BERTHEL Gabriel - LAGASSE Adrian - MSIAH Romain - REMY Léo

Conception et visualisation de tournées agricoles en circuits courts.

Table des matières

Progression du projet :

Implémentation du modèle et des DAO

En parallèle du rendu du Jalon 2, nous avons pensé judicieux de commencer l'implémentation du modèle et des DAO afin de gagner un maximum de temps et d'avoir une base solide pour la suite du projet. En effet, ceux-ci implémentés, la suite n'en sera que grandement facilitée grâce à l'avance prise sur la création des vues.
Au début, nous pensions que nous pourrions passer rapidement aux contrôleurs, mais nous devions d'abord passer par une étape importante : un test complet et exhaustif de la connexion à la base et des DAO.

La difficulté aura été de diffuser les données dans les objets déjà existants. En effet, notre objectif est de fournir une interface assez intuitive puisque la personne manipulant les objets-métier n'aura par exemple qu'à ajouter des éléments dans une liste puis invoquer la méthode update.

Les objets déjà existants référençant ce dernier seront ainsi mis-à-jour, tout comme les informations en base.

Si cela a un coût évident en mémoire, c'est néanmoins une solution élégante d'utilisation d'autant plus que nous avons passé une grande partie de notre temps à rédiger, penser et schématiser le projet. Nous connaissions donc déjà les contraintes.

Le risque avec cette solution, ce sont les boucles infinies. La solution est d'utiliser des méthodes plus spécifiques pour charger les différents tableaux, en imbriquant les appels intelligents.

Les user stories ont ainsi pris tout leur sens. Le modèle en lui-même aurait pu devenir une application en ligne de commande.

Réunions quotidiennes et travail collaboratif

Suite aux commentaires de chacun lors du dernier jalon, a été décidé la mise en place d'une réunion quotidienne durant laquelle tout le groupe travaillerait de concert. Nous avons banalisé les jours de fêtes (24 et 25 décembre & 31 décembre et 1er janvier), avec comme objectif clair de finaliser le logiciel le 30 décembre. Les réunions se tenaient à une heure symbolique de 15h01, et malgré certaines indisponibilités, elles ont été largement respectées et ont permis d'avancer rapidement chaque étape du projet.
Nous avons également mis en place des sprints courts de quelques jours avec une répartition claire des tâches et un backlog mis à jour en continu.
Le fait d'être tous présents nous a permis de demander de l'aide à chacun sur son domaine de prédilection, s'accorder sur la manière d'implémenter certaines fonctionnalités, et de s'adapter les uns aux autres.

Test des DAO

Après la mise en place de ces réunions, la priorité était la mise en place sûre de la connexion à la base de données. Les classes de DAO étant déjà écrites, il a fallu vérifier qu'elles avaient bien le comportement attendu, tout en prenant en compte certaines contraintes décrites lors du premier jalon (la plupart seront gérées plus tard par les contrôleurs).

Les tests sont réalisés sur une base de données à part nommée GDATest, permettant à terme de pouvoir vérifier le bon fonctionnement du logiciel sans avoir à compromettre les données de production.

Lors de ces tests, mis à part les erreurs d'étourderie ici et là, nous avons rencontré deux problèmes majeurs : la gestion de la suppression et le comportement attendu par la base, et un problème de récursivité infinie causée par le remplissage des List dans les objets, qui se faisaient appel les unes aux autres.

Après trois jours de travail, soit la durée prévue du sprint, les DAO ont pu être stabilisés et les tests tous au vert.

Classe validatrice de données et choix des API

Le projet tournant essentiellement autour de la validation de Tournées et l'affichage de cartes, c'est naturellement vers ces aspects que nous avons dû nous pencher pour ce que nous pensions être la partie critique du logiciel (et nous n'avions pas vraiment tort).

La première classe validatrice de données concernait les informations de base saisies par l'utilisateur. Son pseudonyme, son mot de passe, son nom, son numéro de téléphone, et le plus important : son adresse.

Mis à part l'adresse qui a nécessité un traitement particulier, les informations de base ont pu être vérifiées facilement à l'aide de Regex.

S'il est possible de garantir le bon format des adresses avec des Regex, cela n'en dit rien de leur validité. Or, c'est une information assez critique puisque les coordonnées GPS en sont dépendantes.

