Rapport modélisation UML initiale

Membres de l’équipe et rôles (Equipe F) :

-Beurel Simon (Ops)

-Maurois Quentin (Product Owner)

-Aziki Tarik (Software Architect)

-Froment Lorenzo (Quality Assurance Engineer)

Nom de l’équipe :

UberTech

Les hypothèses :

- Toutes les livraisons se passent correctement
  - Nous sommes partis du principe que la livraison s'effectue sans problèmes. Le livreur livre la commande au bon endroit, dans un délai raisonnable, en étant courtois et sans avoir endommagé la commande. Grâce à cette hypothèse, nous enlevons la possibilité au client de signaler un livreur.

- Avis sur le restaurant/livreur
  - Nous avons choisi de ne pas traiter le moment où l'utilisateur donne son avis et/ou note un restaurant ainsi qu'un livreur. Nous partons du principe que l'action est effectuée en « arrière-plan » ce qui permet donc aux administrateurs du campus de voir des statistiques et prendre des décisions.

- Paiement seulement par carte
  - Pour le paiement nous avons fait le choix de le simplifier en partant du fait que les clients peuvent seulement payer à l'aide d'une carte bancaire ce qui nous permet de ne pas détailler le procédé et de seulement dire "procéder au paiement".

Les limites identifiées :

- Connexion des utilisateurs
  - Nous n'allons pas prendre en compte la vérification de la connexion des utilisateurs dans le système avec une BDD contenant potentiellement un email et un mot de passe pour un compte associé (cela pourrait notamment poser des problèmes de RGPD).
- Système de paiement simulé
  - Le système de paiement sera simulé ce qui nous permettra de plus facilement l'implémenter et contrôler les actions qui y sont liées dans l'optique de vérification de la qualité de l'application.
- Pas de système de traque des livreurs
  - Quand un livreur est attribué à une commande et que ce dernier est en cours de livraison, le client (customer) ne pourra pas suivre sur un GPS la position exacte du livreur.

Glossaire :

Order : C'est la commande, là où l'on peut voir les produits commandés par le client ainsi que le total à payer.

CampusAdministrator : Personne ayant plus de droits que quiconque sur l'application, dont le rôle est de gérer l'ajout et la suppression de Restaurant ainsi que des livreurs.

Customer : C'est une personne utilisant l'application et commandant des produits à un restaurant.

GroupOrder : C'est une Order comportant plusieurs clients, mais une seule adresse. Elle regroupe les commandes payées des différents clients y participant.

Order : C'est la commande passée par un client pour acheter des produits auprès d'un restaurant.

Confirmer une commande (côté restaurant) : Marquer sur l'application que tous les éléments d'une commande sont prêts à être livrés.

Consulter les commandes : Consulter les commandes passées par les clients.

Consulter son historique : Pour un client, voir les commandes qu'il a passées auparavant.

Contrôler les partenariats : Ajouter/supprimer des restaurants/livreurs.

Gestion des horaires : Mettre à jour les horaires auxquels un restaurant peut préparer une commande.

DeliveryPerson : Personne chargée d'apporter la commande du restaurant jusqu'à l'adresse indiquée par le client.

Order : C'est la commande, là où l'on peut voir les produits commandés par le client ainsi que le total à payer.

pendingOrder : Endroit où sont stockés les produits sélectionnés par le client avant le paiement.

Passer une commande (côté customer): Création d'une "order" par un client, il commence à sélectionner des articles.

Product : Ce sont les plats composés par les restaurants.

Procéder au paiement : Concerne toutes les étapes de paiement qui ne sont pas détaillées comme expliqué dans les hypothèses.

Restaurant : Les établissements proposant des produits à la vente sur l'application.

