LES MINIMES EN UN CLIC

Projet d'informatisation de la gestion du port de plaisance.

Remise en contexte du projet.

Le gérant du port de La Rochelle a décidé de solliciter notre société afin de l'aider à mettre en place une version informatisée de la gestion de son port ce qui peut également permettre d'avoir une comptabilité en partie informatisée.

Dans le port de La Rochelle on peut trouver deux types d'usagers :

- Les **abonnés**, ils règlent chaque début de mois une cotisation annuelle.
- Les **passagers**, ils règlent à la fin de leur séjour un prix qui varie en fonction du nombre de jours qu'ils ont passé au port.

Chaque usager arrive avec un bateau qui peut être catégorisé par un type :

- soit un bateau **non habitable** (inférieur à 10 m et qui n'utilise aucun service).
- soit un **voilier de type 1** (entre 10 et 25 m, qui peut demander des **services** tel que *l'eau* ou *l'électricité*).
- soit un **voilier de type 2** (supérieur à 25 m, et peut également accéder aux services) .

Quand un usager arrive au port il faut donc lui attribuer une place qui correspond à son type de bateau, sachant que les voiliers de types 2 ne peuvent aller que sur des **grandes places** qui sont en nombre limité. Il existe également des places, appelées **corps mort**, loin des quais, sans services, qui sont à prix réduit par rapport à une place à quai. Les **places normales** peuvent donc accueillir des voiliers non habitables ou des voiliers de type 1 à quais.

Toutes ces données nous permettent finalement de monter une facture pour chaque usager et d'avoir par la suite une trace de la comptabilité. La facture va différer selon les données sélectionnées. Un abonné va payer 500€ / an, tandis qu'un visiteur va payer 20€ / jour. Le type de bateau va aussi influencer la facture, en effet un bateau de type 1 se verra appliquer une majoration de 30%, un voilier de type de 2 une majoration de 60%. 5€ en plus sera demandé par service ajouté. Le prix d'une nuit sur un corps mort est réduit de 50 % par rapport à la place la moins cher à quai.

Conception

1. Environnement choisi.

Notre programme visant à répondre à un problème posé, à savoir gérer les vas et vient dans le port de La Rochelle de manière informatisée, nous avons réalisé celui-ci avec le langage C++. Ce langage descend du langage C mais propose de nouvelles fonctionnalités comme la programmation orientée objet (POO). Cela fait du C++ un langage puissant permettant de programmer en définissant des objets, qui possèdent différentes caractéristiques, et qui vont interagir ensemble pour créer le programme. L'avantage de programmer en POO est d'avoir un programme plus modulable et donc de pouvoir évoluer facilement au cours du temps et du changement des demandes du client.

L'intégralité du programme a été réalisé avec CLion qui est un IDE spécialisé pour le C et le C++ (venant de la suite de logiciel réalisé par JetBrains). Nous avons choisi cet environnement de développement car nous étions déjà familier avec celui ci et il nous permet d'éditer notre code mais aussi de le compiler de manière simplifiée. Le travail étant réalisé en binôme nous avons utilisé un dépot sur Github afin d'avoir des sauvegardes des différentes versions de notre code, mais surtout de pouvoir se partager le code de manière simple et efficace en gérant les conflits qui peuvent survenir quand plusieurs personnes travaillent sur le même code.

Pour réaliser la sauvegarde de données, nous avons décidé de garder celles-ci sous un fichier XML. Le XML est un langage qui utilise des balises pour que la machine puisse lire les données Ces balises permettent de structurer de manière hiérarchisée et organisée les données d'un document. Pour que notre programme C++ ait accès à ce fichier et puisse l'analyser nous avons utilisé TinyXML-2. Le TinyXML-2 analyse le document XML et construit à partir de cela un modèle d'objet de document qui peut être lu, modifié et enregistré.