Ainsi, nous avons mis en place un validateur d'adresse qui s'auto-instancie à partir d'une méthode statique. On commence par formater les informations fournies au format CSV, puis l'on vérifie si cette chaîne existe dans un dictionnaire. Si c'est le cas, on renvoie l'objet représentant une adresse valide déjà existante.

Autrement, on le crée en faisant appel à l'API de géocodage fournie par le gouvernement francais: https://adresse.data.gouv.fr/csv

La limite est évidente, on ne peut que gérer des adresses en France.

Cependant, si on trouve une autre API, on pourrait facilement la remplacer sans rien changer ailleurs dans le projet.

Ainsi, c'est une limite tout à fait acceptable puisqu'elle n'impacte pas le fonctionnement du logiciel.

Concernant l'affichage des cartes, nous avons hésité entre plusieurs implémentations comme l'intégration d'une carte dans la vue grâce à Gluon, mais avons choisi l'utilisation d'une WebView. Ce choix a été motivé par la génération facile d'une URL Google Maps grâce aux coordonnées GPS. Ici, il n'est pas question de savoir si une Tournée est valide,

mais de l'afficher. En théorie, si elle est en base, cette vérification a déjà été effectuée. Nous avons étendu cette méthode à l'affichage des Commandes, qui ne représente qu'un point sur la carte et est donc très facile à implémenter.

Concernant la vérification de la validité des Tournées et le calcul des heures de départ et d'arrivée, nous avons opté pour l'API OpenRouteService d'OpenStreetMap, qui offre la plus grande liberté par rapport aux autres API OpenStreetMap, tout en restant gratuite avec un nombre d'utilisations journalières pertinent pour un projet comme le nôtre.

Pour valider une Tournée nous avons implémenté la méthode suivante. Le producteur sélectionne l'une après l'autre les commandes qu'il souhaite livrer. À chaque sélection, nous transmettons la liste des commandes dans l'ordre de leur sélection. Nous récupérons ensuite leurs coordonnées GPS et ajoutons au début et à la fin celles du lieu du producteur. Ainsi nous pouvons transmettre cette liste de coordonnées via l'API ORS sous format JSON. Nous recevons en retour le trajet et la durée associée aux différents points de passage.

À partir de là, nous pouvons calculer si le trajet est valide. Pour cela nous vérifions pour chaque étape que l'heure d'arrivée est inférieure à l'heure de fin de la commande associée, si ce n'est pas le cas on annule tout et le trajet n'est pas bon. Si l'heure d'arrivée est inférieure à l'heure de début de la commande, alors le producteur attend cette heure avant de continuer.

Nous pouvons noter ici une limite à ce système. Bien qu'étant fonctionnel en tout temps, il pourrait être optimisé de manière à ne calculer que le trajet associé à la commande qui vient d'être ajoutée. Cela n'a pas été fait car le problème n'a pas été envisagé au moment de la conception de la méthode.

Implémentation des contrôleurs

La partie la plus longue et qui nous a demandé le plus de travail après les DAO a été l'implémentation des contrôleurs, qui font le lien entre les vues et nos DAO, et pour lesquels il a fallu prendre en compte toutes les contraintes soulevées précédemment.

Pour les contrôleurs des vues contenant des formulaires (Ajout, Modification, Suppression), nous avons mis en place un système de validateurs de formulaires, qui fonctionnent tous selon le même principe et ont donc permis la systématisation de l'implémentation des contrôleurs. C'est ici que les contraintes sont vérifiées.

Cela permet de séparer la logique des contrôleurs au maximum, pour se concentrer sur la liaison entre vues et modèles.

Comme nous avons fait le choix d'utiliser des Timestamp dans la base de données, il a fallu créer un objet pour gérer la conversion des informations dans la vue vers un objet TimeStamp. Cet objet est aussi capable de formater un Timestamp pour ses différentes utilisations dans les vues. Cela permet de toujours manipuler des Timestamp au lieu de faire des conversions dans tous les sens.

Une des difficultés fut la mise à jour des listes sur les deux écrans de consultation. Une solution a été deux méthodes de fermeture de pop-up: une envoyant un événement vers son parent, et une se contentant de fermer la scène.

Ainsi à la capture de cet événement, les listViews sont rechargées.

Concernant les contrôleurs de connexion, Java n'offrant pas de méthode de hashage simple et robuste, nous avons décidé d'utiliser la bibliothèque argon2-jvm qui permet de hasher et vérifier la validité d'un mot de passe avec l'algorithme Argon2. Grâce à ce dernier, le mot de passe n'est ni stocké en clair dans la base de données, ni envoyé en clair au serveur, ce qui assure l'intégrité et la confidentialité de ce dernier.