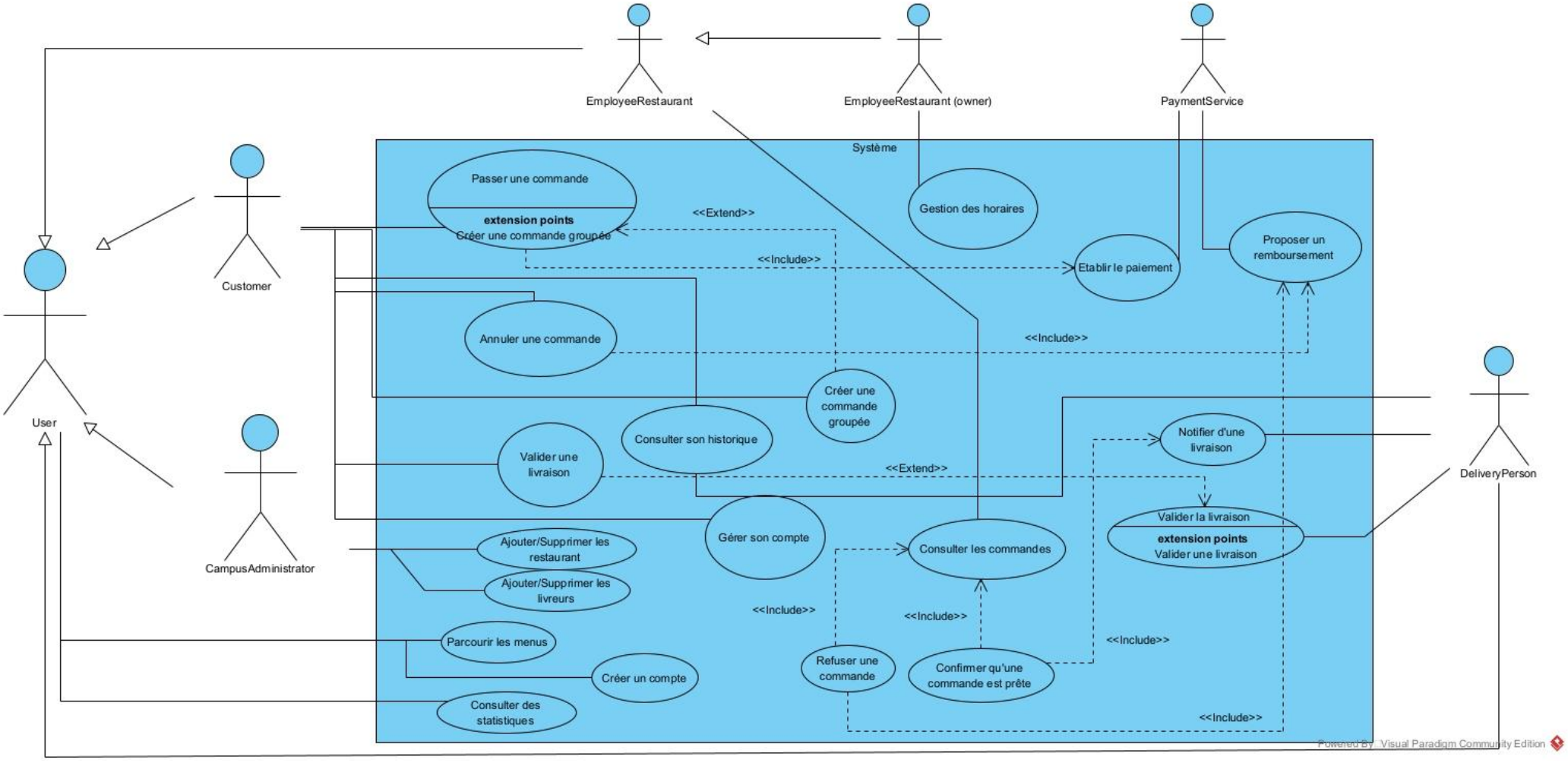
System : entité qui détient et gère les diverses données nécessaires et qui agit en tant que point d'accès pour les différents acteurs.

User : Acteur, utilisateur de l'application que ce soit pour commander, livrer, administrer ou bien préparer des commandes.

Valider une livraison (côté customer): Pour le client, c'est le fait de marquer sur l'application que la commande a été réceptionnée.

Valider une livraison (côté deliveryPerson) : Pour le livreur, c'est le fait de marquer sur l'application que la commande a été donnée au client.

RestaurantEmployee : Personnel qui travaille dans un restaurant. Cette personne peut être le propriétaire du restaurant ou alors un simple employé.


