BERTHEL Gabriel - LAGASSE Adrian - MSIAH Romain - REMY Léo

Conception et visualisation de tournées agricoles en circuits courts.

Table des matières

Progression du projet :

Contact avec le client :

Après avoir finalisé le Jalon 1, il a été rapidement convenu d'une présentation de celui-ci à Mr. DESPORT afin d'obtenir son avis sur le travail effectué, des points à améliorer selon lui, et plus globalement pour être certains d'être sur la même longueur d'onde concernant le logiciel.
Pendant cette présentation, tout avait l'air de convenir, et nous en avons profité pour demander quelques précisions.

Ainsi, nous en avons dégagé ces points importants :

- Il serait préférable que lors de l'affichage d'une Tournée, on affiche les détails d'une Commande au clic sur le point correspondant sur la carte
- Lors de l'affichage d'une commande, afficher si possible le kilométrage total
- Si la validation de la Tournée est impossible avec OpenStreetMap, faire un vol d'oiseau
- Si on a le temps après avoir finalisé le projet, intégrer dans la validation d'une Tournée un temps de battement à chaque Commande pour simuler un déchargement
- Lors de l'affichage d'une Tournée, on prévoit le départ du Producteur de chez lui et son arrivée chez lui. Cela doit aussi être pris en compte dans le calcul des heures de départ et d'arrivée
- Au lancement de l'application, vérifier la connexion Internet
- Pour l'interface graphique, lors de l'utilisation de Listeners, préférer l'usage d'expressions lambda aux classes anonymes, difficiles à maintenir

Des questions ont également été posées concernant OpenStreetMap, mais Mr. DESPORT ne pouvant pas nous aider sur le sujet, nous lirons la documentation.

Conventions de codage :

Lors de la réunion concernant les conventions de codage, nous avons décidé des conventions suivantes :

- Accolades à côté des noms
- Un attribut par ligne
- camelCase
- Majuscules aux noms de classes
- Tab => 4 espaces
- Un saut de ligne entre chaque méthode (on ne colle pas les blocs)
- Commenter la dernière accolade
- Inverser les clauses (if) si possible
- Pas de oneliner sauf si ternaire possible sur le return
- Étendre Exception dans un package dédié pour avoir des Exceptions personnalisées
- Saut de ligne entre bloc déclaration et bloc logique
- Pas de saut de ligne dans les blocs sauf déclaration/logique
- Espace entre valeurs et opérateurs
- Espace après les virgules
- Espace après if, for, else
- Package en minuscule sauf acronyme
- CONSTANTE
- Commentaire JavaDoc
- IdBDD, port, tous les trucs de setup dans un json à part
- Documenter chaque classe

La plupart de ces conventions de codage seront vérifiées à l'aide du plugin checkStyle sur Eclipse, et son équivalent sur IntelliJ. Un fichier de configuration se trouve dans src/ressources

Lors de cette réunion ont également été dégagées des conventions concernant Git et les différents commits.
Ainsi, les commits doivent contenir un maximum de mots français, l'anglais pouvant être présent lorsque nécessaire ("pull request" par exemple est plus clair en anglais). Chaque aspect du logiciel doit avoir sa branche, et chacun d'entre nous a la sienne.

User stories et constitution du backlog :

Une réunion a également été organisée pour écrire les user stories.
Les prérequis du Jalon 1 ne nous ayant pas inscrits à 100% dans une démarche Agile, nous n'avons pas vraiment besoin d'écrire ces user stories. Cela a tout de même été réalisé par souci de clarté, et pour faciliter les tests.
L'écriture des user stories prend en compte les contraintes levées lors du premier jalon.

Les user stories et leur décomposition simple sont dans le fichier User_stories.pdf, présent en annexe.

Écriture des equêtes :

L'écriture des requêtes a été réalisée avec l'aide des différentes user stories et contraintes relevées plus tôt. Nous avons donc une à plusieurs requêtes par user story.

Les requêtes ont été écrites en gardant en tête l'utilisation de JDBC et ses PreparedStatements, d'où la présence de "?" partout.
Chaque requête a été écrite en langage naturel et SQL.
Les requêtes de sélection ont en plus été écrites en algèbre relationnelle, calcul relationnel de domaines et calcul relationnel de tuples.

Les requêtes d'insertion, de modification et de suppression n'ont pas droit à ce traitement puisque nous n'avons pas vu en cours leur traduction en langage relationnel.

