**DOCUMENTATION TECHNIQUE**

Lorsque j'ai commencé à travailler sur le projet Ventalis, j'ai accordé une attention particulière à la configuration de mon environnement de travail afin de l'optimiser au maximum. Voici mes réflexions initiales sur la configuration de l'environnement technologique pour le développement de l'application Django Ventalis.

Choix de Django : J'ai délibérément choisi Django en raison de ses nombreux avantages, notamment sa structure claire basée sur le modèle MVC (Modèle-Vue-Contrôleur) et sa réputation de cadre robuste pour le développement web. Django facilite la création d'applications performantes et évolutives.

Configuration de l'environnement virtuel : Au départ, j'ai opté pour PyCharm comme mon environnement de développement principal. En utilisant PyCharm, j'ai pu créer et gérer facilement mon environnement virtuel sans avoir à me soucier de son activation ou de son désactivation. Cela m'a permis de travailler de manière plus fluide et de me concentrer davantage sur le développement du projet. Cependant, j'ai également gardé Visual Studio Code (VSC) ouvert en parallèle, car il offre des extensions intéressantes, telles que la visualisation claire et facile de la base de données SQLite et la gestion du contrôle de version Git. Par contre, j'ai trouvé le débogage plus avancé dans PyCharm.

Gestion des dépendances : J'ai utilisé le fichier requirements.txt pour répertorier toutes les dépendances du projet, y compris les packages Django et autres bibliothèques tierces utilisées. Cette approche m'a permis de gérer facilement les dépendances et assurer la reproductibilité de l'environnement de développement.

Contrôle de version : J'ai utilisé un système de contrôle de version tel que Git pour gérer le code source du projet. Cela m'a permis de suivre les modifications apportées au code, de travailler sur différentes fonctionnalités dans des branches séparées. Même après avoir fusionné certaines branches, j'ai conservé les branches pour des raisons de suivi et de sécurité. Ainsi, je peux toujours revenir en arrière en cas de problème ou pour analyser l'évolution du projet à partir de zéro.

Pour le développement des API, j'ai utilisé Django REST Framework (DRF) avec la package de Simple JWT Token, qui offre des fonctionnalités de token d'accès et de rafraîchissement. J'ai testé les API à l'aide de l'outil Postman, j'ai également configuré la durée de validité des tokens. Bien que j'aie défini une expiration de 120 minutes pour les tokens d'accès et d'un jour pour les tokens de rafraîchissement, j'ai parfois trouvé que le délai était assez court, surtout lors de la réalisation de tests avec différents profils ou lors de l’ecriture de code des fonctionnalités des API. J'ai également du etre attentive à chque fois de coller correctement le token dans l'en-tête de chaque requête, car la moindre erreur, y compris un espace supplémentaire, pouvait entraîner un échec de l'authentification. Dans certains cas, j'ai même dû vérifier les tokens via l'endpoint api/token ou api/token/refresh de mon application Django ou en utilisant des commandes curl dans le terminal.

Diagrammes et le Modèle Conceptuel de Données (MCD) : En ce qui concerne les diagrammes et le Modèle Conceptuel de Données (MCD), vous pouvez les trouver dans le dépôt Git du projet, car leur taille ne permet pas de les inclure directement dans un document Word.

Transaction SQL.

Pour demontrer la transaction SQL je peux me baser sur le views.oy de l’application eshop.

class OrderUpdateConsultantView(LoginRequiredMixin, UpdateView):  
 model = Order  
 fields = ['ordered', 'commentaire']  
 template\_name = 'eshop/order\_update\_consultant.html'  
 success\_url = reverse\_lazy('home')  
  
 # On utilise une transaction atomique pour garantir la cohérence des données  
 @transaction.atomic  
 def form\_valid(self, form):  
 # Récupération de l'objet Order à mettre à jour  
 order = self.get\_object()  
  
 # Mise à jour de l'objet Order et sauvegarde en base de données  
 response = super().form\_valid(form)  
  
 # Création du commentaire  
 Comment.objects.create(  
 order=order,  
 consultant=self.request.user.consultant,  
 body=form.cleaned\_data['commentaire']  
 )  
  
 # Message de confirmation  
 messages.success(self.request, "La commande a été mise à jour avec succès.")  
 return response

et la transaction SQL (le ficher se trouve dans git)

BEGIN;  
  
UPDATE eshop\_order  
SET user\_id = 63,  
 product\_id = 17,  
 quantity = 1000,  
 ordered = 1,  
 ordered\_date = '2023-05-10 18:33:23.046468',  
 commentaire = 'Commentaire lors d''une transaction'  
WHERE id = 49;  
  
COMMIT;

L'objectif de la transaction est de mettre à jour les informations d'une commande existante. Dans ce cas, la commande avec l'ID 49 a été sélectionnée pour être mise à jour.

Les colonnes mises à jour dans cette transaction sont les suivantes :

user\_id : L'identifiant de l'utilisateur associé à la commande est mis à jour avec la valeur 63.

product\_id : L'identifiant du produit associé à la commande est mis à jour avec la valeur 17.

quantity : La quantité de la commande est mise à jour avec la valeur 1000.

ordered : Le statut de la commande est mis à jour à 1, indiquant qu'elle a été passée.

ordered\_date : La date de la commande est mise à jour avec la valeur '2023-05-10 18:33:23.046468'.

commentaire : Le commentaire associé à la commande est mis à jour avec la valeur 'Commentaire lors d'une transaction'.

Cette transaction garantit l'intégrité des données en exécutant toutes les mises à jour de manière atomique. Cela signifie que toutes les modifications seront apportées avec succès ou aucune modification ne sera effectuée si une erreur se produit.

En résumé, le but de cette transaction est de mettre à jour les détails d'une commande spécifique, tels que le commentaire et le statut de la commande. Conformément aux besoins du cahier des charges, seuls les champs de commentaire et de statut sont modifiés, car c'est ce qui est demandé par le client. Les autres informations de la commande, telles que l'utilisateur associé, le produit et la quantité, restent inchangées lors de cette transaction car le consultant doit se limiter seulement aux chanemements de status de la commande et ajouts de ses commentaires.