUT /recipes/<id>/comments (modifie les commentaires de la recette)
- PUT /cart/id_cart/items/id_ingredient (modifie la quantité de l'ingrédient id_ingredient du cart id_cart)
- DELETE /recipes/<id_recipes> (supprime la recette)
- DELETE /recipes/<id>/likes (supprime la recette de la section recette aimée)
- DELETE /cart/<id_cart>/items/<id_ingredients> (supprime l'ingrédient id du cart id)

7 - Implémentation de l’interface utilisateur

Afin d’accélérer le développement par la réutilisation de composantes, il a été décidé d’utiliser le framework Javascript VueJS. Celui-ci permet de définir les pages qui seront accessibles aux utilisateurs et de présenter les données pertinentes en communiquant avec le serveur par requêtes AJAX.

Voici la liste des pages définies ainsi que leur raison d’être :

- /home (début de l’application)
- /signup (permet de créer un compte)
- /login (permet de se connecter pour accéder à des fonctionnalitées)
- /recipes (permet de voir toutes les recettes)
- /recipes/create (permet de créer une recette)
- /recipes/:id (permet de voir une recette)
- /ingredients (permet de voir la liste d’ingrédients)
- /users/:id (permet de voir les recettes d’un utilisateur ainsi que ses recettes aimées)
- /cart (permet de voir le panier de l’utilisateur courant)
- /commands (permet de voir les commandes de l’utilisateur courant)
- /account (permet de voir les informations du compte de l’utilisateur courant)
- /account/edit (permet de modifier les informations du compte de l’utilisateur courant)
- /logout (permet de se déconnecter)
- toute page inconnue affichera une page 404 NOT FOUND

8 - Sécurité du système

Afin d'assurer une sécurité du système, certains mécanismes ont été mis en place, dont l'authentification (avec autorisation), le hachage de mots de passe ainsi que la création d'utilisateurs aux privilèges restreints.

8.1 - Système d'authentification

Comme expliqué à la section 2.3, l'application est dotée d'un système d'authentification et d'autorisation. Cette technique utilise la technologie de tokens d'accès JWT. Celle-ci permet de générer un token qui sera stocké dans les cookies du client. Ce token contient une version encryptée de l'objet « User » de la base de données. Ainsi, chaque fois qu'un token est envoyé, ce dernier est décrypté afin d'obtenir l'objet « User ». Il devient alors très facile de comparer les IDs afin de voir si celui présent dans le token correspond bel et bien à celui envoyé dans les paramètres de la requête.

Cette solution est sécuritaire car, premièrement, le token était d'une longueur respectable et générée de façon aléatoire, il est pratiquement impossible de trouver un token par hasard, et qui correspondrait à un utilisateur présentement connecté. Deuxièmement, le token expire à chaque 15 minutes, ce qui réduit largement le temps permis pour trouver le token de l'utilisateur.

8.2 - Hachage des mots de passe

Afin d'éviter l'accès aux mots de passe en clair en cas de vol de données, les mots de passe sont toujours hacher afin leur sauvegarde. De plus, un « salt » aléatoire est généré afin d'empêcher toute possibilité de décryptage. La seule façon de comparer les mots de passe est de comparer leurs versions encryptées.

8.3 - Privilèges de la base de données

Afin de limiter davantage les actions permises, le serveur sera connecté sous l'utilisateur « api ». Ce dernier possède des restrictions quant à toute modification des tables « statiques ». De plus, certaines actions non-permises dans l'interface ne le sont pas non plus avec cet utilisateur. Cela permet une deuxième couche de protection afin de contrôler davantage les modifications. De plus, seuls les fournisseurs de produits alimentaires (ceux qui offrent et livrent les paniers contenant les ingrédients) peuvent modifier la date d'arrivée d'une

commande. Cette situation est très logique puisque ce sont les seuls qui sont au courant du moment de la livraison. Il leur sera alors donné un accès au compte « provider ».

8.4 - Non implémenté

Bien qu'il aurait été très souhaitable d'utiliser le protocole HTTPS afin de ne pas envoyer les mots de passe en format texte clair, il a été décidé qu'une telle implémentation dépassait les exigences du projet.

De plus, l'affichage des mots de passe SQL dans les scripts bash pourrait constituer une faille de sécurité. Encore une fois, il a été jugé acceptable dans le cadre du projet de les laisser ainsi.

9 - Organisation, gestion de l'équipe et division des tâches

L'organisation et la gestion d'équipe se sont effectuées à l'aide de la plateforme Discord et de l'outil Trello. En effet, Discord, une plateforme de communication, nous a permis de discuter du projet et de son avancement. L'outil Trello, quant à lui, a permis la création, le suivi et l'assignation des tâches.

Le projet a été décomposé en étapes, qui ont été assignées de la façon suivante :

1. Initialisation du projet : Gabriel St-Pierre
2. Modélisation des données : Gabriel St-Pierre et Olivier Gingras
3. Création des tables, contraintes, gâchettes et remplissage des tables : Olivier Gingras, Gabriel St-Pierre et Victor Mariton
4. Mise en place du serveur : Olivier Gingras et Gabriel St-Pierre
5. Mise en place de l'interface graphique et liaison au serveur : Gabriel St- Pierre et Olivier Gingras
6. Sécurité (authentification et autorisation) : Gabriel St-Pierre
7. Montage des containers Docker : Gabriel St-Pierre
8. Rédaction du rapport : Olivier Gingras et Gabriel St-Pierre

10 - Revue de code

Afin d'accéder au code source de l'application, il suffit de suivre le lien suivant: https://gitlab.com/glo2005-h19-equipe23/app.git. Par contre, le dépôt étant privé, il faut en demander l'accès en contactant les membres de l'équipe et en leur fournissant le nom d'utilisateur du compte Gitlab qui sera utilisé afin d'accéder au code.