Projet Nostos Marine

1) Sujet initial et objectifs

Nostos Marine propose d'acheminer des marchandises entre Saint Malo et Jersey. Il possède un bâteau et un entrepôt. Pour le moment les clients sont les transporteurs de l'île qui commande différents *shipments.* Ces *shipments* vont être livrés par camions généralement à Saint Malo dans l'entrepôt par des *dropoffs* différents. Ils vont transiter alors dans l'entrepôt jusqu'à être défini dans des *groupages* qui vont appartenir à un *trip* correspondant à leur voyage de traversée jusqu'à l'île. Ils sont alors confiés au client, la plupart du temps, le *freight forwarder* qui va les acheminer aux différents destinataires.

Objectif principal : créer une base de données en python qui *modélise* les différentes interactions et objets pour pouvoir suivre le statut des différents paquets et en connaitre les caractéristiques (en réfléchissant à une interface)

Objectifs secondaires (au choix) : prévoir un plan de chargement optimisé dans les conteneurs des *shipments* / prévoir les plannings en fonction des arrivages dans l'entrepôt / prévoir les groupages appropriés et l'ordre d'entrée des paquets en fonction de leur adresse de destination / utiliser une ORM / éditer un cargomanifest

Demandes Nostos Marine :

- Utiliser au maximum la librairie standard de python
- Modulariser le code
- Effectuer les tests avant le code
- Utiliser GitHub avec des Pull Request sur les différentes branches

2) Historique du projet

Dès le début du projet, deux parties principales ont été délimitées : la logique métier et l'interface.

AVRIL et MAI : MODÉLISATION de l'ENTREPÔT et INTERFACE

-> Logique

Modélisation

Changements de modélisation pendant les deux mois avec affinement. Le problème était les définitions sur lesquelles nous n'étions pas toujours en accord, nous avions du mal à identifier les objets impliqués et la façon dont ils interagissaient.

Nous avions commencé avec seulement des *packages* qu'on pouvait ajouter, supprimer modifier dans la base de données. Puis il y a eu des *inshipments* et *outshipments* liés aux sorties et entrées dans l'entrepôts qui ne correspondaient pas toujours à une entité physique. Nous les avons transformés en *shipments* (commande du *freight forwarder*) et *dropoffs* (un camion) qui correspondent mieux à la réalité.

Une longue réflexion sur le type de nos objets, qui a évoluée nous a également permis d'arriver à la version finale qui est utilisée par l'interface : *NamedTuple* ou classe en héritant pour les packages au début, set, classe en héritant pour les *Shipments* avant de carrément créer une classe

TypedSet pour généraliser le concept de set contenant un type d'objet bien précis, qui apparaissait de manière récurrente dans la modélisation de notre base de données.

Le TypedSet est en lui-même le symbole de notre réflexion sur le choix de la structure de donnée. C'est la concrétisation de nos principes, simple d'utilisation car intuitif et parfaitement adapté au contexte. Nos objets ont beaucoup évolué au cours du projet. Comme le mot d'ordre était simplicité, l'idée était d'essayer d'avoir les objets les plus proche des types de base pythons. A chaque que l'objet se complexifiait, il fallait donc faire évoluer son type pour s'adapter. Ce travail a été plutôt difficile au début, mais vraiment formateur.

Les noms des objets se sont révélés eux aussi fondamentaux pour savoir directement de ce dont on parle.

Caractéristiques et attributs des objets : le poids important.

Sauvegarde et décorateur

La sauvegarde a également été une problématique importante. Comment garder en mémoire notre base de données et préserver l'unicité des identifiants générés ? Le problème majeur était que modifier un objet contenant des packages impliquait de modifier les packages et donc de réécrire la database. Nous avons donc opté pour un décorateur se chargeant d'ajouter à toute opération de modification sur un élément de la structure de donnée sans sauvegarde pour ne pas être sensible à la casse. Plus précisément nous avons commencé par utiliser un simple fichier texte où les packages étaient répertoriés. La structure se complexifiant, nous avons remplacé se mécanisme par une sauvegarde avec pickle. Il a fallu pour certains de nos objets un peu comlexes (TypedSet), s'adapter à ce choix, car ceux-ci n'étaient pas correctement sauvegardés par pickle. Nous avons donc dû changer le __reduce__ de cette classe pour avoir une sauvegarde efficace.

Attribuer un id unique aux objets

Nous avions besoin de générer des id unique pour nos packages. Au départ, la génération se faisait dans la classe avec une simple variable globale de classe. Ceci n'était pas très "pythonique", nous avons ensuite utilisé un itérateur dans la classe, ce qui était déjà plus clair mais posait des problèmes structurels (peu pratique pour la sauvegarde) et n'était pas très modularisé. Nous avons finalement opté pour une classe de générateurs d'id.

