Présentation du contexte

L'organisation cliente : l'entreprise de location de véhicules XXX

XXX est spécialisée dans la location de véhicules haut de gamme et de luxe, avec chauffeur. La société propose des courses sur Paris et sa périphérie avec des trajets prédéfinis (d'un lieu de départ à un lieu de destination) comme, par exemple, Paris-Versailles, Paris-Deauville, Paris-Fontainebleau, Paris-Chantilly.

Dans une démarche qualité, XXX veille à respecter les engagements suivants :

- proposer une réservation rapide et simple 24/24 et 7j/7, par téléphone ou sur le site ;
- offrir la possibilité de réserver un véhicule pour un départ immédiat ou pour une date et une heure précise ;
- équiper les véhicules d'une tablette avec une connexion *Wi-Fi* et de chargeurs pour téléphones intelligents (*smartphones)* ;
- fournir des chauffeurs professionnels en adéquation avec la qualité du service ;
- avertir la personne cliente, dans un délai respectable, en cas d'annulation et procéder à tout remboursement, le cas échéant ;
- veiller au respect des règles de sécurité en vigueur ;
- protéger les données bancaires et personnelles et ne jamais les divulguer à un tiers.

Pour accompagner son activité, XXX dispose :

- d'un site *«www.XXXhubert.com»* qui permet la consultation de l'offre et la réservation d'un véhicule avec chauffeur ;
- d'une application «*GestActivité*», écrite en C#, qui permet aux salariés de XXX de suivre l'activité de l'entreprise. Elle propose notamment les modules "consultation des caractéristiques des locations", "gestion du parc automobile" et "statistiques sur les locations" ;
- d'une base de données qui héberge toutes les données nécessaires à ces deux applications ;
- des applications mobiles développées par deux grands acteurs de ce marché, Uber et Chauffeur Privé, à qui XXX paie des frais de service.

L'entreprise prestataire de services

DevApp, entreprise de services du numérique (ESN) située en région parisienne, a développé pour XXX, le site et l'application «*GestActivité*».

DevApp s'occupe, par contrat, de la maintenance corrective de cet ensemble.

Le projet

L'offre XXX évolue. L'entreprise a décidé de se lancer dans la location de voiture sans chauffeur ce qui implique des modifications de la base de données et des applications (site *«www.XXXhubert.com»* et application «*GestActivité*»). XXX confie ces évolutions à DevApp.
Par ailleurs, XXX s'interroge sur le développement de sa propre application mobile et demande l'assistance de DevApp dans cette réflexion.

Vous devez intervenir dans l'équipe DevApp qui est affectée au nouveau projet confié par XXX.

Présentation de la nouvelle activité "location sans chauffeur"

- **Les formules sans chauffeur** : plusieurs formules de location sans chauffeur seront proposées à la clientèle. Une formule donne droit à une durée de location définie (4 heures, 24 heures, 48 heures) et un nombre de kilomètres inclus forfaitairement. Tout kilomètre supplémentaire effectué par le client sera facturé à un tarif qui dépend du modèle du véhicule loué.

- **Réservation d'une location sans chauffeur** : elle sera réalisée via le site *«www.XXXhubert.com».* Le formulaire de réservation proposera à la clientèle de choisir la formule et le modèle de véhicule

souhaités. Un véhicule, correspondant au modèle choisi par la personne cliente, sera affecté automatiquement.
Lorsque la personne cliente aura fourni les renseignements nécessaires au dossier de location, elle devra procéder au règlement.

- **Retrait du véhicule loué :** lors du retrait du véhicule, un dépôt de garantie sera demandé dont le montant dépend du modèle du véhicule et la personne cliente devra signer un document indiquant qu'elle a pris connaissance de l'état du véhicule loué. Elle pourra souscrire une assurance lui offrant des garanties supplémentaires en cas de dommages sur le véhicule.

- **Restitution du véhicule loué :** lorsque la personne cliente restituera le véhicule loué, XXX contrôlera le véhicule pour savoir s'il a ou non subi des dommages.
Par ailleurs, le nombre de kilomètres indiqué au compteur sera relevé et comparé au nombre de kilomètres inclus dans la formule : si le forfait kilométrique est dépassé, le montant dû au titre du dépassement devra être réglé.