De plus, afin de se conformer aux recommandations de l'ANSSI, nous avons implémenté dans le validateur de données une méthode liée aux mots de passe qui s'assure que lors

de sa création, ce dernier fait au minimum 9 caractères, qu'il contient au moins 1 caractère spécial, 1 majuscule, 1 minuscule et 1 chiffre. En outre, nous n'avons pas mis de limite à la taille des mot de passe pour permettre aux utilisateurs qui le désirent de générer un long mot de passe et de le conserver via un gestionnaire de mots de passe.

Au moment de la connexion du producteur, nous chargeons toutes les Tournées lui étant associées puis, à partir de ces Tournées, chargeons toutes les Commandes et tous les Véhicules associés. De ce fait, nous pouvions manipuler ces données sous forme d'objets en garantissant la propagation des modifications.

Ainsi pour la partie concernant l'ajout d'une Tournée, nous nous sommes servis des méthodes offertes par les DAO. Après avoir terminé la sélection des Commandes (cf. plus haut), nous créons une nouvelle Tournée avec toutes ces données puis appelons la méthode add afin de l'inscrire dans la base de données. Nous avons suivi une procédure similaire pour les autres objets.

Pour la modification d'une Tournée, un problème se posait. Si un Producteur commençait à modifier une Tournée puis finalement se rétractait, l'objet associé à la Tournée étant modifié, il ne correspondait plus à ce qu'il est censé représenter (notamment avec ses Commandes associées). Pour régler ce problème, nous avons mis en place un système de sauvegarde lorsque le Producteur commence à modifier une Tournée. La liste des commandes est stockée dans une copie indépendante de la liste associée à la Tournée. Si le Producteur décide d'annuler, nous supprimons toutes les Commandes dans la Tournée puis transformons la liste copiée en itérateur afin de remettre les anciennes Commandes dans la Tournée. De ce fait, nous nous débarrassons des Commandes ajoutées et remettons les anciennes.

Test exhaustif du logiciel

Une fois les contrôleurs implémentés, certains bugs restaient cachés. Il a donc fallu les dénicher. Ainsi dans les deux derniers jours, chaque contrainte a été testée et les bugs découverts puis corrigés.

Par souci de temps, nous n'avons pas fait de test unitaire sur les validateurs de formulaire. A la place, nous avons déroulé toutes les user stories lorsque nous avons testé le logiciel.

Ainsi ont été testés les validateurs de données, d'adresses et de Tournées. Les DAO et le connecteur sont aussi testés, ainsi que le générateur d'URL pour l'affichage, et l'objet DateManager.

Néanmoins, avoir séparé la validation des formulaires des contrôleurs permet de réaliser des tests unitaires. Cela fait partie des tâches qui restent potentiellement à réaliser, idéalement en reprenant toutes les user-stories en tests comme nous l'avons fait plus concrètement. Cela permettrait de systématiser ce processus.

L'un des bugs n'ayant pas pu être corrigé concerne la WebView et l'affichage de la carte, notamment pour les Tournées. Lors d'une utilisation trop violente de la carte, le logiciel freeze et lance une NullPointerException en boucle et devient inutilisable, même lorsque l'on ferme la vue qui affiche la carte. Il faut alors le fermer et le relancer, et être plus délicat avec la carte. Même si nous avons des idées, nous ne savons pas vraiment à quoi est dû ce bug et sommes dans l'impossibilité de le corriger.

Une alternative non satisfaisante serait d'importer le noyau chromium capable de gérer des pages plus modernes.Cependant, il faut passer un swing-node ou bidouiller javaFX au détriment des performances. Il n'y a pas encore de support officiel à ce jour.

Une autre solution plus élégante mais hors-sujet serait de continuer d'utiliser les WebViews, mais en hébergeant notre propre API de visualisation. En effet, il existe une API JavaScript du nom de Leaflet capable de générer des cartes interactives.

C'est même l'API utilisé sur le site principal d'Open Street Map. On aurait pu passer les données en POST et faire tourner un petit serveur node-js renvoyant les pages, qui une fois chargé dans les WebViews auraient été bien plus stables car il y aurait beaucoup moins à charger.