Les requêtes rangées par user story se trouvent dans le fichier Requêtes.pdf, présent en annexe.

Documentation :

La documentation a été réalisée en suivant les normes de la JavaDoc. Les classes possèdent chacune une description de leur rôle. De même pour les méthodes, à l'exception des accesseurs. Pour chacune de ces méthodes étaient décrits ses paramètres grâce à @param et son éventuel retour via @returns.

La documentation a d'abord été écrite via StarUML. Une fois cela fait, nous avons généré le code et la documentation associée.

Il y a cependant eu un problème, les fichiers générés contenaient des annotations propres à StarUML, rendant la lecture du code moins lisible et sa différenciation avec la documentation plus compliquée.

Nous avons donc cherché à supprimer ces annotations de manière systématique grâce aux outils offerts par GNU/Linux. La commande suivante nous a permis de le faire rapidement : find -type f -exec sed -i -e '/\/\/ ---------- /d' -e '1,5d' {} \;

Une fois la commande effectuée, toutes les annotations ont disparu ainsi que les 5 premières lignes indiquant que StarUML avait été utilisé.

StarUML ne générant pas très bien l'héritage d'une classe abstraite avec générique, nous avons dû adapter manuellement la documentation et le type du générique de chaque DAO.

Sécurité et tests :

Nous interagissons avec des utilisateurs en permanence, ce qui représente un risque d'injection SQL. Ainsi, il sera impératif d'utiliser des requêtes préparées afin de les éviter.

Utiliser des requêtes préparées, c'est aussi jouir de la sanitation des données fournie par le pilote JDBC-MySQL maintenu par Oracle. Il est d'ailleurs possible de récupérer le code source et de le compiler soi-même.

Néanmoins, dans le cadre de ce projet, nous avons tenu à effectuer deux injections SQL (une de premier ordre, et une de second ordre) via les tests unitaires. Si ce n'est pas exhaustif, cela permet tout de même de vérifier que les requêtes préparées protègent bien des injections les plus courantes.

L'on pourrait écrire plus de tests dans le futur afin de vérifier si une sanitation supplémentaire est nécessaire.

En parallèle, le compte root par défaut sera supprimé et nous nous connecterons à la base via un compte logiciel avec des droits en moins. Par exemple, le drop est impossible.

Nous distinguons d'ailleurs trois usages: développement, tests et production. Ainsi, il y aura trois scripts de mise en route de la base de données. En effet, la base de tests ne peut être la base en production. Quant au développement, cela nous permet d'accéder à une version modifiée sans pour autant en changer pour le reste de l'application.

Ainsi, le connecteur permet de gérer plusieurs connexions que nous chargeons via un fichier de configuration (src/ressources/config.properties) et que nous identifions par l'environnement (development, testing, production). Ces dernières sont dans un HashMap ce qui garantit l'unicité pour chaque identifiant.

Ce connecteur est d'ailleurs testé. Nous vérifions que créer une instance, par exemple de production, renvoie bien la même instance s'il l'on redemande production juste après. Nous vérifions aussi qu'invoquer une instance qui n'existe pas dans la configuration ne renvoie à rien. Enfin, nous vérifions que si la clef est dans l'HashMap, alors la connexion est bien créée.

Enfin, il faudra garantir l'uniformité des données. Ainsi, à déjà été mise en place la classe validatrice de données et ses tests. Si les requêtes préparées fournissent un rempart aux injections, il est tout de même intéressant de garder des données homogènes. Lorsqu'il s'agit de SIRET ou de plaque d'immatriculation, nous connaissons à l'avance les formats à respecter.

Il manque quelques méthodes à implanter telles que les heures ou les adresses, mais cela donne déjà une idée de la majorité des formats à respecter. Quant aux adresses et aux coordonnées GPS, il pourrait être intéressant d'utiliser l'API de géocodage fournie par le gouvernement français https://adresse.data.gouv.fr/api-doc/adresse. Il est aussi bien possible d'obtenir des coordonnées à part d'une adresse que l'inverse, ce qui pourrait s'avérer très pratique.

