PCBuilder App

Rapport final du projet

Cours : SEG 2505 - Introduction au génie logiciel
Professeur : Laurent FREREBEAU
Session : Automne 2024

Équipe de projet : ***39***

Nom complet	Numéro étudiant	Rôle
Ghali Kinany Alaoui	300344809	Maintainer
Ayouch Salma	300347136	Participant
Bennasser Soukaina	300360203	Participant

Lien du dépôt GitHub:

https://github.com/uOttawa-2024-2025-seg2505-projet/groupe-39.git

2. Description des choix de conception

Hiérarchie des classes:

Chaque rôle (Administrateur, Demandeur, Assembleur, Magasinier) est défini dans une classe propre à lui, et tous ces derniers héritent de la classe abstraite User qui contient les attributs communs à tous. Cette généralisation permet de factoriser du code et de le centraliser en une classe ce qui rend le système plus flexible et maintenable en cas de modifications nécessaires.

Hiérarchie des interfaces:

Chaque rôle (Administrateur, Demandeur, Assembleur, Magasinier) est encapsulé dans une interface qui lui correspond:
- IManageRequests pour la gestion des commandes des demandeurs.
-IManageStock pour la gestion du stock des magasiniers.
-IAdministrator pour la gestion des utilisateurs et des données système par les administrateurs.
-IAssembler pour la gestion des commandes en attente par les assembleurs.
Cela permet de créer un contrat qui force toutes les instances de ces classes d'assurer les rôles qui leur sont assignés
De plus, toutes ces interfaces héritent IUser qui gère l'authentification des rôles.

Gestion des connexions :

Le système utilise une classe UserManager qui, à l'aide d'une combinaison clé/valeur (email,mot de passe), permet l'authentification et l'autorisation ou non des connexions même après l'ajout, la suppression, ou la modification de certains acteurs lors de l'exécution de l'application.

Gestion des données:

Une base données SQLite est utilisée pour stocker les identifiants par défaut, les demandeurs, le stock, ainsi que les commandes à tout moment lors de l'exécution de l'application. SQLite a été préféré par rapport à firebase pour son fonctionnement local et direct, contrairement à firebase qui est hébergé dans un cloud, bien que SQLite nécessite l'écriture de beaucoup plus de code.
La recherche des données par colonnes permet de faciliter la gestions des permissions et des priorités (Ex: Admin peut voir toutes les commandes, mais Requester peut voir que celles qui lui sont propres)

Le format .csv a été choisi pour sa simplicité de traitement de données pour initialiser les utilisateurs par défaut, et une classe CSVImporter permet de traiter les données provenant de fichier prédéfinis ou externes et de les charger au lancement de l'application la rendant plus facile à utiliser.

Gestion des commandes:
Les Requester peuvent initier des commandes selon les contraintes imposées par le client, et le système de commande est étroitement lié au stock qui permet de déterminer si une commande est réalisable ou pas, selon la disponibilité des composants. Ces commandes sont gérées par leur statut qui est initialement “en attente d’acceptation”, et sur lequel l’assembleur a le contrôle.

3. Description des exigences supplémentaires proposées, traitées ou écartées

- Les Requester doivent pouvoir évaluer les Assembleur à l’issu de leur commande: proposée mais non traitée

- Les noms et prénoms doivent être optionnels lors de la création d’utilisateur: pour tous rôles, seuls les email et mot de passe uniques suffisent pour identifier un utilisateur. (traitée)

- la réinitialisation de la base de données doit se faire directement sur l’interface utilisateur: Il faut éviter de devoir parcourir les fichiers stockés dans l’appareil android afin de réinitialiser la base donnée, améliorant l’expérience utilisateur (traitée)

- Il doit y avoir au moins 2 options possibles pour chaque composant: Un inventaire diversifié est important pour servir les différents besoins de tous les Requester (traitée)

- Les commandes liés à un Requester doivent être supprimées en même temps que ce dernier (traitée)

- L’assembleur ne peut refuser une demande valide: L’assembleur doit assurer son rôle, sauf s’il a une raison valide (manque de stock) (traitée)

- Les commandes refusées contenant des composants qui viennent d’être remis en stock doivent automatiquement être validés: proposée mais non traitée