2. Choix des classes.

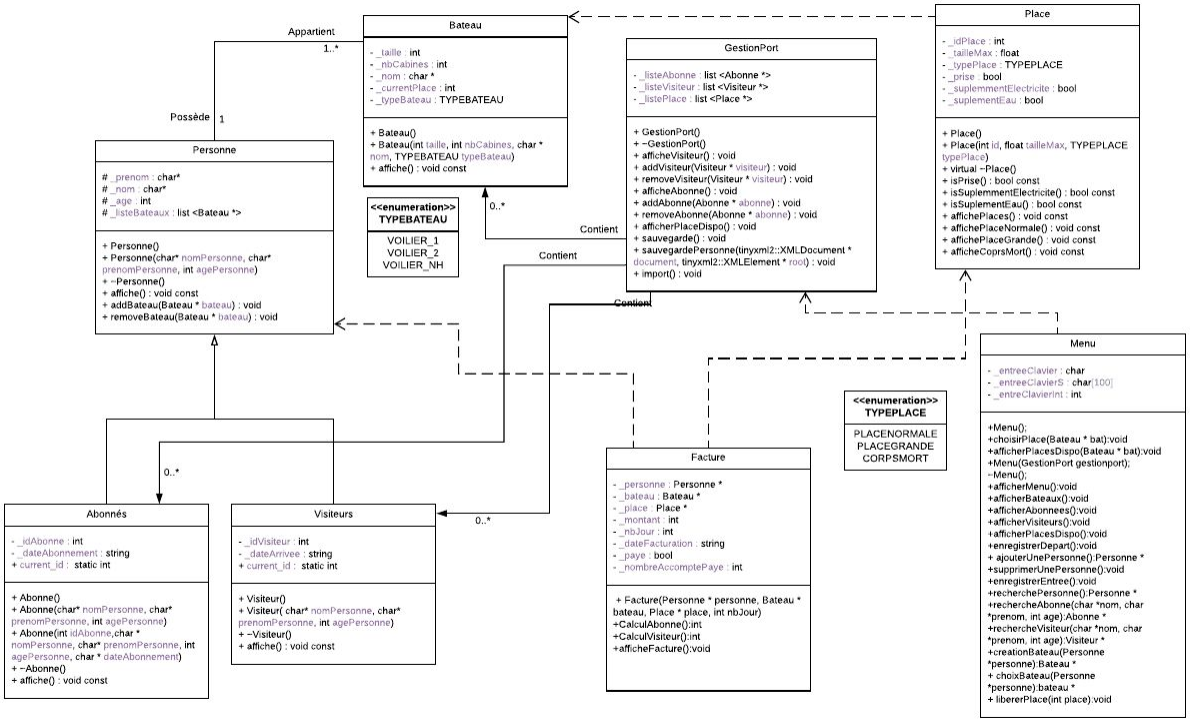
Afin de réaliser un programme qui se rapproche au plus près de ce que le client nous a demandé, nous avons réalisé au préalable une liste des classes utiles pour répondre

aux différentes fonctionnalitées demandées et également un diagramme de classe qui permet de représenter les liaisons se trouvant entre les différentes classes. Par soucis de lisibilité nous avons décidé de ne pas faire apparaître les accesseurs dans notre diagramme.

→ Les classes abonnés et visiteurs héritent de la classe Personne afin d'éviter la duplication de code pour les fonctions qui se gèrent de la même manière pour un abonné ou un visiteur (comme l'ajout d'un bateau) ou bien les attributs communs (le nom ou prénom par exemple).
→ La classe Bateau va permettre de récupérer la liste des bateaux présents dans le port, ou possédé par un usager, en connaissant le type du Bateau (grâce à une énumération), sa taille, son nom et également la place à laquel il a été attribué.
→ La classe place nous permet de gérer plusieurs choses. Comme cette classe est reliée une énumération TYPEPLACE on va pouvoir choisir le type de la place (Corps mort, normale ou grande). On va également pouvoir gérer si la place est prise ou non grâce à un booléen, valider s'il y a ou non un supplément pour les services, et afficher une liste de place depuis une autre classe (disponibles ou non).
→ Nous avons décidé de créer une classe Menu afin de gérer dans un même objet les fonctions qui seront appelées pour créer l'interface (qui est du texte dans le terminal). Ce menu rassemble la possibilité :

- d'afficher différentes listes comme celle des abonnées ou des visiteurs par exemple
- d'afficher les places disponibles dans le port
- d'ajouter une personne (visiteur ou abonné) dans le port en entrant ses informations (sur sa personne, sur son bateau et sur le type de service qu'il aimerait avoir)
- de supprimer une personne du registre.

→ La classe GestionPort permet d'avoir un objet qui rassemble toutes les informations du port et les conserve. Elle nous permet donc d'avoir la liste des visiteurs, des abonnés, des places ou encore des bateaux présents au port. Nous pouvons grâce à cette classe modifier ces liste en y ajoutant ou en y supprimant des éléments. Ces éléments sont sauvegardés et importés dans un fichier XML.
→ La classe facture permet de calculer la facture d'une personne en fonction de son statut (abonné ou visiteur), de son bateau, de sa place et du nombre de jours sur place. On peut ensuite grâce à la construction de cette facture l'afficher en prenant en compte le fait qu'elle ait été payé ou non.