Sont ainsi implémentées :

- src/utility/DatabaseConnection.java
- src/utility/ValidateurDonnee.java

Leurs tests respectifs étant :

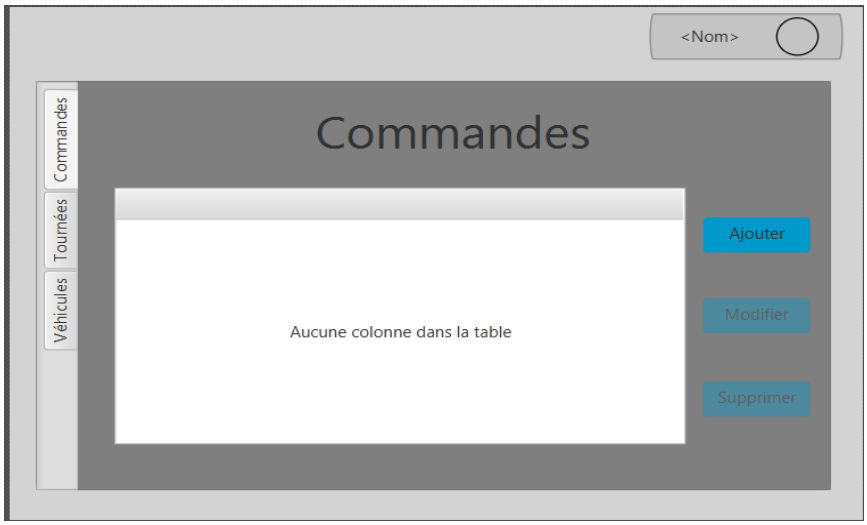
- src/tests/database/DatabaseConnectionTest.java
- src/tests/database/SQLInjectionTest.java
- src/tests/utility/ValidateurDonneeTest.java

Vues :

Comme notre projet est conçu avec une architecture 3-tiers en tête, nous avons choisi d'utiliser des fichiers FXML pour représenter nos vues, qui seront chargés par les contrôleurs des vues correspondantes. Pour créer l'interface graphique de notre application nous avons opté pour l'utilisation du logiciel SceneBuilder, très pratique pour générer des fichiers FXML. Nous avons suivi au maximum le visuel de notre maquette, favorisant la praticité à l'esthétique, qui nous semble moins prioritaire pour ce jalon numéro deux. Effectivement, nous nous sommes concentrés sur deux aspects : est-ce que nos vues contiennent toutes les informations nécessaires et est-ce que l'expérience utilisateur nous semble satisfaisante ?

Nous avons donc privilégié la clarté par rapport à l'affichage de toutes les informations dont un producteur ou un administrateur pourrait avoir besoin. De plus, nous nous sommes rendu compte que notre application pourrait nécessiter énormément de vues différentes, donc nous avons pensé, pour certaines d'entre elles, un agencement qui nous permettrait de centraliser les informations d'une manière propre et claire, mais qui rendrait nos contrôleurs plus volumineux.

Cette décision peut se voir à la première page qu'un utilisateur peut voir après sa connexion. Ces deux vues, un côté producteur et l'autre côté administrateur, montrent plusieurs onglets qui permettent de consulter diverses informations qui dépendent du type de l'utilisateur. Chaque onglet affiche une liste et plusieurs boutons qui permettent d'ajouter un élément à celle-ci ou de manipuler un des éléments une fois sélectionné, comme la modification ou suppression de cet élément.



Exemple 1: Menu de sélection des producteurs

Si l'on veut entrer plus en détails dans la structure des vues de cette application, nous nous sommes convenues sur des conventions pour maintenir une certaine cohérence d'une vue à l'autre:

- Les titres en police 25-35 (relatif en fonction de la taille du titre).
- Les sous-titres en taille de police 25.
- Les labels et autres éléments textuels en taille de police 20.
- Le nom des attributs (fx:id) qui représentent les éléments de la vue en camelCase, pour respecter les conventions établies pour l'implémentation de l'application.

Nous avons tout de même quelques doutes sur certains éléments des vues, notamment lorsqu'il est question d'afficher un grand nombre d'informations. Nous hésitons entre l'utilisation des listViews ou des tableViews. Nous ne savons pas encore si nous utilisons des observableLists, qui sont des collections qui permettent la communication entre la vue et le modèle, car un événement est lancé à chaque ajout ou modification d'une donnée dans cette collection.

Cela permet aussi un changement en temps réel de la vue lorsqu'une information est modifiée ou ajoutée. Le dilemme est que nous ne savons pas si nous aurons besoin de ces collections observables, car les modifications des informations se font en grande partie dans des pop-ups et nous ne savons pas comment la vue réagit lorsqu'une information est modifiée en dehors de la fenêtre d'affichage.

