1 Communication

Il nous a semblé nécessaire, pour des ventes de proximité, que clients et producteurs puissent directement discuter entre eux. Dans cette partie, nous allons voir comment les outils de communication entre les utilisateurs ont été mis en place. Il y a deux éléments principaux de communication : les messages et le fil d'actualité.

1.1 Messagerie

La messagerie reprend les codes d'une messagerie de base. Chaque utilisateur, qu'il achète ses produits ou qu'il les vende, a accès à son espace messagerie une fois connecté.

1.1.1 Réception (/messagerie/)

Une fois sur la page de sa messagerie, l'utilisateur peut voir la liste des mails présents dans sa boîte de réception. Les messages sont récupérés dans la base de données via une jointure entre la table "messages" et la table "utilisateurs" (voir volet BD). Si le message n'a pas été lu, la ligne commence par [NON LU] écrit en rouge. Ensuite vient le pseudonyme de la personne qui a envoyé le message, l'heure d'envoi, puis l'objet du message indiqué par l'expéditeur. Pour consulter un message, l'utilisateur clique sur l'objet : c'est un lien cliquable qui ajoute /id à la fin de l'URL, avec id l'identifiant du message (voir volet BD). De cette manière, si l'utilisateur a effectivement accès à ce message (vérification effectuée dans la requête SQL pour éviter qu'un utilisateur ne puisse lire tous les messages du site), le message sur lequel il a cliqué lui est montré dans la même fenêtre que sa boîte de réception. Le message est automatiquement marqué comme lu via une requête UPDATE.

Depuis sa boîte de réception, l'utilisateur peut également sélectionner plusieurs messages et les supprimer. J'ai fait en sorte que les messages ne soient pas réellement supprimés de la base de données, mais plutôt masqués pour l'utilisateur, en vue d'un éventuel ajout d'une corbeille. Cet ajout n'a pas été fait, j'ai cependant pris la décision de garder le fait de masquer les messages plutôt que de les supprimer, pour des raisons de modération. Si l'on veut réellement supprimer ces messages, il suffit de remplacer la requête UPDATE par une requête DELETE.

La boîte de réception comporte également un bouton "Écrire un message", qui permet à l'utilisateur d'être redirigé vers la page d'envoi de messages. Il peut également cliquer sur "Répondre" sur le message qu'il est en train de lire s'il en lit un, ce qui va le rediriger vers la page d'envoi de messages avec l'objet automatiquement rempli : "Re : " suivi de l'objet du message de base.

1.1.2 Envoi (/ecrire)

Sur la page d'envoi, l'utilisateur qui souhaite envoyer un message doit indiquer le pseudonyme de la personne à qui il souhaite envoyer le message (s'il n'est pas correct, l'utilisateur aura un message d'erreur), ainsi qu'un objet et le corps du message. J'ai volontairement mis un input de type "text" pour l'écriture du corps du message, plutôt qu'un textarea comme on pourrait s'y attendre, dans le but d'encourager l'utilisateur à écrire des messages plus courts, dans l'esprit de ceux que l'on peut envoyer sur les réseaux sociaux sur une seule ligne, pour réduire la taille de la base de données et diminuer la quantité d'informations à envoyer. Une fois son message écrit et tous les champs nécessaires remplis, l'utilisateur peut appuyer sur le bouton "envoyer". Flask va récupérer les données envoyées par le formulaire avec la méthode POST, et les inscrire dans la base de données.

Ce module d'envoi de messages aurait dû être présent sur les pages des producteurs, mais ce n'est pas le cas. Il est donc présent sur /ecrire, dans la version dans laquelle il aurait dû être sur les pages des producteurs, avec un champ "Destinataire" pour indiquer à qui ce message doit être envoyé.

1.2 Fil d'actualité (/publications)

Une fois connecté, chaque utilisateur peut aller sur son fil d'actualité.

Il peut tout d'abord poster une publication : il entre le message à publier, puis clique sur "Poster". Sa publication est ensuite insérée dans la base de données.

Il y a un système d'abonnement : chaque utilisateur peut "suivre" un autre utilisateur, et verra ainsi ses publications apparaître dans son fil d'actualité. Dans mon code, l'utilisateur se suit automatiquement lui-même, c'est-à-dire que ses propres publications apparaissent dans son fil

d'actualité. Cela n'est pas inscrit dans la base de donnés (table "follow", voir volet BD), mais est fait directement via la requête SQL qui sert à récupérer les messages.

Ainsi, pour chaque publication qu'il peut voir, l'utilisateur a accès au pseudonyme de l'utilisateur qui a posté la publication, ainsi que la date de publication, en plus du message.

Il y a également un système de commentaires pour chaque publication : l'utilisateur peut commenter une publication, et son commentaire sera visible par toutes les personnes qui suivent l'individu qui a posté la publication. Pour récupérer les commentaires des publications, j'ai fait une boucle sur chaque publication qui sera récupérée par l'utilisateur, et pour chaque publication, je fais une requête SQL comportant un JOIN entre la table commentaires et la table utilisateurs (voir volet BD) et en filtrant sur l'identifiant de la publication passée en revue. Il me semble important de parler de la complexité de cette méthode ici : je pense que celle-ci est (trop) lourde : on a du $O(n)$ pour la récupération des publications, puis pour chaque publication on fait une jointure entre deux tables et on filtre les commentaires correspondant à la publication. J'ignore si la jointure est refaite à chaque fois ou si celle-ci est stockée en cache ou quelque chose de la sorte, mais cette méthode me semble au moins être en $O(n^2)$. Je n'ai pas trouvé d'alternative à cette méthode, j'ai cependant conscience que si mon évaluation de la complexité est correcte, cette méthode de récupération des commentaires n'est pas viable avec une quantité importante de publications/commentaires.