Rapport Final | Groupe 11

https://github.com/uOttawa-2024-2025-seg2505-projet/groupe-11.git

Choix de conception:

Pour le nom de notre produit, nous avons décidé de choisir un nom mémorable, et **BuildItTech** capture parfaitement notre objectif: permettre aux utilisateurs de construire facilement leur propre PC. Le nom communique clairement l'objectif principal de l'application en combinant une action, « Build-It » ou « Construisez-le », et une référence explicite à la technologie, « Tech ». Ce choix reflète l'idée DIY, « Do It Yourself », de l'assemblage de PC tout en soulignant la base technologique de notre service. Nous avons décidé de créer une application conviviale et visuellement attrayante. Pour nous, ceci est important pour le succès de notre application de commande de PC sur mesure. Durant la construction de l'application, notre priorité a été de miser sur la simplicité et la fonctionnalité, garantissant ainsi une expérience fluide pour les utilisateurs qui souhaitent parcourir, personnaliser et acheter leur PC idéal. Voici les choix de conception qui définissent **BuildItTech** et les éléments qui la rendent unique.

Pour commencer, la simplicité est la pièce fondamentale du design de notre application. Naviguer dans l'application doit être intuitif, même pour les utilisateurs moins familiers avec la technologie. Pour cela, nous avons opté pour une interface bien organisée, avec des sections bien définies. La page de **Requester** est conçue pour minimiser les distractions, permettant ainsi aux utilisateurs de se concentrer sur l'essentiel: choisir et personnaliser les composants de leur PC. L'utilisation de polices lisibles et d'un contenu organisé de manière logique garantit une expérience utilisateur simple. Pour les pages **Administrator**, **StoreKeeper**, et **Assembler**, nous avons gardé la même idée, simple et claire. Les champs de texte pour créer des utilisateurs **Requester** sur la page **Administrator** sont organisés et étiquetés pour une compréhension facile de ce qui est nécessaire pour certaines fonctions. Similairement, pour la page **Assembler**, les champs de textes sont aussi étiquetés pour la facilité d'utilisation. Aussi, il se trouve des vues défilantes pour voire des commandes en attentes, acceptés, et livrés pour que l'utilisateur avec le rôle d'**Assembler** peut vérifier les commandes d'une façon organisée. Pour le dernier rôle, **StoreKeeper**, les mêmes approches ont été adoptées pour visualiser, ajouter et supprimer des composantes de PC sur mesure dans le stock. Notre application simplifie le processus de commande qui est un élément clé du parcours utilisateur. Le processus comprend un nombre minimal d'étapes et de choix clairs et simples pour l'utilisateur. Des indicateurs d'erreur guident les utilisateurs, réduisant ainsi les abandons de panier.

En conclusion, chaque décision de conception pour **BuildItTech** s'articule autour de la création d'une expérience utilisateur simple, efficace et agréable. Du design et le nom accrocheur aux fonctionnalités conviviales, l'application représente notre objectif de rendre l'assemblage de PC sur mesure accessible à tous.

Exigences supplémentaires:

- Classes:

Pour améliorer la fonctionnalité et l'organisation de notre projet, nous avons incorporé quatre classes essentielles : **Component**, **DataBaseInitializer**, **DataBaseHelper**, et **StockManager**. La classe Component est spécialement conçue pour créer et modifier des composants informatiques, ce qui nous permet de gérer plus efficacement des informations détaillées sur des éléments individuels. La classe **DataBaseInitializer** est responsable de l'initialisation des données à partir d'un fichier externe, ce qui inclut la mise en place d'enregistrements initiaux et la création de profils d'utilisateurs. Elle garantit que notre base de données est pré-remplie avec les données nécessaires, ce qui permet de gagner du temps lors du développement. La classe **DataBaseHelper** est essentielle dans la gestion de la base de données SQLite pour notre application. Elle gère non seulement la création et l'initialisation de la base de données, mais facilite également diverses tâches telles que l'ajout, la mise à jour et la suppression d'utilisateurs et d'articles en stock. Elle fournit des mécanismes d'authentification pour la connexion des utilisateurs, gère les rôles et récupère les données des utilisateurs en cas de besoin. En outre, la classe gère l'inventaire grâce à des méthodes qui permettent d'ajouter, de modifier et de supprimer des articles en stock. Elle prend en charge le traitement des commandes en ajoutant de nouvelles commandes, en vérifiant la disponibilité du stock et en mettant à jour l'état des commandes (par exemple, en attente, accepté, expédié). Le **DataBaseHelper** comprend également des fonctionnalités permettant d'assembler et d'expédier les commandes tout en garantissant que les quantités en stock sont correctement mises à jour. Ces fonctionnalités garantissent une interaction transparente entre l'application et la base de données, ce qui permet de rationaliser les opérations critiques telles que la gestion des utilisateurs, le contrôle des stocks et le traitement des commandes. Enfin, la classe **StockManager** se concentre sur la création, la modification et la suppression des éléments de stock. En traitant ces opérations dans un fichier séparé, elle améliore l'organisation et la maintenabilité du code de la classe StoreKeeper. Cette approche améliore la clarté et simplifie le débogage tout en garantissant que chaque classe a un rôle bien défini dans l'application.

Retour d’expérience:

- **Organisation d’équipe:**

Puisque nous sommes deux dans le groupe, l’organisation et l’allocation des tâches étaient plutôt faciles. Grâce à une communication claire et une bonne compréhension des tâches demandées, nous avons pu diviser les responsabilités en fonction de nos points forts individuels. Cette approche simple nous a permis non seulement de rester sur la bonne voie, mais aussi d’avoir une bonne compréhension de ce que l’autre a fait puisqu’il était très facile de communiquer et de discuter nos idées. La collaboration s'est bien déroulée et l'efficacité de notre travail d'équipe a contribué de manière significative à la réussite globale du projet.

- **Difficultées:**

Bien que notre projet de codage ait été réussi, par contre, nous avons rencontré plusieurs difficultés durant chaque livrable. Au départ, nous avons eu du mal à placer correctement notre projet sur GitHub, ce qui a nui à notre capacité à collaborer efficacement. Ce problème a été résolu après une réunion avec le professeur, qui nous a guidés tout au long du processus. Notre tentative de création d'une base de données à l'aide de Firebase a constitué un autre obstacle important. Les difficultés rencontrées pour la comprendre et la mettre en œuvre ont entraîné des retards et la perte de points sérieux. Par conséquent, nous avons décidé de passer à SQLite, qui s'est montré être une solution plus facile à gérer pour nos besoins. Nous avons également rencontré des problèmes de compréhension de certaines exigences du projet, qui ont nécessité des éclaircissements supplémentaires de la part de notre professeur. Le débogage a été un autre obstacle, car nous avons rencontré des erreurs complexes que nous n'avons pu résoudre qu'en demandant de l'aide à nos collègues, à des ressources en ligne et à au professeur. La conception de l'interface utilisateur pour chaque page a présenté sa propre série de défis, mais grâce à la régularité, aux essais et aux erreurs, à la pratique et aux tests, nous sommes arrivés à créer un design fonctionnel et cohérent. Ces obstacles, bien que frustrants, ont finalement renforcé nos compétences en matière de résolution de problèmes et de collaboration. Nous avons également connu beaucoup de difficultés pour lancer l’application en raison de la faible capacité de mon ordinateur, qui m'empêchait la plupart du temps même d'être capable de tester l'application.