Vous trouverez ci-dessous un lien pour voir le diagramme de classe plus en détail (afin de pouvoir zoomer) :
https://www.lucidchart.com/invitations/accept/622ff913-494b-4105-bdae-327b9859b434

3. Choix de conception.

En réalisant le choix de nos classes pour répondre au mieux à la demande, nous avons fait certains choix de conception, que ce soit pour faciliter la réalisation du code ou bien par soucis d'interprétation de la demande.

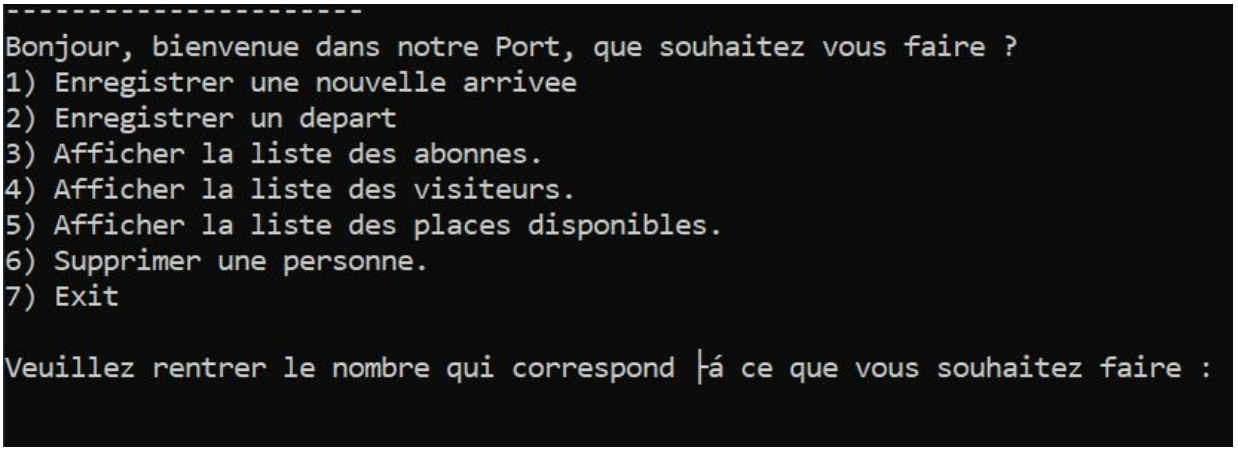
→ Les énumérations TYPEBATEAU et TYPEPLACE plutôt que de l'héritage : cela permet simplement de pouvoir récupérer le type des places ou des bateaux facilement grâce à des getters et donc faciliter leur gestion en fonction de leur type.
→ Le placement des bateaux : pour le placement des bateaux nous avons fait quelques choix de conception. Les voiliers non habitables auront la possibilité de se mettre soit sur un corps mort soit sur place à quai (même s'ils ne pourront pas choisir de services). Les voiliers de type 1 auront une place à quai s'ils ont choisi de prendre

un ou plusieurs services , sinon ils pourront choisir entre une place à quai et un corps mort. Ensuite nous avons choisi de réserver les grandes places uniquement au voiliers de type 2, car elles sont en nombre limité et ce type de voilier ne peut accoster que sur ce type de place.
→ Les services font partis de la classe Place et sont modélisés sous forme de booléens. Les booléens seront mis à faux par défaut (si la place est de type Corps Mort les services resteront à faux), autrement ils seront ajustés à vrai ou faux en fonction du choix d'utilisation ou non de ceux ci par la personne réservant une place. Nous sommes donc parti du principe que toute place à quais ou grande place est capable d'avoir de l'eau et de l'électricité si l'utilisateur le désire.
→ Afin de conserver une trace de tous les usagers ayant payé au moins un jour, pour des raisons comptables, nous pouvons depuis le menu déclarer le départ du port d'une personne et ainsi le garder dans nos fichiers de sauvegarde.
→ Nous avons fait le choix de laisser la possibilité à un abonné d'avoir plusieurs bateaux, ainsi s'il change de bateau un jour, nous aurons un historique de tous les bateaux ayant séjourné au port de La Rochelle.
→ Plusieurs fois dans notre code nous avons fait appel à la bibliothèque STL en utilisant des listes de pointeurs. Cette bibliothèque simplifie grandement l'utilisation d'une liste. Ainsi nous pouvons avoir accès grâce à des fonctions primaires au premier élément de la liste par exemple.