EmployeeRestaurant
EmployeeRestaurant (owner)
PaymentService
Système
Passer une commande
extension points
Créer une commande groupée
<<Extend>>
Gestion des horaires
<<Include>>
Etablir le paiement
Proposer un remboursement
Customer
Annuler une commande
<<Include>>
Créer une commande groupée
User
Consulter son historique
Notifier d'une livraison
Valider une livraison
<<Extend>>
DeliveryPerson
Valider la livraison
extension points
Valider une livraison
Gérer son compte
Consulter les commandes
CampusAdministrator
Ajouter/Supprimer les restaurant
Ajouter/Supprimer les livreurs
<<Include>>
<<Include>>
<<Include>>
Parcourir les menus
Refuser une commande
Confirmer qu'une commande est prête
Créer un compte
Consulter des statistiques
<<Include>>
Powered By Visual Paradigm Community Edition

Compréhension du diagramme de classes :

Pour notre diagramme de classes, nous avons décidé de partir sur un modèle initial de 14 classes.

Lorsqu'un utilisateur sera présent sur un terminal, il sera connecté et son compte sera un objet d'une classe Customer, RestaurantEmployee, DeliveryPerson ou CampusAdministrator. Ces 4 classes seront celles qui représenteront les différents utilisateurs qui utiliseront notre application (cf Glossaire). Nous allons vous expliquer en détail ces 4 classes ici :

- Customer :

  Cette classe représente un client voulant commander une commande sur l'application. Elle possède en attribut différentes informations sur le client comme la localisation, le nom, le prénom etc.. Cette classe possède également l'historique de commandes réalisées par le client, un moyen de paiement et également un tableau de produits qui correspond au panier en cours de réalisation, que le client peut valider puis payer pour pouvoir transformer ce panier (ie. pendingOrder) en commande.

- RestaurantEmployee :

  Cette classe représente une personne travaillant dans un restaurant. Cette personne peut être un simple employé ou alors le propriétaire du restaurant (cela est représenté par l'attribut isOwner). Un RestaurantEmployee possède comme attribut le restaurant dans lequel il travaille. Cette classe sera chargée de traiter les différentes commandes qui arrivent dans un restaurant en changeant leur état pour valider la commande, l'annuler etc..

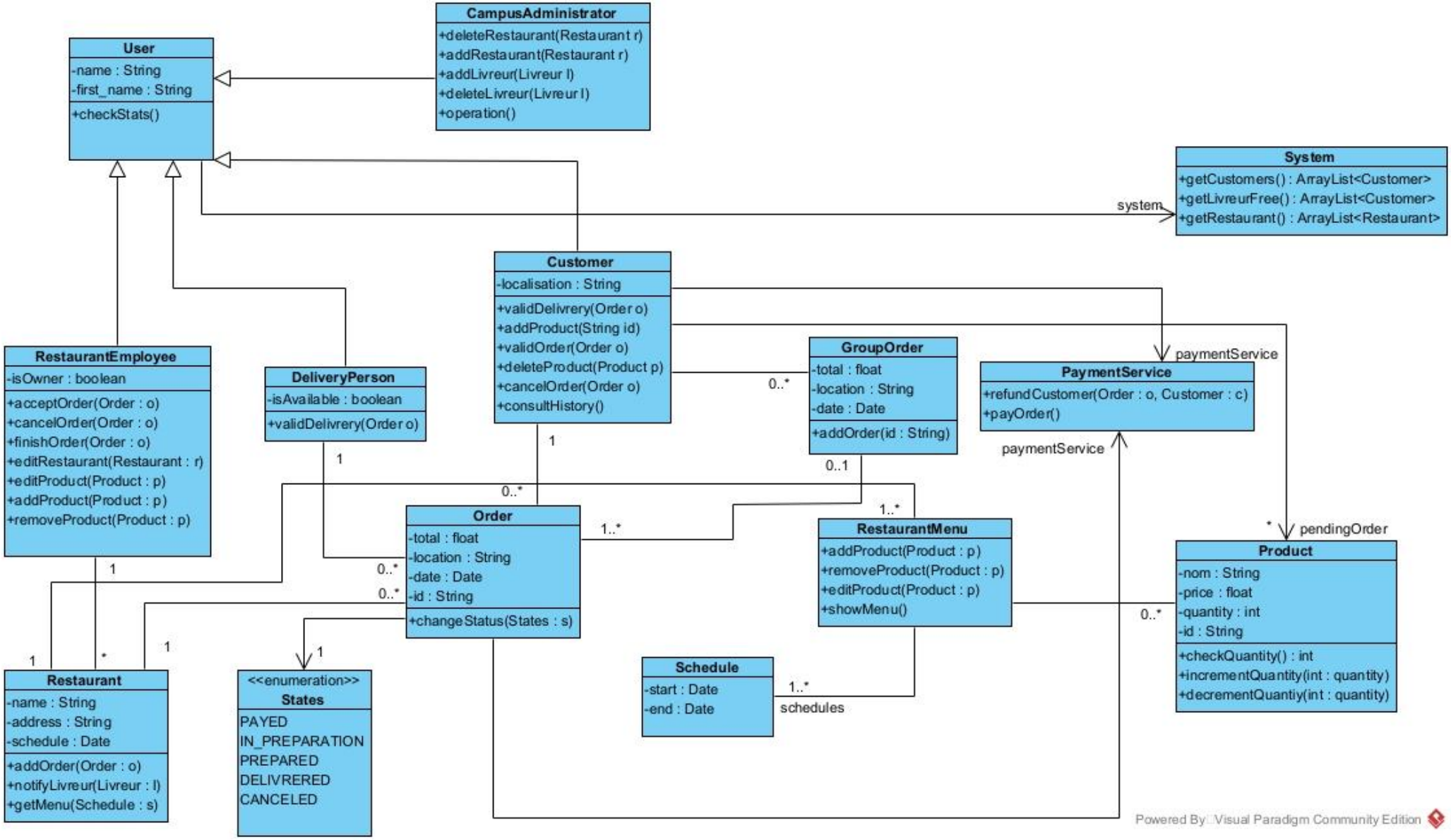
- DeliveryPerson :