Pour l'instant, l'équipe DevApp a modifié :

- la base de données afin qu'elle intègre les données indispensables à la consultation et la réservation de véhicule sans chauffeur *;*
- les deux fonctionnalités de consultation et de réservation, disponibles sur le site *«www.XXXhubert.com»,* qui sont en phase de test*.*

Évolution de la structure de la base de données

Dans le cadre de la nouvelle activité de location sans chauffeur, un module permettant de gérer les **processus de retrait et de restitution du véhicule** doit être ajouté dans l'application «*GestActivité*». Vous devez modifier la base de données pour permettre le développement de ce module. La structure de la base de données actuellement opérationnelle vous est fournie dans le dossier documentaire.

Le retrait du véhicule

Avant d'être confié pour une location sans chauffeur, le véhicule subit un contrôle de la part d'un salarié de XXX en présence de la personne cliente : différents éléments sont vérifiés et les dommages constatés sur le véhicule sont saisis sur tablette et consignés dans l'application «*GestActivité*». Pour chaque élément endommagé (aile avant gauche, calandre, etc.), on conservera le degré de gravité du dommage. Un rapport d'état du véhicule avant la location sera édité par XXX et signé par la personne cliente pour approbation.

Lors du retrait du véhicule, XXX propose à la personne cliente de souscrire une assurance. Cette souscription est facultative. En cas de vol ou de dommages sur le véhicule pendant la location, les garanties prises en charge et les montants de franchise dépendront donc de l'assurance souscrite.

Il y a plusieurs garanties possibles (vol, dommage, etc.). Pour chaque garantie, il existe une franchise de base. Une franchise est la somme maximum qui reste à la charge de la personne cliente, si celle-ci est déclarée responsable au regard de la garantie concernée (vol, dommage, etc.) ou s'il n'existe pas de recours contre un tiers identifié.

Chaque assurance porte un nom, une description et propose différentes garanties. Pour chaque garantie d'une assurance complémentaire, un taux de réduction compris entre 0 et 100% est appliqué sur la franchise de base en cas de dommage.

Voici trois exemples d'assurance :

- l'assurance «essentielle» propose
  - la garantie vol avec la franchise de base.
- L'assurance «confort» propose
  - la garantie vol avec une réduction de 80 % de la franchise de base,
  - la garantie dommage avec une réduction de 50 % de la franchise de base,
  - la garantie bris de glace avec la franchise de base.

- L'assurance «rachat de franchise de base» propose
  - la garantie vol avec une réduction de 100 % de la franchise de base,
  - la garantie dommage avec une réduction de 100 % de la franchise de base.

Lors du retrait du véhicule, la personne cliente doit verser un dépôt de garantie qui dépend du modèle de véhicule loué.
La restitution totale ou partielle de ce montant dépendra des dommages constatés lors de sa restitution et de l'assurance souscrite par la personne cliente.

La restitution du véhicule

L'employé de XXX qui réceptionnera le véhicule à la fin de la location sans chauffeur devra vérifier son état. L'application «*GestActivité*» sera utilisée pour pouvoir enregistrer les différents dommages et leur degré de gravité constatés lors de la restitution du véhicule. Un rapport d'état du véhicule après la location sera édité par XXX et signé par la personne cliente pour approbation.

La différence entre l'état avant la location et après la location sert à déterminer si des dommages sont à imputer à la location. Dans ce cas, l'employé qui réceptionne le véhicule détermine un coût estimatif des réparations en fonction d'une matrice de dommages établie par XXX et ce coût estimé est enregistré au niveau de la location. La gestion de la matrice des dommages sort du cadre de cette mission.