Les observableLists ne fonctionnent pas avec les listViews, mais fonctionnent avec les tableViews, de plus, les tableViews sont généralement utilisées pour afficher des informations sur plusieurs colonnes, ce qui ne nous intéresse pas forcément. Bien que la différence soit minime et, en partie négligeable, nous ne savons donc pas encore s'il est pertinent d'utiliser des listViews ou des tableViews.

L'inventaire de tous les éléments modifiés pour ce jalon se trouvent dans ajouts-jalon-2.txt

Participation personnelle :

BERTHEL Gabriel :

Jalon 1:

- Participation aux réunions
- Création du git-hub et écriture du READMe
- Première jet des conventions de codage
- Mise au propre de l'Impact Map
- Deux diagrammes de séquences niveau système (Création et validation d'une tournée et affichage d'une tournée valide).
- Un diagramme de séquences niveau système (Gestion des comptes)
- Ajout contrôleurs au diagramme de classes, et réorganisation de certains éléments (avant le jalon 1, corrigé par la suite)
- Écriture du premier rapport à l'exception de la partie concernant la maquette, ce qui a permis de décharger le groupe

Jalon 2:

- Implémentation, écritures et documentation de:
  - src/utility/DatabaseConnection.java,
  - src/tests/database/DatabaseConnectionTest.java
  - src/tests/database/SQLInjectionTest.java
  - src/tests/utility/ValidateurDonnee.java
- Mise à jour du connecteur sur le diagramme de classes, ajout des méthodes propres au validateur de données. Rectification d'une erreur que j'avais introduite.
- Structuration des fichiers et mise à jour du READMe

Lors du premier jalon et suite au travail d'Adrian, j'ai effectué une réorganisation du diagramme de classes. Ensuite, j'y ai complété le MVC suivant un exemple dans le cours. Cependant, nous utilisons JavaFX et non Swing. De plus, j'avais mal compris le principe de classe concrète. Cela aurait pu "mettre en danger" le projet, cependant le titre est désormais rectifié puisque j'ai corrigé mes bêtises en mettant à jour le diagramme de classes.

Plusieurs fois je me suis senti frustré de ma contribution car j'ai beaucoup de mal lorsque chacun choisit ses tâches librement parmi un large éventail. J'ai l'impression d'être en attente perpétuelle et de toujours me retrouver à la ramasse. C'est pourquoi j'ai écrit une grande partie du rapport. J'ai voulu décharger le groupe à un moment où nous avions des examens en plus des échéances.

C'est aussi pourquoi j'ai tenu à implémenter, tester et documenter les connecteurs et la classe validatrice de données. Je suis conscient que c'est contraire à la méthodologie AGILE mais Romain travaillant sur les vues compte tenu de son expérience, j'avais vraiment besoin de faire quelque chose de concret pour apaiser mon anxiété.

Pour être tout à fait honnête, j'ai vraiment beaucoup de mal avec la méthodologie AGILE que nous ne pouvons pas respecter. Nous n'avons strictement aucune expérience dans la réalisation de MVC. Nous découvrons JavaFX et nos TD en salle machine n'ont pu être réalisés dans de bonnes conditions ayant passé la grande partie du temps à installer les librairies.

Néanmoins, je crois sincèrement que l'implémentation des contrôleurs et du modèle pourrait être plus "agile".

Personnellement, j'aimerais qu'on ait des sprints plus courts, avec des thématiques bien précises. Par exemple… Dans les 3 jours à venir, on s'occupe de l'authentification des producteurs et administrateurs. Qui s'occupe du modèle, qui s'occupe du contrôleur, qui fait les tests, qui fait la doc.

Je ne parle que pour moi, mais je pense qu'on devrait tenter de travailler en vocal sur le serveur Discord. Cela serait plus motivant de tous collaborer sur une fonctionnalité bien précise et communiquer au fur et à mesure de ce développement.

Rien n'interdit d'être de son côté, mais il faudrait vraiment implémenter la réunion quotidienne. Soit pour partager son avancement, soit pour travailler en parallèle.

LAGASSE Adrian :

Lors du premier jalon, j'ai d'abord participé à l'écriture des scenarii nominal et alternatif. Ensuite j'ai réalisé le MCD sous JMerise, puis le diagramme de classe des DAO et des objets métier sur StarUML (Gabriel l'ayant ensuite complété avec les contrôleurs et soigné la mise en forme).