- Une commande doit pouvoir être retournée pour une raison spécifique: une commande est retournée à cause de matériel défectueux, nécessitant l’introduction d’un nouvel état “Retourné” (proposée mais non traitée)

4. Retour d'expérience :

organisation en équipe du groupe:
L'organisation du groupe s'est faites pour chaque livrable selon les étapes suivantes:

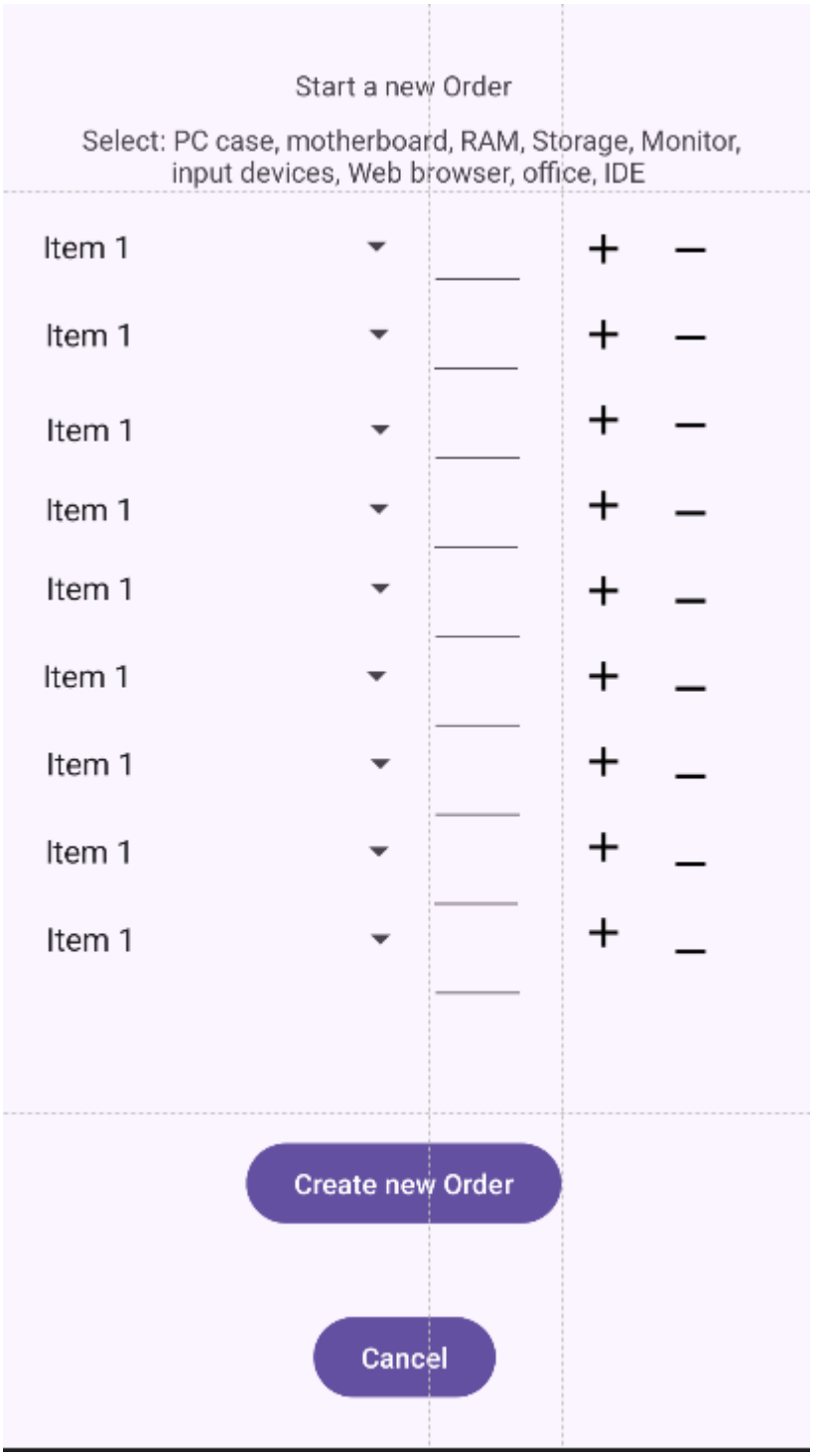
- Lecture en groupe des exigences et d'idées pour les implémenter.
- Lecture du livrable suivant pour délimiter les exigences demandées (il se peut que des exigences soit demandés plus tard, et doivent donc être implémentés au bon moment)
- Création de l'interface utilisateur pour le rôle concerné et ses fonctionnalités
- implémentation de la logique des fonctionnalités
- tests avec des exemples
- modélisation UML des modifications ajoutées
- Réalisation du PDF de démonstration
- pré-évaluation du livrable faîtes par le Maintainer
- ajout des artifacts et autres (APK…)
- Soumission du livrable