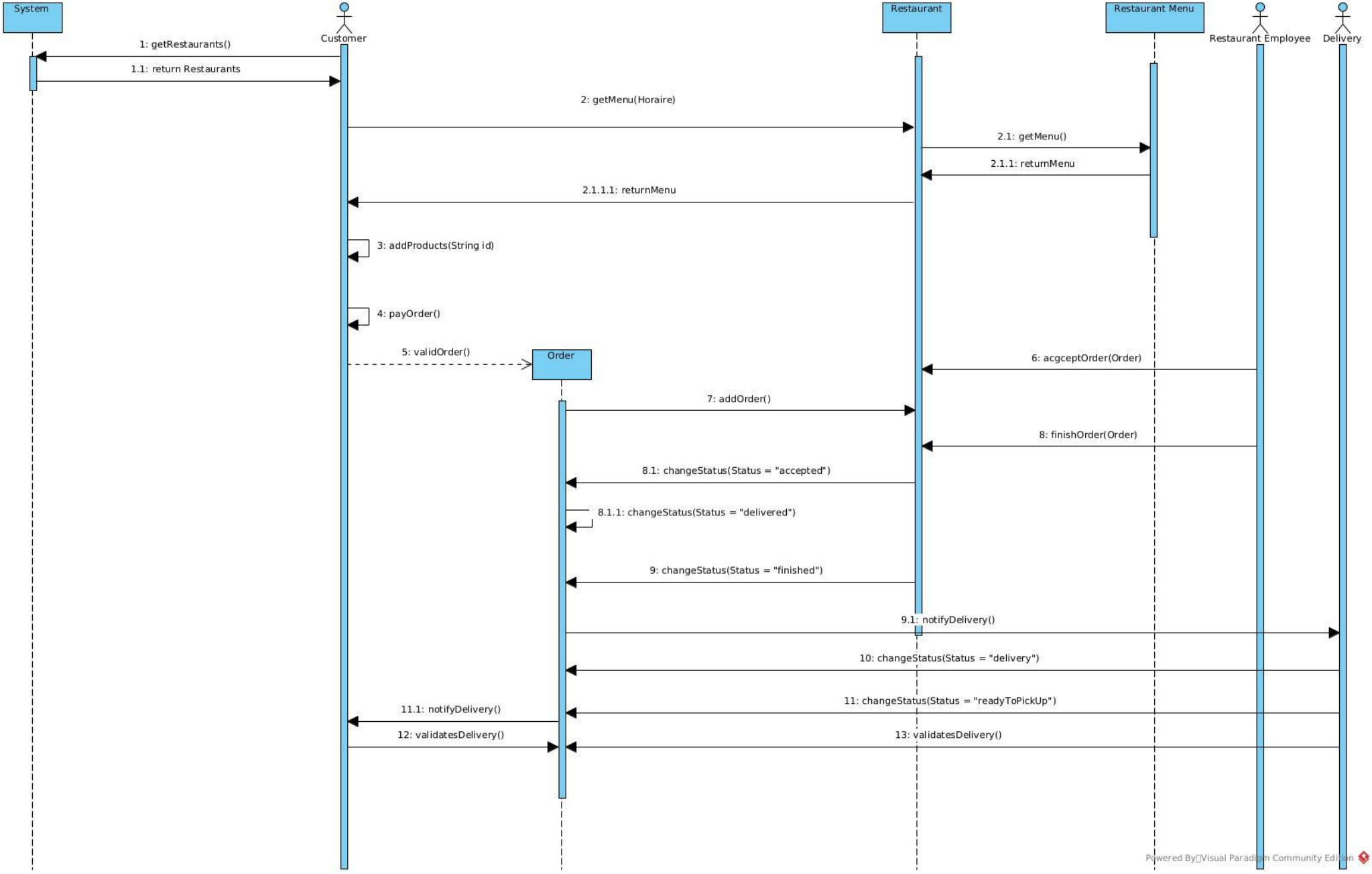
  Cette classe représente un livreur qui aura pour but de livrer une commande au lieu de livraison indiqué par le Customer. Il possède comme attribut son historique de commandes déjà réalisées dans le passé, ce qui lui permet de pouvoir consulter ses statistiques à tout moment.

