Diagramme de classes

Pour la gestion de notre projet, nous avons créé un diagramme de classe. Celui-ci nous sert à répertorier toutes les classes présentes ainsi que leurs interactions entre elles.

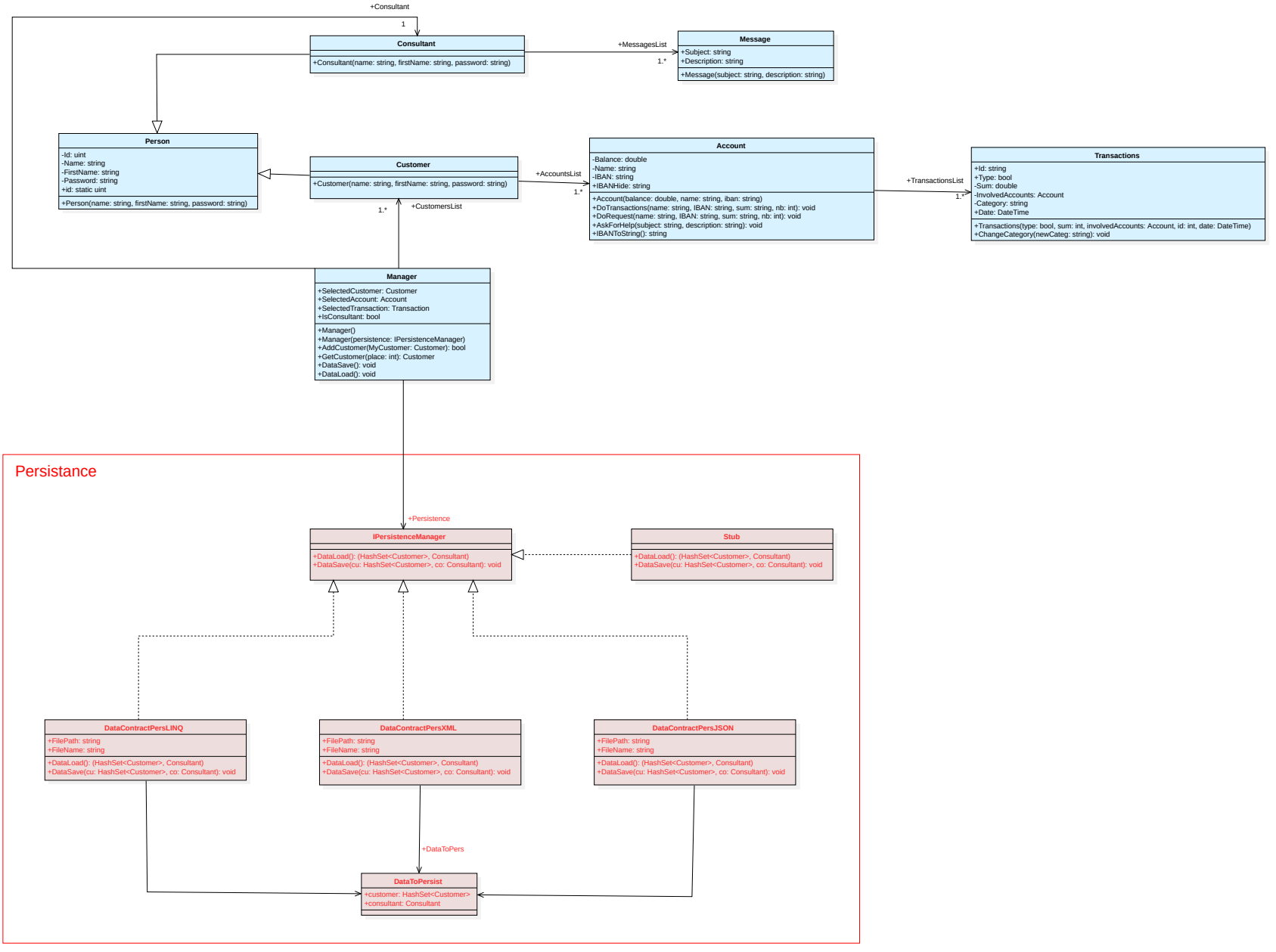
Nous avons donc créé un Manager qui va nous permettre de gérer une grande partie de l'application. En effet, il possède la liste des clients afin de gérer la connexion. Il peut ajouter un client ainsi que gérer la sauvegarde des données. Pour cela, il possède un attribut de type IPersistenceManager qui aura un type particulier (DataContractPersLINQ, DataContractPersJSON ou DataContractPersXML). Le manager possède aussi les attributs SelectedTransaction, SelectedAccount et SelectedCustomer afin de faire le DataBinding. Il permet de faire le lien entre toutes les classes et les vues lors de l'exécution de l'application.