Pour le second jalon, j'ai réalisé la documentation et effectué les opérations indiquées dans la partie documentation. J'ai ensuite écrit les scripts SQL de déploiement de la base de données. Enfin, j'ai relu les requêtes écrites par Léo afin de corriger quelques erreurs.

J'évalue mon implication à moyen. J'ai effectué les tâches qui m'étaient attribuées et proposé mon aide pour des relectures. Cependant, ma motivation n'était pas toujours là et j'ai retardé plusieurs fois certaines tâches au profit d'autres activités.

MSIAH Romain :

- Création de quelques diagrammes de séquence.
- Retranscription de la maquette papier (faite par Léo) sur Figma.
- Création des vues du dossier src/views.

Pour ces 2 jalons, mon travail était surtout de l'assistance pendant nos nombreuses réunions, sur les différents éléments du projet que nous avons pensés ensemble. Par exemple la toute première version du MCD (mise au propre ensuite par Adrian), la création des scénarios d'utilisation et les conventions de codages, qui ont été choisis selon les préférences de la majorité. Au premier jalon, j'ai reproduit la maquette papier, conçue par Léo, sur l'outil en ligne Figma et j'ai proposé des légères modifications de la maquette originale en gardant les idées de design pensées par Léo.

Si l'on se concentre sur mon travail, purement personnel, au premier jalon, j'ai fait des diagrammes de séquences sur StarUML, notamment ceux concernant la connexion des producteurs, l'inscription des administrateurs et l'ajout d'un véhicule.

Pour le second jalon j'ai utilisé SceneBuilder pour concevoir les vues de notre application. J'ai suivi un maximum la maquette conçue au premier jalon, qui contenait plus ou moins déjà des idées sur l'apparence et l'esthétique des vues, en plus des éléments utiles à chaque vue comme des boutons ou des champs. J'ai ensuite repris les conventions de codage qui ont été établies au début du jalon pour faire un fichier de configuration checkStyle.

En termes d'implication, j'aurais clairement pu être plus actif. J'ai notamment pris beaucoup de temps pour faire les vues, ce qui se voit sur le manque de créativité des vues par rapport à la maquette. ma perte de motivation, vers la fin du second jalon, a rendu la création du fichier de configuration checkStyle plus lente qu'elle n'aurait dû être.

REMY Léo :

- Création du serveur Discord
- Organisation d'une partie des réunions
- Contacts avec le client
- Mises à jour régulières du contenu présent sur le serveur
- Travail de "secrétaire" pendant les réunions
- Dégagement des contraintes futures
- Mise en place du MCD en collaboration avec l'ensemble du groupe
- Sprint 0 avec l'ensemble du groupe
- Scénarios d'utilisation avec l'ensemble du groupe
- Définitions des sprints
- Écriture des user stories
- Constitution première du backlog
- Premier jet de la maquette
- Écriture des requêtes
- Écriture des parties concernant la maquette dans le rapport du jalon 1
- Écriture des parties concernant le contact avec le client, les conventions de codage, user stories et constitution du backlog, requêtes dans le rapport du jalon 2
- Relecture des rapports et correction orthographique

Je trouve ma participation actuelle largement insuffisante. Je suis présent lors des réunions et participe à tout le travail collectif, mais mon travail individuel est pour l'instant insuffisant par rapport à celui des autres membres du groupe. Gabriel étant autant voire plus impliqué que moi dans l'organisation même du projet, je ne sais même pas si l'on peut encore me considérer comme le SCRUM Master.

Malgré tout, mon travail insuffisant n'a pas fait trembler le projet une seule seconde. L'implication des autres membres est ce qui le fait tenir debout.

Je pourrais penser que l'ajout de fonctionnalités et l'utilisation d'outils de plus en plus complexes par chacun des autres membres peut être dangereux pour la réalisation du projet (il ne reste que 20 jours après ce jalon), mais la vérité est que je devrais plus m'impliquer pour maîtriser ces outils. Avec la fin des cours et des contrôles continus, j'ai à présent tout le temps nécessaire pour m'impliquer dans le projet.

L'implémentation étant ma partie de prédilection, c'est maintenant que je dois fournir le plus d'efforts, pour ne pas avoir à pâlir devant les autres.

Selon moi il n'y a rien d'autre à ajouter qu'une meilleure implication de ma part pour faire fonctionner correctement le groupe. Chacun a son domaine de prédilection sur lequel il travaille depuis le début, le groupe fonctionne parfaitement et le projet avance pour le moment sans difficulté.