Rapport individuel SAE 3.01c

Ilane Riotte

Synthèse :

Le projet MobiList a pour objectif de permettre le recensement numérique des biens mobiliers pour faciliter les démarches d'assurance en cas de sinistre. L'application distingue deux rôles principaux : l'assuré (qui gère ses biens) et l'assureur (qui gère ses clients). Pour cette dernière semaine de développement, je me suis concentré sur la fiabilisation de l'application et l'amélioration des droits de l'assureur. L'objectif était de sécuriser la saisie des données via des contraintes strictes et de permettre à un assureur d'effectuer n'importe quelle action (ajout, modification, suppression) pour le compte de son client, comme lors d'une assistance téléphonique.

Analyse :

Mon travail cette semaine s'est passé autour de deux grands axes :
Le renforcement de l'intégrité des données (Typage et Contraintes), j'ai retravaillé le fichier forms.py pour intégrer des contraintes de validation strictes sur tous les formulaires. Jusqu'ici, certaines saisies pouvaient provoquer des erreurs.

- J'ai ajouté des validateurs pour vérifier les types de données (dates cohérentes, prix positifs, chaînes de caractères nettoyées).
- Cela garantit que la base de données ne reçoit que des informations valides et évite les plantages inattendus lors de la soumission des formulaires.

Gestion complète par l'Assureur : J'ai modifié l'architecture des vues (views.py) et des templates HTML pour permettre à l'assureur d'intervenir directement sur le dossier d'un client.

- Adaptation des routes : J'ai modifié les fonctions d'ajout, de modification et de suppression pour qu'elles détectent si l'utilisateur est un assureur. Si c'est le cas, l'application identifie le client ciblé (via les relations Logement/Pièce) et redirige l'assureur vers la liste du client, et non vers son propre tableau de bord.
- Interface adaptative : J'ai mis à jour les templates (mes_logements.html, logement_pieces.html, etc.) pour afficher les boutons d'action (Ajouter/Modifier/Supprimer) même pour l'assureur, tout en adaptant les liens de retour (boutons "Annuler" intelligents) pour garantir une navigation fluide.

Démonstration de compétences :

AC21.01 | Élaborer et implémenter les spécifications :

- Implémentation de la fonctionnalité permettant à l'assureur de gérer les biens d'un client (scénario d'assistance téléphonique).
- Traduction des règles de gestion (contraintes de saisie) dans le code.

AC21.02 | Appliquer des principes d'ergonomie :

- Mise en place de redirections contextuelles : quand un assureur modifie un bien, il est redirigé vers le dossier du client et non le sien.

- Affichage conditionnel des boutons d'action dans les templates selon le rôle (Assuré ou Assureur).

AC21.03 | Bonnes pratiques de conception :

- Réutilisation des mêmes formulaires (forms.py) et des mêmes vues (views.py) pour les deux types d'utilisateurs, en ajoutant des conditions logiques plutôt que de dupliquer le code.

AC23.01 | Concevoir et développer des applications communicantes :

- Gestion des requêtes POST et GET complexes dans views.py pour transmettre les identifiants (ID pièce, ID logement) entre les pages lors des actions de l'assureur.

AC24.04 | Manipuler des données hétérogènes :

- Mise en place de contraintes de typage fort dans les formulaires Flask-WTF (Date, Float, String) pour assurer la cohérence des données avant l'envoi en base de données.

AC25.02 | Formaliser les besoins :

- Prise en compte du besoin métier : l'assureur doit avoir la main sur les données de l'assuré pour l'aider, sans pour autant se connecter à sa place.

AC26.03 | Compétences interpersonnelles :

- Coordination avec l'équipe pour fusionner la logique de gestion des droits (Assureur) avec les autres fonctionnalités (génération PDF, optimisation images) sans créer de régressions.