Chaque personne s'occupe d'une ou plusieurs étapes du processus décrit et des réunions sont organisées environ toutes les semaines, ou si un problème majeur est rencontré, en virtuel pour s'assurer du bon fonctionnement du projet.

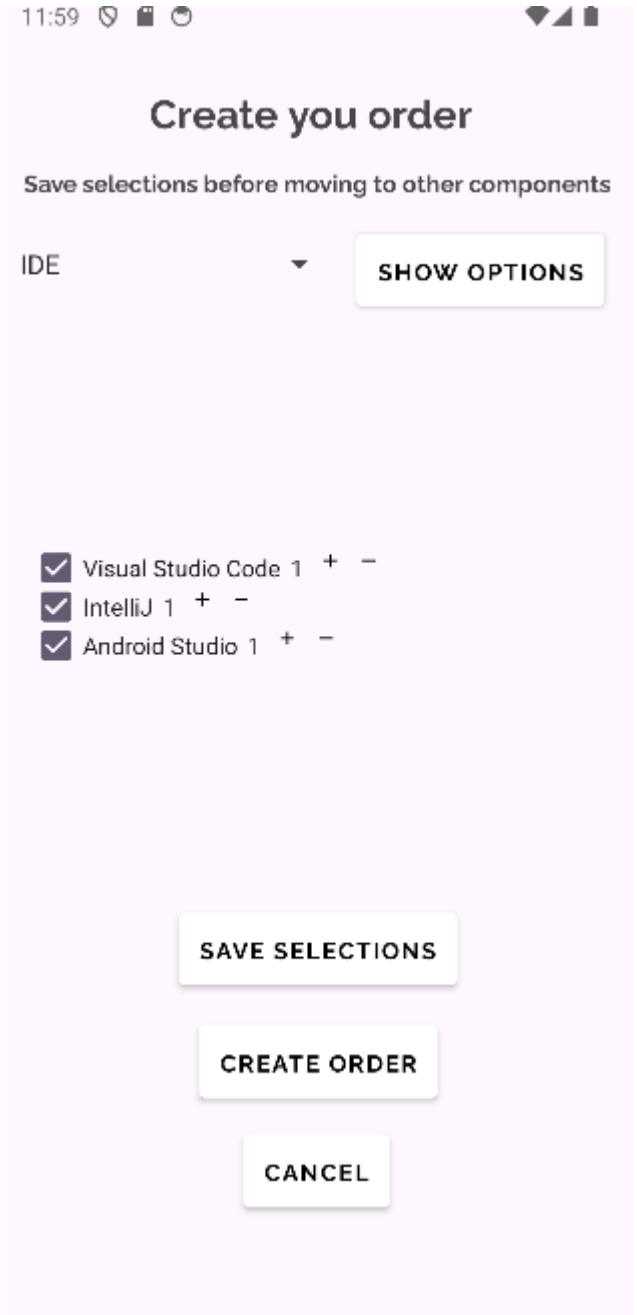
Nous avons rencontré des difficultés majeurs lors du premier livrable lié à l'installation d'Android Studio; mais aussi à GitHub qui nécessitait la mise en place de clé SSH et l'utilisation de cette dernière pour pouvoir contribuer. Une membre du groupe a notamment dû acheter un nouveau pc pour un développement plus fluide dans l'environnement Android, qui devenait de plus en plus lourd à charger.

Le respect des échéances constituait un grand frein puisque le travail demandé est séquentiel, et que certains ne pouvaient avancer sans que d'autres aient terminé leur part de travail. Cela était le cas lors des 2 premiers livrables, mais cela a été résolu par la suite, en s'y prenant à l'avance.