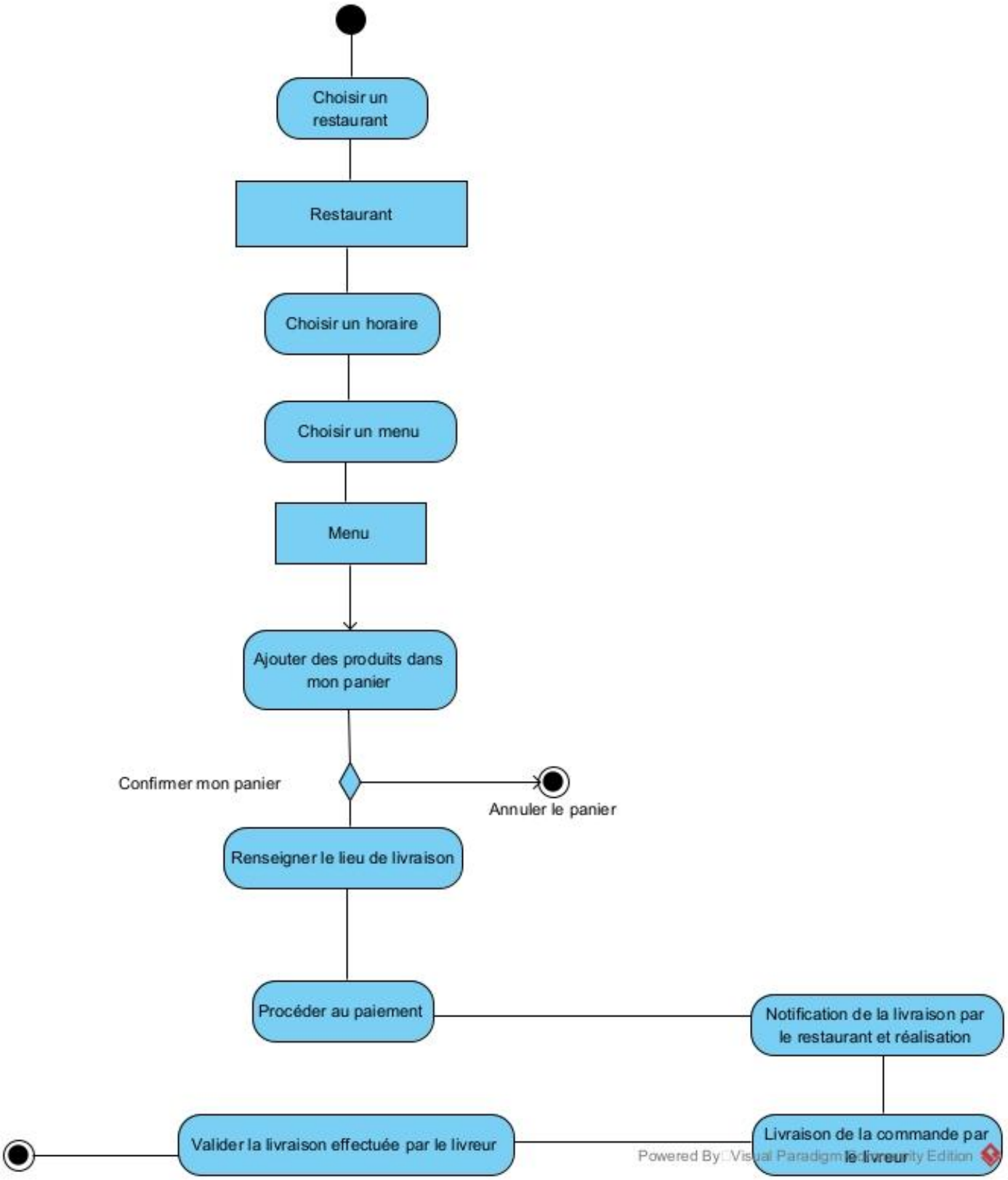
- CampusAdministrator :