C'est une perspective d'amélioration convenable, mais cela sortirait du cadre du projet et demanderait du temps supplémentaire. Cependant, c'est un changement qui une fois de plus ne modifierait en rien le fonctionnement de l'application.

Dernière ligne droite

Après avoir tenu nos délais, la dernière ligne droite consistait en quatre tâches, réparties entre les membres du groupe :

- Mise à jour du diagramme de classes
- Rédaction d'un document décrivant en détails le set-up du logiciel
- Dump de la base de données
- Correction orthographique des 90 classes et des différents documents

Ces 4 tâches ont été réalisées hors réunion par chacun des membres du projet, puis mises en commun lors de notre dernière réunion de 15h01, le 5 janvier.

Participation personnelle

BERTHEL Gabriel

- Classe Multiton de Connexion à la base de données, et ses tests.
- Classe ConfigManager pour lire simplement config.properties
- Classe DateManager pour gérer les conversion de/vers Timestamp
- Classe ValidateurAdresse ensuite étendue par Adrian.
- Débogage et complétion des DAO avec Léo et Adrian en vocal.
- Classe validatrice de données, sauf code postal et mot de passe.
- Extension de plusieurs formulaires de validation
- Quelques contrôleurs (mais surtout du débogage)
- Mise à jour des ListView
- Soirée debuggage avec toute l'équipe

Tests:

- Injection SQL, Connecteur BDD
- Tests classe validatrice de données, sauf code postal et mot de passe.
- Tests des DAO avec l'ensemble du groupe.

J'ai l'impression d'avoir été le plus actif lorsque le groupe n'avait pas le moral, et un peu plus en retrait quand tout le monde avançait vite. Je me rappelle d'un soir assez difficile ou l'on a dû git-revert à cause de gros soucis liés à la propagation des données dans les objets-métiers. J'ai persévéré toute la nuit pour trouver une solution, qui a ensuite été entendue par Adrian le lendemain.

J'ai aussi fait beaucoup de débogage après la réunion car c'est souvent le moment où je me sentais le plus à l'aise. Souvent quand Leo décelait des bugs, je me retrouvais à déboguer ce qui avait été fait dans l'après midi.

Je pense sincèrement avoir pu débloquer la citation à plusieurs reprises, ou en tout cas faire en sorte que les problèmes soient derrière nous pour la réunion du lendemain. Pour le sommeil, ça fait du bien de savoir que les problèmes sont réglés.

Je tiens a préciser que l'ensemble du groupe a été très investi en vocal, et on s'est beaucoup entre-aidés. Il y énormément de travail qu'on ne voit pas forcément avec le gestionnaire de versions.

Je pense sincèrement que sans être en vocal avec Léo et Adrian, on ne s'en serait pas sorti sur les DAO. En parallèle, avoir Romain pour s'occuper des vues et des liaisons avec les contrôleurs nous a permis de nous concentrer sur le modèle sans préoccupations, puis d'implémenter les contrôleurs tous ensemble.

C'est le premier projet de ma vie où je suis content d'être en groupe.

LAGASSE Adrian

- La plupart des DAO
- Les contrôleurs pour les consultations et la modification (avec la coopération de Romain pour les Tournées)
- Le calcul et la vérification des trajets
- La connexion des utilisateurs
- Certains validateurs de formulaires
- Diagramme de classes

J'ai apprécié la manière dont nous avons travaillé avec mes camarades, nous étions bien organisés et la division du travail nous a bien aidé. Certaines fois, les problèmes nous faisaient déprimer ou perdre notre sang froid mais nous les résolvions ensuite.

Les réunions organisées par notre SCRUM Master nous ont permis d'avancer vite et de rester concentrés sur le travail, en plus de pouvoir rapidement poser des questions lorsque nous ne comprenions pas quelque chose.

Le problème qui m'a posé le plus de soucis était celui de la récursivité infinie dans les DAO. J'étais d'abord parti sur une implémentation indépendante de chacun des DAO sauf que, lors des premiers tests, nous nous sommes rendu compte que les DAO s'appelaient entre eux mutuellement. Grâce à Gabriel, nous avons pu trouver une implémentation qui ne pose pas de problème lors de la création de chacun.