Shipments

Le shipment était au début un simple dictionnaire. Il y avait dans ce choix une redondance : la clé était l'identifiant du package mais était également stockée dans le package. Nous avons ensuite opté pour une classe dérivant de set en définissant une *get-method* pour garder la possibilité de retrouver un paquet spécifique. Plus tard, cette réflexion nous a pour généraliser la démarche et créer un *TypedSet* ne contenant plus directement la donnée des paquets, mais utilisant une property allant les chercher dans la database.

-> Interface

Choix du mode d'interaction avec l'utilisateur

Nous avons d'abord choisi de nous documenter et d'utiliser argparse. Nous avons rencontré plusieurs problèmes : à chaque ligne de commande on re exécute le code ce qui pose le problème de la persistance des données immédiatement, peu lisible pour l'utilisateur.

Nous nous sommes redirigés vers un fichier python **interface.py** qui permettait d'interagir avec l'utilisateur en lui demandant ce qu'il veut visualiser et en présentant les fonctionnalités (présentation dans la documentation du code).

Dans ce fichier, nous avons commencé par réécrire l'interface avec des **input**. Cela a ensuite posé des problèmes pour la validation de la forme des données (chiffres, id existants...). Il aurait fallu gérer les erreurs de façon générale dabs le code. Nous avons donc décidé d'utiliser **click** avec les conseils de notre encadrant. Le type **Click.Choice** permettant d'indiquer à l'utilisateur s'il rentre une mauvaise donnée.

Choix de la persistance des données

Nous écrivions au début les données dans un fichier csv ce qui nous semblait pertinent pour l'exploiter avec plus de facilité pour l'utilisateur ensuite.

Cependant, dès qu'on modifier nos objets dans la partie logique, il fallait changer la façon d'écrire dans le csv de l'utilisateur. De plus, on ne pouvait pas ouvrir le csv en même temps de se servir de l'interface. Nous avons donc changé de façon de sauver les informations en utilisant **pickle,** ceci était bien plus simple et beaucoup plus efficace (quoique plus technique pour certains détails pointus)

Choix des fonctionnalités pratiques pour l'utilisateur

A force de se servir de l'interface pour la tester, nous nous sommes aperçus de fonctionnalités qui pourraient améliorer l'interaction avec l'utilisateur :

- Rentrer les paquets par références pour ne pas avoir à tout re remplir mais faire X paquets identiques directement
- Modifier les objets qu'on a défini que ce soit pour les objets qu'ils contiennent ou certaines de leurs caractéristiques qui peuvent changer (ex : date)
- Afficher l'id attribué automatiquement au paquet rentré ou tout autre objet
- Pouvoir sortir de la saisie en mettant un décorateur qui gère le **contrôle c**
- Afficher dans un *view* les objets de la base de données

Modularisation des données

Le fichier **interface.py** est devenu de plus en plus complexe et mélangeait l'usage de fonction métier de bas niveau, séparer la persistance, les lignes d'interfaces pures. Après différentes étapes, nous avons réussi à décortiquer le code :

- Service layer : contient les fonctions liées à la logique métier
- Display : contient toutes les fonctions d'affichages qui sont liées aussi à la logique métier
- Interface Commands : fonctions qui permettent de faire les opérations demandées par l'interface en passant par les autres fichiers
- Prompt : fonctions regroupant les entrées de données de l'utilisateur
- Confirm : fonctions qui demandent les confirmations à l'utilisateur de ce qu'il a rentré

Cette modularisation nous a permis à chaque évolution ensuite de la logique de modifier plus efficacement l'interface et s'est révélé comme un atout.

JUIN : FINALISATION de la MODÉLISATION et GESTION des TRIPS

Nous avons enfin le dernier mois traité la partie du transfert des colis de l'entrepôt à Jersey. Nous avons donc ajouté des fonctionnalités de l'interface liées aux conteneurs, voyage et groupages. Nous avons aussi veillé à adapter la partie logique dans les caractéristiques de chaque objet. Pour cela, nous avons décidé de faire :

- Une fonction qui permet à partir d'un *trip* donné par l'utilisateur donne un plan de chargement de conteneurs en tenant compte que les *shipments* soient dans le minium de containers possibles
- Une fonction qui permet de générer un **pdf** de ce plan de chargement regroupant les images obtenues avec **matplotlib**
- Une fonction qui permet d'obtenir le Cargo Manifest du voyage
- Une fonction qui génère un pdf du planning d'arrivée

Nous avons aussi ajouté un générateur de pdf pour avoir un planning d'arrivage des *dropffs* dans l'ordre.

3) Bilan et objectifs atteints

Objectif principal : la base de données a été construite dans python, nous avons modularisé, il y a des vus disponibles dans le terminal en lançant l'interface, les modules utilisés font partie en majorité de la librairie standard, la modélisation a été faite sur appui des demandes et description de Pierre Venin en essayant d'être la plus simple possible sans perte d'information, le stockage et la sauvegarde des données est faite au fur et à mesure, on peut accéder à toutes les données par l'interface et les modifier.

Objectifs secondaires : pdf du planning d'arrivée des *dropoffs*, cargo manifest, plan de chargement du trip