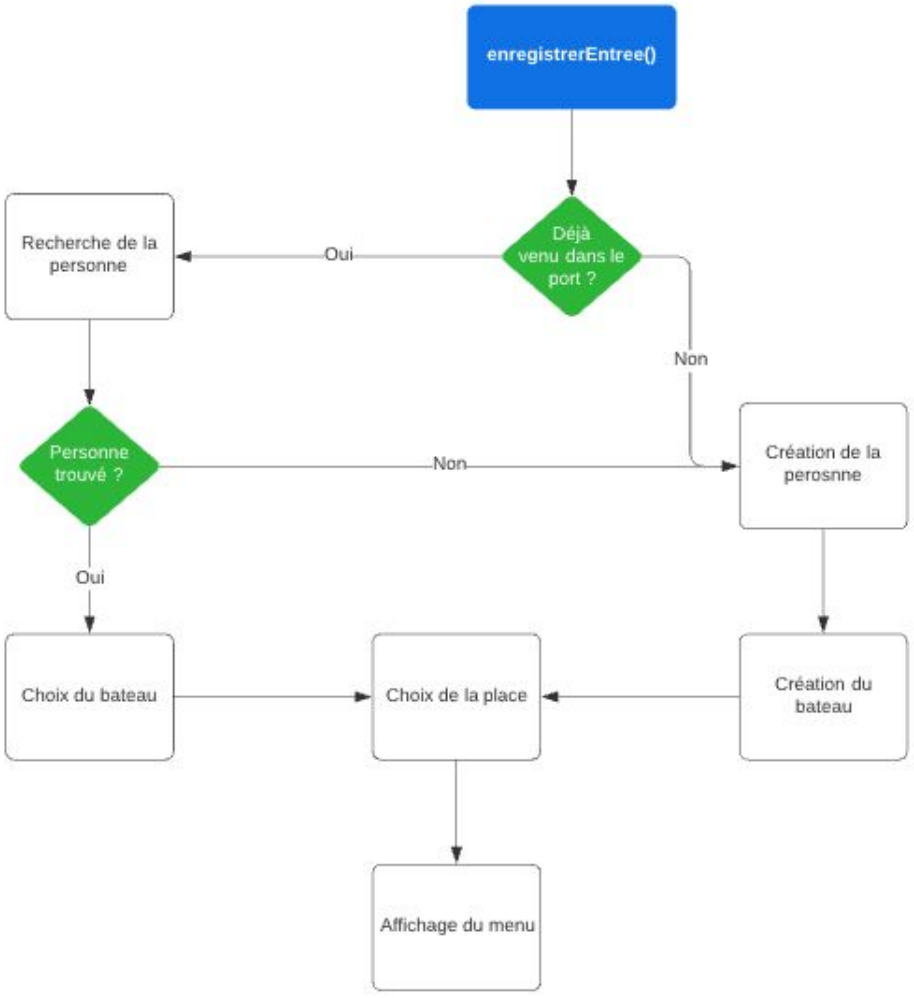
Réalisation

1. Le menu de navigation

Comme demandé, la navigation se fait via des saisies de l'utilisateur sous forme de chiffre comme l'illustre l'image suivante :



Chaque fonction permet de faire une action précise sur la base de données. Cela peut aller à l'enregistrement d'une arrivée dans le port à la simple visualisation des abonnées et des visiteurs. Lors du choix d'une action, plusieurs questions sont posées afin de collecter toutes les informations nécessaire au bon fonctionnement de l'application.



Nous avons choisi de garder l'historique des passages dans le port. C'est pourquoi, une fonction de recherche existe. Elle nous permet de rechercher un abonné ou un visiteur avec son nom, prénom, âge. Elle nous permet de ne pas créer un visiteur en double lors de son deuxième passage par exemple. De plus, chaque personne a une liste de bateaux car il est possible d'avoir plusieurs bateaux par personne. Cette possibilité nous a obligé de créer une fonction permettant de choisir le bateau concerné par l'action effectué.

Pour enregistrer un départ du port, on utilise de nouveau la fonction rechercher pour avoir la personne et son bateau et on libère la place. On génère alors une facture sur la durée total du séjour.