J'ai eu quelques difficultés à faire fonctionner correctement le calculateur de Tournée. Il a fallu trouver une bibliothèque gérant le JSON en Java, bien formater la String à envoyer puis récupérer les données et les extraire, Java ne nous a pas facilité la tâche. Ensuite, il a fallu manipuler et synchroniser les listes de Commandes et les durées du trajet grâce à des itérateurs. La chose la plus compliquée et pour laquelle nous n'avons pas compris le fondement de l'erreur a été avec les Timestamp. Étant des objets je pensais qu'un simple passage par référence et une modification via le setter permettait de pouvoir manipuler la même donnée une fois de retour dans le contexte initial. Le problème fut résolu en recréant l'objet à chaque modification.

Je suis content du résultat que nous avons produit et de la rapidité avec laquelle nous avons pu produire une version fonctionnelle. J'espère pouvoir retravailler avec mes camarades lors d'un futur projet.

MSIAH Romain

- Création du logo.
- Modifications des vues (lors de problèmes repérés ou d'améliorations suggérées)
- Refonte graphique du logiciel (style.css)
- Implémentation de DateManagerTest.java
- Implémentation initiale de ControllersUtils.java, améliorée par Gabriel
- Implémentation de UserAuth.java.
- Implémentation de ProdSelectMenuCtrl.java.
- Implémentations initiales de AddTourCtrl.java et ModifyTourCtrl.java, largement améliorées par Adrian.
- Implémentation initiale de FormAddTourValidator.java, avant modification des contrôleurs précédents.
- Tri des éléments dans les vues avec des ListView.
- Modification de l'affichage des éléments des ListView.
- Créations des vues FXML de changement de mot de passe.
- Création et modification du fichier viewsInfo.odt, qui donne des informations sur les vues FXML pour aider lors de l'implémentation des contrôleurs.
- Création du dump

Comme mes camarades appréhendaient l'aspect graphique du projet et que j'ai un attrait particulier pour tout ce qui est du côté visuel, j'étais l'intermédiaire entre le groupe et les vues FXML. C'était ma zone de confort, donc j'ai entrepris de travailler sur la refonte graphique du logiciel. Bien que ce n'était pas la partie prioritaire, j'ai pris cette initiative, car je pense qu'un visuel satisfaisant permet une certaine motivation, une idée de progression. J'ai ensuite aidé le reste du groupe sur la partie dans laquelle je suis le moins à l'aise. Néanmoins, même si ma contribution me semble moins importante, je suis persuadé d'avoir aidé les autres dans l'implémentation des contrôleurs. Il a souvent fallu des modifications par les autres membres du groupe, mais j'étais très présent pour répondre aux différentes questions sur les vues.

Pour ce qui est de mon ressenti global du projet, je trouve que nous avons fait un très bon travail et que, en tant que groupe, nous avons été efficaces et bien organisés.

REMY Léo

- Banalisation des jours de fêtes
- Organisation des réunions de 15h01
- Mise à jour du backlog en temps réel
- Trace de l'avancée de chacun
- Organisation des sprints et orientation du travail à effectuer
- Tests des DAO avec l'ensemble du groupe
- Validateur de code postal & son test
- Choix de l'API OpenRouteService et de l'affichage par WebView
- Génération des URL Google Maps & son test
- Implémentation des contrôleurs avec l'ensemble du groupe
- Attention particulière aux validateurs des formulaires d'ajout
- Vérification de la prise en compte des contraintes
- Test du logiciel et relecture des classes
- Correction orthographique globale
- Plan du rapport et rédaction des parties générales

Mon rôle pour ce dernier Jalon s'est orienté vers la partie organisation, qui me tient particulièrement à cœur et est pertinente dans mon projet de poursuite d'études. Ayant choisi des parties de l'implémentation pas spécialement complexes et ne rencontrant pas de réel problème, j'ai pu prendre le temps de vérifier les différentes parties pour peaufiner au maximum le logiciel.

J'ai essayé au maximum de prendre en compte les problèmes soulevés lors du deuxième jalon pour les corriger et permettre un confort de travail pour l'ensemble du groupe.
L'organisation des réunions de 15h01 a permis de renforcer non seulement le rythme de travail, mais aussi la confiance que chacun pouvait avoir dans le sien et celui des autres.
L'implémentation des DAO et des contrôleurs a été un travail d'équipe particulièrement enrichissant et étonnamment agréable malgré le nombre de problèmes soulevés.
Je ne sais pas si c'est spécialement grâce à moi (à mon avis c'est grâce au travail et à la rigueur de tous), mais nous avons pu tenir nos délais sans se sentir pressés par le temps ou écrasés par la charge de travail.