La lecture et la compréhension des exigences étaient très importantes pour le fonctionnement du projet. En effet, lors du livrable 3, pour l'implémentation du système de commande, une mauvaise compréhension des exigences a mené à l'interface utilisateur suivante:



Cette dernière utilise des Spinner qui permettent une sélection unique, ce qui ne répond pas aux choix multiples possibles imposés par les exigences. Avec le début d'implémentation de la logique, cela a coûté au groupe environ 5 heures de travail. Après s'être rencontré, le groupe est parvenu à une solution avec l'interface suivante :



Suite à un brainstorm avec tous les membres de l'équipe, nous sommes arrivés aux besoins suivants pour de potentiels futurs projet:

- L'utilisation de GitHub pour le travail collaboratif a permis de travailler de manière fluide notamment avec le mécanisme de branches.
- Il faudrait plus commenter le code afin de pouvoir comprendre le travail effectué, et mieux communiquer sur ce qu'il a été réalisé, et ce qu'il reste à réaliser
- Il faudrait faire des réunions plus souvent pour coordonner le travail et ne pas avancer dans des directions opposés pendant trop longtemps.

6. tableau de synthèse des contributions

Ayouch Salma 300347136	Bennasser Soukaina 300360203	Kinany Alaoui Ghali 300344809
Implémenter le rôle "Administrator" pour : ● Ajouter, modifier et supprimer des utilisateurs "Requester". ● Assurer que les informations utilisateurs incluent : email (unique), mot de passe ● Tester la fonctionnalité d'authentification et de gestion des utilisateurs. Mettre à jour le fichier README Réaliser des tests unitaires pour vérifier les transitions d'état des commandes. Mettre à jour le modèle de données pour inclure une relation entre utilisateurs et commandes. Concevoir l'interface utilisateur pour le rôle "Assembler" : ● Une liste des commandes en attente. ● Un écran détaillé pour chaque commande, permettant de valider ou rejeter la	Concevoir l'interface utilisateur pour le rôle "StoreKeeper", incluant une liste tabulaire des composants. Implémenter les fonctionnalités suivantes : ● **Ajout d'un composant** : matériel ou logiciel, avec des attributs tels que type, sous-type, titre, quantité et commentaires. ● **Modification d'un composant** : mise à jour des quantités et des détails. ● **Suppression d'un composant** : retirer un élément du stock. ● **Visualisation du stock** : afficher une liste de tous les composants existants. Mettre en place un cycle de vie des commandes avec des états : ● En attente d'acceptation. ● Acceptée, en cours d'assemblage. ● Rejetée. ● Livrée.	Ajouter une base de données SQLite pour stocker temporairement les utilisateurs en mémoire. Mettre en place des validations pour les champs obligatoires de StoreKeeper (titre unique, quantités numériques positives, etc.) Tester les fonctionnalités liées au rôle StoreKeeper pour s'assurer qu'elles fonctionnent correctement avec SQLite. Concevoir l'interface utilisateur pour le rôle "Requester" : ● Une liste déroulante pour sélectionner les composants matériels (PC Case, RAM, Storage, etc.). ● Des cases à cocher ou des listes multiples pour sélectionner les composants logiciels. ● Un bouton pour valider et envoyer la commande. Implémenter les fonctionnalités suivantes : ● **Validation d'une**

commande. Tester tous les scénarios possibles, y compris : ● Création d'une commande complète par un Requester. ● Modification et visualisation du stock par le StoreKeeper. ● Validation, rejet et clôture des commandes par l'Assembler. Conception des diagrammes classe, état	Tester toutes les transitions d'état des commandes et leur interaction avec le stock. Ajouter des messages d'erreur appropriés en cas de rupture de stock ou de données invalides. Conception des diagrammes séquence, activité	**commande** : Vérifier si le stock contient suffisamment de composants pour assembler la commande. Passer la commande à l'état "Acceptée, en cours d'assemblage". ● **Rejet d'une commande** : Ajouter un commentaire expliquant la raison du rejet. Mettre la commande à l'état "Rejetée". ● **Clôture d'une commande** : Consommer les composants nécessaires du stock. Mettre la commande à l'état "Livrée".

7.Estimation du temps passé sur chaque livrable:

livrable 1: 35h
livrable 2: 40h
livrable 3: 45h
livrable 4: 30h
livrable 5: 10h