  Cette classe représente les différentes administrateurs du campus, qui pourront gérer (ajouter/supprimer) les différents restaurants présents sur l'application ainsi que les différents livreurs.

Il est également important de noter que toutes ces classes possèdent comme attribut le System. Le System leur sert de point d'entrée pour toutes leurs opérations.


CampusAdministrator
+deleteRestaurant(Restaurant r)
+addRestaurant(Restaurant r)
+addLivreur(Livreur l)
+deleteLivreur(Livreur l)
+operation()
User
-name : String
-first_name : String
+checkStats()
System
+getCustomers() : ArrayList<Customer>
+getLivreurFree() : ArrayList<Customer>
+getRestaurant() : ArrayList<Restaurant>
system
Customer
-localisation : String
+validDelivrery(Order o)
+addProduct(String id)
+validOrder(Order o)
+deleteProduct(Product p)
+cancelOrder(Order o)
+consultHistory()
paymentService
RestaurantEmployee
-isOwner : boolean
+acceptOrder(Order : o)
+cancelOrder(Order : o)
+finishOrder(Order : o)
+editRestaurant(Restaurant : r)
+editProduct(Product : p)
+addProduct(Product : p)
+removeProduct(Product : p)
DeliveryPerson
-isAvailable : boolean
+validDelivrery(Order o)
GroupOrder
-total : float
-location : String
-date : Date
+addOrder(id : String)
PaymentService
+refundCustomer(Order : o, Customer : c)
+payOrder()
paymentService
Order
-total : float
-location : String
-date : Date
-id : String
+changeStatus(States : s)
RestaurantMenu
+addProduct(Product : p)
+removeProduct(Product : p)
+editProduct(Product : p)
+showMenu()
pendingOrder
Product
-nom : String
-price : float
-quantity : int
-id : String
+checkQuantity() : int
+incrementQuantity(int : quantity)
+decrementQuantiy(int : quantity)
Restaurant
-name : String
-address : String
-schedule : Date
+addOrder(Order : o)
+notifyLivreur(Livreur : l)
+getMenu(Schedule : s)
<<enumeration>>
States
PAYED
IN_PREPARATION
PREPARED
DELIVRERED
CANCELED
Schedule
-start : Date
-end : Date
schedules
0..*
1
1..*
0..1
Powered By Visual Paradigm Community Edition


System
Customer
Restaurant
Restaurant Menu
Restaurant Employee
Delivery
1: getRestaurants()
1.1: return Restaurants
2: getMenu(Horaire)
2.1: getMenu()
2.1.1: returnMenu
2.1.1.1: returnMenu
3: addProducts(String id)
4: payOrder()
5: validOrder()
Order
6: acgceptOrder(Order)
7: addOrder()
8: finishOrder(Order)
8.1: changeStatus(Status = "accepted")
8.1.1: changeStatus(Status = "delivered")
9: changeStatus(Status = "finished")
9.1: notifyDelivery()
10: changeStatus(Status = "delivery")
11: changeStatus(Status = "readyToPickUp")
11.1: notifyDelivery()
12: validatesDelivery()
13: validatesDelivery()
Powered By Visual Paradigm Community Edition


Choisir un restaurant
Restaurant
Choisir un horaire
Choisir un menu
Menu
Ajouter des produits dans mon panier
Confirmer mon panier
Annuler le panier
Renseigner le lieu de livraison
Procéder au paiement
Notification de la livraison par le restaurant et réalisation
Livraison de la commande par le livreur
Valider la livraison effectuée par le livreur
Powered By: Visual Paradigm Community Edition