VIGNON CHAUDEY
Clément

Rapport individuel SAE 3.01c

Synthèse :

Le but de notre projet MobiList est de répertorier l'inventaire des objets mobiliers couverts par une assurance dans un domicile. Cet inventaire est essentiel pour la déclaration de sinistre auprès de l'assurance en cas de catastrophe naturelle entraînant des dégâts matériels. Le but est d'automatiser la mise à jour de l'inventaire lors d'une nouvelle acquisition et de conserver électroniquement les justificatifs d'achat, les factures et les photos. L'objectif est d'évaluer un état financier des pertes potentielles, pièce par pièce, en appliquant une vétusté basée sur le prix d'achat et la date du sinistre.
Durant cette 4ème semaine nous nous avons finalisé le code du site et nous avons passé la BD en PostgreeSQL en utilisant Docker.

Analyse :

Pendant la 4ème semaine je me suis notamment occupé de la partie éco-conception du site. Pour se faire, j’ai réfléchi à comment réduire l’empreinte environnementale de notre code. Ainsi, j’ai réussi à identifier 3 points majeurs à optimiser :

- Mieux gérer le stockage des images : Le stockage de factures en haute résolution prend beaucoup de place. J'ai implémenté un système de redimensionnement automatique via la bibliothèque Pillow dans views.py. Désormais, chaque image uploadée est traitée par la méthode img.thumbnail((800, 800)). Cela permet de réduire drastiquement le poids des fichiers (passant de plusieurs Mo à quelques Ko) sans perte de lisibilité pour l'assureur, économisant ainsi de l'espace disque sur le serveur et de la data lors de la consultation.
- Optimisation JavaScript : J'ai identifié que le script de recherche searchTable.js déclenchait des calculs à chaque caractère saisi, sollicitant inutilement le processeur de l'utilisateur. J'ai optimisé ce processus en intégrant un délai, afin que la fonction de filtrage ne s'exécute qu'après une courte pause, de 200ms.
- Refactoring du code : Pour rendre le code plus léger, j'ai factorisé les fonctions de validation dans forms.py. Au lieu de dupliquer les contrôles pour chaque champ, j'ai créé des fonctions réutilisables comme check_name_format ou check_simple_text.

Voici le code du fichier searchTable.js avant la mise en place d’un délai :

```
function setupTableSearch(inputId, tableId) {
    const input = document.getElementById(inputId);
    const table = document.getElementById(tableId);

    if (!input || !table) return;

    input.addEventListener('keyup', function () {
        const filter = input.value.toLowerCase();
        const rows = table.querySelectorAll('tbody tr');

        rows.forEach(row => {
            const text = row.textContent.toLowerCase();
            row.style.display = text.includes(filter) ? '' : 'none';
        });
    });
}
```

Voici le code après :où on peut voir l’ajout d’une variable timeout pour implémenter le délai.

```
function setupTableSearch(inputId, tableId) {
    const input = document.getElementById(inputId);
    const table = document.getElementById(tableId);

    if (!input || !table) return;

    let timeout = null;

    input.addEventListener('keyup', function () {
        clearTimeout(timeout);
        timeout = setTimeout(() => {
            const filter = input.value.toLowerCase();
            const rows = table.querySelectorAll('tbody tr');

            rows.forEach(row => {
                const text = row.textContent.toLowerCase();
                row.style.display = text.includes(filter) ? '' : 'none';
            });
        }, 200); // On attend 200ms de pause avant de filtrer
    });
}
```

Démonstration de compétences :

AC21.01 | Élaborer et implémenter les spécifications fonctionnelles et non fonctionnelles à partir des exigences

• Analyse des besoins : gestion des logements, pièces, biens, utilisateurs, sinistres.
• Implémentation des fonctionnalités selon les exigences du sujet.
• Prise en compte des contraintes : sécurité, validation des formulaires, gestion des erreurs.
• Ajout de contraintes de performance énergétique via le redimensionnement d'images et l'optimisation SQL.

AC21.02 | Appliquer des principes d’accessibilité et d’ergonomie
• Utilisation de labels explicites pour chaque champ de formulaire.
• Boutons d’action clairs et accessibles.
• Messages d’erreur et de succès affichés en couleur et accessibles.
• Navigation cohérente et structurée dans toutes les pages.

AC21.03 | Adopter de bonnes pratiques de conception et de programmation
• Respect des conventions PEP8 pour le code Python.
• Organisation du projet en modules : views.py, forms.py, database/, templates/, static/.
• Utilisation de l’ORM SQLAlchemy pour la gestion des données.
• Validation des données côté serveur avec WTForms.

AC22.01 | Choisir des structures de données complexes adaptées au problème

• Utilisation de modèles relationnels pour représenter les liens entre utilisateurs, logements, pièces, biens, sinistres.
• Relations SQLAlchemy : 1-1, 1-N, N-N.
• Utilisation de listes et dictionnaires pour transmettre les données aux templates.

AC23.01 | Concevoir et développer des applications communicantes

• Application web communicante via le protocole HTTP.
• Utilisation de Flask pour gérer les routes et les échanges entre client et serveur.
• Gestion des formulaires et des fichiers.

AC23.02 | Utiliser des serveurs et des services réseaux virtualisés

• Déploiement et exécution de l’application sur un serveur local Linux.
• Utilisation de Flask comme serveur applicatif.

AC24.03 | Organiser la restitution de données à travers la programmation et la visualisation

• Affichage des statistiques dans le tableau de bord.
• Utilisation de tableaux et de cartes pour visualiser les données.
• Restitution claire et synthétique des informations dans les templates HTML.

AC24.04 | Manipuler des données hétérogènes

• Gestion de différents types de données : chaînes, dates, fichiers, nombres.
• Manipulation de données issues de plusieurs modèles .
• Conversion et validation des formats.
• Manipulation de fichiers binaires (images) avec la bibliothèque Pillow pour conversion et optimisation.

AC25.02 | Formaliser les besoins du client et de l’utilisateur

• Rédaction des spécifications fonctionnelles.
• Traduction des besoins en fonctionnalités concrètes dans l’application.

AC25.03 | Identifier les critères de faisabilité d’un projet informatique

• Analyse de la faisabilité technique : choix de Flask, SQLAlchemy, WTForms.
• Prise en compte des contraintes de sécurité, d’ergonomie et de performance.
• Analyse de l'impact de l'infrastructure Docker/PostgreSQL sur la performance globale.

AC25.04 | Définir et mettre en œuvre une démarche de suivi de projet

• Organisation du code en modules et fichiers pour faciliter le suivi.
• Utilisation de messages flash et de logs pour le suivi des actions utilisateur.
• Tests unitaires présents dans le dossier tests/.

AC26.02 | Appliquer une démarche pour intégrer une équipe informatique au sein d’une organisation

• Respect des conventions de développement pour faciliter le travail en équipe.
• Documentation du code et des fonctionnalités.
• Utilisation de Git.

AC26.03 | Mobiliser les compétences interpersonnelles pour travailler dans une équipe informatique

• Communication des choix techniques et des solutions aux membres de l’équipe.
• Collaboration sur la conception et la validation des fonctionnalités.

AC26.04 | Rendre compte de son activité professionnelle

• Rédaction de rapports et de documentation sur l’avancement du projet.
• Présentation des fonctionnalités réalisées et des difficultés rencontrées.
• Utilisation de messages flash et de logs pour tracer les actions.