Nous avons donc créé une interface IPersistenceManager, qui possède uniquement les méthodes DataLoad et DataSave. Celle-ci permet de gérer de différentes manières la persistance. Nous avons donc une classe DataContractPersLINQ pour écrire dans des fichiers XML avec LINQ, une classe DataContractPersXML pour écrire et lire les données dans un fichier avec XML, et une classe DataContractPersJSON pour lire et écrire les données dans un fichier en JSON. Ces trois classes utilisent la classe DataToPersist pour définir les données qui seront écrites et lues dans les différents fichiers et implémentent l'interface IPersistenceManager.

Nous avons aussi un Stub, qui permet d'avoir des données au tout premier lancement de l'application, c'est-à-dire quand les fichiers ne sont pas encore écrits.

Pour gérer les différents comptes, Client (Customer) et Conseiller (Consultant), nous avons créé une classe Person qui possède les attributs que les deux comptes auront en commun. Ils auront en effet un ID, un mot de passe, un nom et un prénom afin de pouvoir les différencier et les connecter. Les classes Consultant et Customer hériteront donc de la classe Person.
La classe "Person" possède aussi un attribut ID qui est statique. Cet attribut sert à définir l'ID des personnes quand les comptes seront créés. Nous ajoutons à chaque fois 1 à cet attribut afin de posséder l'ID des prochains clients. Un client possède une liste de comptes, dans lesquels sera présent, pour chacun d'entre eux, une liste de transactions, un solde, un nom et un IBAN. Cela permet aux différents utilisateurs d'accéder à plusieurs comptes avec un seul identifiant.

Une transaction possède un ID, un type de type booléen (True pour un virement, False pour la réception d'un virement). Chacun des deux comptes (émetteur et récepteur du virement) reçoit une transaction avec une catégorie et une date. La classe transaction possède la méthode "ChangeCategory" car un utilisateur pourra décider de changer la catégorie d'une transaction quand il sera sur la page de celle-ci.


+Consultant
1
Consultant
+Consultant(name: string, firstName: string, password: string)
+MessagesList
1.*
Message
+Subject: string
+Description: string
+Message(subject: string, description: string)
Person
-Id: uint
-Name: string
-FirstName: string
-Password: string
+id: static uint
+Person(name: string, firstName: string, password: string)
Customer
+Customer(name: string, firstName: string, password: string)
+AccountsList
1.*
Account
-Balance: double
-Name: string
-IBAN: string
+IBANHide: string
+Account(balance: double, name: string, iban: string)
+DoTransactions(name: string, IBAN: string, sum: string, nb: int): void
+DoRequest(name: string, IBAN: string, sum: string, nb: int): void
+AskForHelp(subject: string, description: string): void
+IBANToString(): string
+TransactionsList
1.*
Transactions
+Id: string
+Type: bool
-Sum: double
-InvolvedAccounts: Account
-Category: string
+Date: DateTime
+Transactions(type: bool, sum: int, involvedAccounts: Account, id: int, date: DateTime)
+ChangeCategory(newCateg: string): void
1.*
+CustomersList
Manager
+SelectedCustomer: Customer
+SelectedAccount: Account
+SelectedTransaction: Transaction
+IsConsultant: bool
+Manager()
+Manager(persistence: IPersistenceManager)
+AddCustomer(MyCustomer: Customer): bool
+GetCustomer(place: int): Customer
+DataSave(): void
+DataLoad(): void
Persistance
+Persistence
IPersistenceManager
+DataLoad(): (HashSet<Customer>, Consultant)
+DataSave(cu: HashSet<Customer>, co: Consultant): void
Stub
+DataLoad(): (HashSet<Customer>, Consultant)
+DataSave(cu: HashSet<Customer>, co: Consultant): void
DataContractPersLINQ
+FilePath: string
+FileName: string
+DataLoad(): (HashSet<Customer>, Consultant)
+DataSave(cu: HashSet<Customer>, co: Consultant): void
DataContractPersXML
+FilePath: string
+FileName: string
+DataLoad(): (HashSet<Customer>, Consultant)
+DataSave(cu: HashSet<Customer>, co: Consultant): void
DataContractPersJSON
+FilePath: string
+FileName: string
+DataLoad(): (HashSet<Customer>, Consultant)
+DataSave(cu: HashSet<Customer>, co: Consultant): void
+DataToPers
DataToPersist
+customer: HashSet<Customer>
+consultant: Consultant