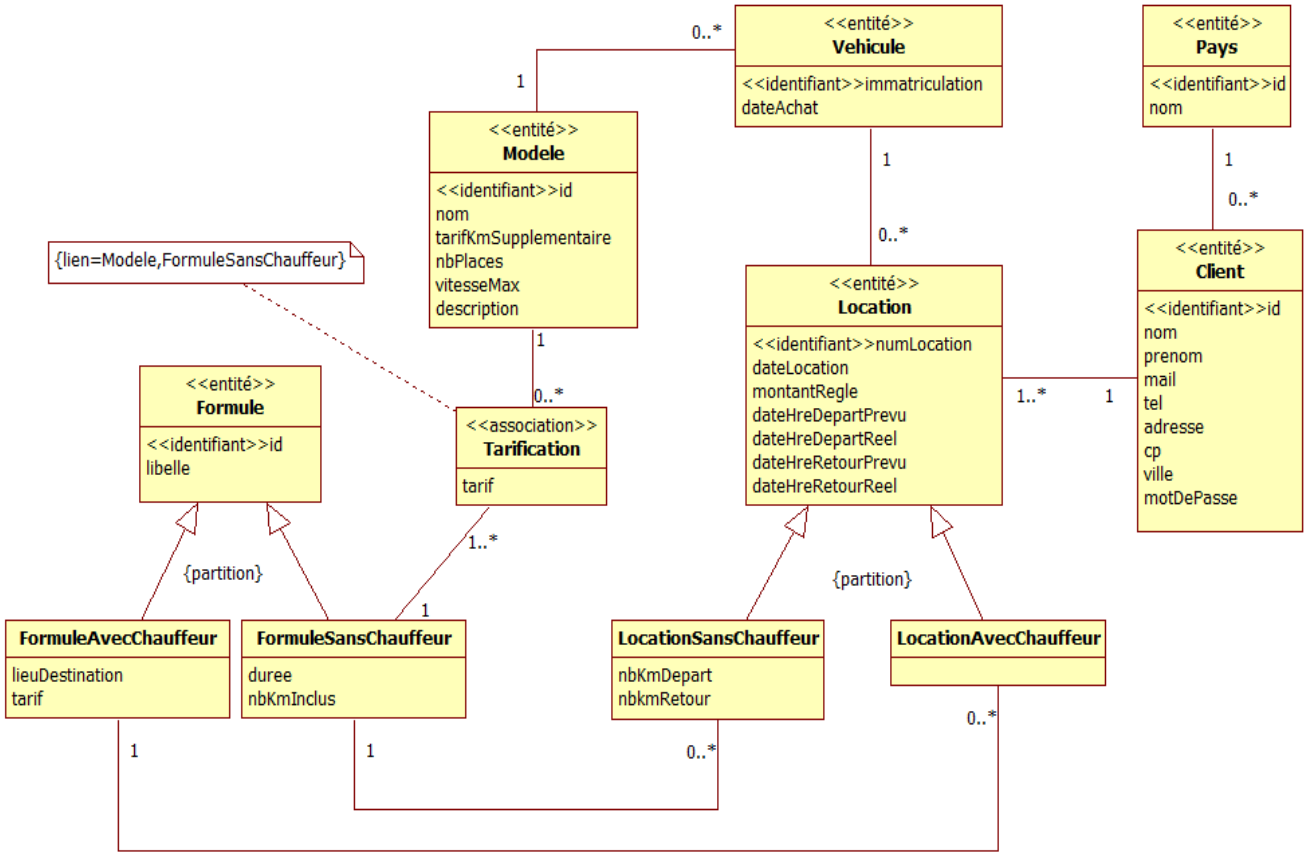
Mettre à jour le diagramme de classes en annexe, afin de tenir compte du nouveau besoin.

Annexe documentaire

Document 1 : Structure de la base de données

Modélisation conceptuelle

Diagramme de classes :