- **Leçons tirées pour les futurs projets:**

Dans notre projet, plusieurs aspects se sont bien déroulés et ont fourni de bonnes informations pour les projets futurs, bien qu'il y ait également eu des défis qui ont mis en évidence des domaines à améliorer. Notre capacité à collaborer efficacement en équipe de deux a été un atout majeur, car nous avons divisé les tâches clairement et maintenu une communication cohérente, ce qui nous a permis de travailler efficacement. La structure modulaire de notre code, avec des classes distinctes a non seulement amélioré l'organisation mais a également rendu le débogage et les modifications futures plus faciles à gérer. Cependant, le projet n'a pas été sans difficultés. Les difficultés initiales liées à la mise en place du projet sur GitHub ont retardé les progrès jusqu'à ce que nous demandions de l'aide au professeur, soulignant l'importance d'une meilleure préparation et d'une meilleure pratique des systèmes de contrôle de version dans les

projets futurs. Le choix de Firebase pour la base de données s'est également montré problématique, car nous avons rencontré des difficultés qui ont entraîné des retards et des pertes de points; finalement, le passage à SQLite s'est révélé comme une solution plus simple et plus efficace. Les incompréhensions concernant les exigences du projet ont entraîné des inefficacités, soulignant la nécessité d'une communication plus claire et d'une clarification plus tôt avec le professeur. Les problèmes de débogage étaient fréquents et, bien qu'ils aient finalement été résolus grâce à une aide extérieure et à la persévérance, l'adoption de pratiques de débogage systématiques, telles que les tests unitaires et la documentation détaillée, pourrait permettre d'éviter de tels retards à l'avenir. En plus, la création d'une interface utilisateur cohérente pour l'application a nécessité beaucoup d'essais et d'erreurs, ce qui suggère que l'utilisation de prototypes au stade de la conception pourrait simplifier le processus dans les projets à venir. En nous appuyant sur notre travail d'équipe efficace et nos pratiques de codage modulaire, tout en relevant ces défis par une meilleure planification, évaluation de la technologie et communication, nous pourrons approcher les projets futurs avec plus de confiance et de succès.

Tableau de synthèse:

Olivier Storey 300360991:

- Organiser toutes les demandes du projet dans un document pour chaque livrable pour bien organiser les tâches correspondant à nos forces pendant toute la durée du projet.
- Créer le nom et le logo de l'application **BuildItTech**
- Planifier, modéliser, créer et modifier toutes les interfaces utilisateur pour chaqu'une des pages de l'application (**WelcomePage**, **Administrator**, **StoreKeeper**, **Requester** et **Assembler**)
- Planifier et créer des boutons, champs de texte, menus déroulants, champs de visualisation des composantes de PC dans le stock et des commandes de PC sur mesure, etc.
- Communiquer avec Melvyn sur la façon dont certaines choses doivent fonctionner afin qu'il puisse ajouter leurs fonctionnalités. (Bouton Login/Logout, Bouton pour ajouter au stock, etc.
- Créer une classe de **StockManager** pour faciliter les fonctions de **StoreKeeper**
- Ajouter la fonctionnalité pour la page **StoreKeeper**
- Manipulation des données afin de les visualiser et de les manipuler davantage, si nécessaire, sur la page **StoreKeeper**
- Creation et modification du ficher README.md
- Modification du ficher UML.md et creation des diagrammes d'état, de séquence, et d'activité
- Testing du projet à chaque livrable

Melvyn Avoa 300357232

- Testing du projet à chaque livrable
- Création de la base de donnée SQlite et application des fonctionnalités de la base de donnée au système de l'application ainsi que des fichiers utiles à l'utilisation de SQLite tels que pc_managment.db et Databasehelper.java
- Gestion des connexions et des utilisateurs (Implémentation de la page de connexion)
- Ajouts des fonctionnalités à la page Requester
- Implémentation des fonctionnalités de Administrator
- Implémentation des fonctionnalités de Assembler
- Implémenter le fonctionnement des classes (**WelcomePage**, **Administrator**, **StoreKeeper**, **Requester** et **Assembler**)
- Génération des fichiers APK à chaque livrable
- Livraison sur brightspace
- Fixation de la communication des différentes communication des classes avec la base de donnée
- Création, ajout et implémentation de toutes les classes de test au complet (AdministratorTest, AssemblerTest, RequesterTest….)
- Ajout d'informations au Readme.

Estimation du temps global pour chaque livrable:

Nous avons consacré environ 15-20 heures au premier livrable, 18-22 heures au deuxième, 19-22 heures au troisième et 20-24 heures au quatrième.