2. L’import et la sauvegarde

Nous avons choisi d’utiliser le format XML qui permet de stocker facilement des informations simples. Ce format utilise des balises. Dans notre projet les balises sont des abonnés, des visiteurs, des bateaux, ou encore des places. Chaque attribut sont stockés dans les balises.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<App>
    <ListePlaces currentID="18">
        <place idPlace="1" tailleMax="36" typePlace="0" prise="true" suppElec="true" suppEau="true"/>
        <place idPlace="2" tailleMax="23" typePlace="2" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="3" tailleMax="45" typePlace="1" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="4" tailleMax="9" typePlace="0" prise="true" suppElec="false" suppEau="false"/>
        <place idPlace="5" tailleMax="14" typePlace="2" prise="true" suppElec="false" suppEau="false"/>
        <place idPlace="6" tailleMax="10" typePlace="1" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="7" tailleMax="36" typePlace="0" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="8" tailleMax="23" typePlace="2" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="9" tailleMax="45" typePlace="1" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="10" tailleMax="33" typePlace="0" prise="true" suppElec="false" suppEau="false"/>
        <place idPlace="11" tailleMax="14" typePlace="2" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="12" tailleMax="10" typePlace="1" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="13" tailleMax="36" typePlace="0" prise="true" suppElec="false" suppEau="false"/>
        <place idPlace="14" tailleMax="23" typePlace="2" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="15" tailleMax="45" typePlace="1" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="16" tailleMax="9" typePlace="0" prise="false" suppElec="false" suppEau="false"/>
        <place idPlace="17" tailleMax="25" typePlace="2" prise="true" suppElec="false" suppEau="false"/>
    </ListePlaces>
    <ListeAbonnes currentID="3">
        <abonne prenom="dawen" nom="calippe" age="23" idAbonne="0" dateAbonnement="Mon Apr 13 17:29:01 2020">
            <Bateau nom="Vaillant" taille="9" nombreCabine="0" typeBateau="2" currentPlace="5"/>
            <Bateau nom="temp" taille="9" nombreCabine="0" typeBateau="2" currentPlace="4"/>
        </abonne>
        <abonne prenom="jean" nom="dupont" age="23" idAbonne="1" dateAbonnement="Mon Apr 13 17:29:01 2020">
            <Bateau nom="bateau" taille="23" nombreCabine="4" typeBateau="2" currentPlace="17"/>
        </abonne>
        <abonne prenom="aurore" nom="dupuie" age="23" idAbonne="2" dateAbonnement="Mon Apr 13 17:29:01 2020">
            <Bateau nom="navire" taille="9" nombreCabine="0" typeBateau="2" currentPlace="1"/>
        </abonne>
    </ListeAbonnes>
    <ListeVisiteurs currentID="3">
        <visiteur prenom="Dawen" nom="Calippe" age="23" idVisiteur="0" dateArrivee="Mon Apr 13 21:45:28 2020
">
            <Bateau nom="bateauDawen" taille="25" nombreCabine="5" typeBateau="0" currentPlace="10"/>
        </visiteur>
        <visiteur prenom="aurore" nom="Calippe" age="23" idVisiteur="1" dateArrivee="Mon Apr 13 21:45:28 2020
">
            <Bateau nom="bateauAurore" taille="25" nombreCabine="5" typeBateau="0" currentPlace="13"/>
        </visiteur>
    </ListeVisiteurs>
</App>
```

CurrentId nous permet de savoir à quel Integer nous nous sommes arrêté et permet d’associer automatiquement un nouvelle id à la création d’un objet. Le fichier sauvegarde.xml permet donc de stocker toutes les données lors de la fermeture de l’application. C’est ce même fichier qui est appelé à l’ouverture de l'application pour l’importation des données.

Conclusion

Ce projet était un projet complet afin d'appliquer l'ensemble des connaissances vues en cours et en TD. De plus en nous permettant de travailler en binôme ce travail nous a poussé à nous organiser pour nous répartir le travail sans se gêner mutuellement. Le projet était complet et rassemblait plusieurs problématiques concrètes que l'on trouve en POO (utilisation de l'héritage etc) ou bien avec le langage C++ (problèmes de gestion de mémoire par exemple).

Pour faire le bilan de notre projet, nous pouvons dire que nous avons cherché à répondre au maximum aux attentes du client, cependant nous pouvons imaginer une version plus évoluée de cette application avec une meilleure interface graphique que celle du terminal et également une meilleure gestion des inputs (avec des exceptions pour chaque mauvais cas de figure).