Document : Classes implémentées

***Remarque** : seuls les attributs et méthodes utiles sont représentés.*

```
class Modele
{
  // Attributs privés
        private  int id;
        private  string libelle;
        private  double tarifKmSupplementaire;  // représente le tarif du km supplémentaire

  // Constructeur de la classe Modèle
        public Modele(int unId, string unLibelle, double unTarifKmSupp)
        {
                this.id =  unId ;
                this.libelle = unLibelle;
                this.tarifKmSupplementaire = unTarifKmSupp ;
        }
  // Méthode
        public double GetTarifKmSupplementaire() { // retourne le tarif du km supplémentaire  }
}
```

```
class Vehicule
{
  // Attributs privés
        private  string immatriculation;
        private  DateTime dateAchat;
        private  Modele leModele;

  // Constructeur de la classe Vehicule
        public Vehicule(string uneImmat, DateTime uneDate, Modele unModele)
        {
                this.immatriculation = uneImmat ;
                this.dateAchat = uneDate;
                this.leModele = unModele ;
        }
  // Méthode
        public Modele GetLeModele()  { // retourne le modèle du véhicule }
}
```

```
abstract class Formule
{
  // Attributs privés
        private  int id ;
        private  string libelle ;

  // Constructeur de la classe Formule
        public Formule(int unId, string unLibelle)
        {
                this.id = unId ;
                this.libelle = unLibelle ;
        }
  // Méthode
        public string GetLibelle() { // retourne le libellé de la formule  }
}
```

```
class FormuleSansChauffeur : Formule      // hérite de Formule
{
  // Attributs privés
        private  int duree ;                          // durée de la location en heures
        private  double nbKmInclus ;        // forfait kilométrique

  // Constructeur de la classe FormuleSansChauffeur qui appelle le constructeur de la classe mère
  // grâce à la syntaxe :base(…)
        public FormuleSansChauffeur (int unId, string unLibelle, int uneDuree,
                                        double unNbKmInclus) : base(unId, unLibelle)
        {
                this.duree = uneDuree ;
                this.nbKmInclus = unNbKmInclus ;
        }

  // Méthodes
        public double GetNbKmInclus() { // retourne le forfait kilométrique  }
        public int GetDuree(){ // retourne la durée }
        public double GetTarif(Modele unModele) { // retourne le tarif de la formule sans chauffeur
                                                  // pour le modèle de véhicule passé en paramètre}
}
```

```
class FormuleAvecChauffeur : Formule        //  hérite de Formule
{
  // Attributs privés
        private string lieu ;
        private double tarif ;

  // Constructeur de la classe FormuleAvecChauffeur qui appelle le constructeur de la classe mère
  // grâce à la syntaxe :base(…)
        public FormuleAvecChauffeur (int unId, string unLibelle,
                                        string unLieu, double unTarif):base(unId, unLibelle)
        {
                this.lieu = unLieu ;
                this.tarif = unTarif ;
        }

  // Méthodes
        public string GetLieu()  { // retourne le lieu  }
        public double GetTarif()  { // retourne le tarif de la formule avec chauffeur }
}
```

```
abstract class Location
{
  // Attributs privés
        private  int  numLocation ;                  // numéro de la location
        private  DateTime dateLocation ;
        private  double montantRegle ;  // montant total réglé pour la location
        private  Vehicule  leVehicule ;

  // Constructeur de la classe Location
        public Location(int unNumLoc, DateTime uneDate, double unMontantRegle,
                        Vehicule unVehicule)
        {
                this.numLocation = unNumLoc;
                this.dateLocation = uneDate;
                this. montantRegle= unMontantRegle;
                this.leVehicule = unVehicule ;
```

```
        }

  // Méthodes
        public Vehicule GetLeVehicule() { // retourne le véhicule concerné par la location }
        public double GetMontantRegle () { // retourne le montant total réglé par le client }
}
```

```
class LocationAvecChauffeur : Location   // hérite de Location
{
  //Attribut privé
        private FormuleAvecChauffeur  laFormuleAvecChauffeur ;

  // Constructeur de la classe LocationAvecChauffeur qui appelle le constructeur de la classe mère
  // grâce à la syntaxe : base(…)
        public LocationAvecChauffeur(int unNumLoc, DateTime uneDate ,
                double unMontantRegle,Vehicule unVehicule,
                FormuleAvecChauffeur uneFormuleAvecChauffeur) : base (unNumLoc,
                uneDate, unMontantRegle,unVehicule)
        {
                this.laFormuleAvecChauffeur = uneFormuleAvecChauffeur ;
        }

}
```

```
class LocationSansChauffeur : Location        // hérite de Location
{
  //Attributs privés
        private double nbKmDepart; // kilométrage au compteur lors du retrait du véhicule
        private double nbKmRetour; // kilométrage au compteur lors de la restitution
        private FormuleSansChauffeur  laFormuleSansChauffeur;

  // Constructeur de la classe LocationSansChauffeur
        public LocationSansChauffeur(int unNumLoc, DateTime uneDate ,
                double unMontantRegle, double unNbKmDepart, Vehicule unVehicule,
        FormuleSansChauffeur uneFormuleSansChauffeur)
                //  …

// Méthodes
        // méthode qui permet de modifier le nombre de kms lu au compteur lors de la
        // restitution du véhicule
        public void SetNbKmRetour(double nbKm)
        {
                this.nbkmRetour = nbKm;
        }

        // méthode qui permet d'obtenir le surplus à régler au titre du dépassement du forfait
        // kilométrique
        public double GetMontantDepasForfait()
        {
                …
        }
}
```