BTS SERVICES INFORMATIQUES AUX ORGANISATIONS

E5 : Production et fourniture de services informatiques

SESSION 2018

Durée : 4h00 Coefficient : 5

CAS Localux

Ce sujet comporte 18 pages dont 10 pages de documentation.
La candidate ou le candidat doit vérifier que le sujet qui lui est remis est complet.

Aucun matériel ni document autorisé

Dossier documentaire

Barème

Mission 1	Retrait et restitution du véhicule loué	40 points
Mission 2	Gestion du dépassement du forfait kilométrique	40 points
Mission 3	Choix de solution pour le développement d'une application mobile	20 points
	Total	100 points

Présentation du contexte

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

Localux 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, Localux 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, Localux dispose :

- d'un site *«www.localuxhubert.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 Localux 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 Localux paie des frais de service.

L'entreprise prestataire de services

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

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

Le projet

L'offre Localux 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.localuxhubert.com»* et application «*GestActivité*»). Localux confie ces évolutions à DevApp.
Par ailleurs, Localux 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 Localux.

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.localuxhubert.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, Localux 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.localuxhubert.com»,* qui sont en phase de test*.*

Mission 1 : Retrait et restitution du véhicule loué

Documents à utiliser : 1, 2 et 3

Création d'un déclencheur (*trigger*)

Dans la base de données, on trouve deux types de formules :

- les formules de type "avec chauffeur" qui ont toutes la même durée de location : 24h ;
- les formules de type "sans chauffeur" qui permettent de louer un véhicule pour une durée exprimée en heures : 4 heures, 24 heures, 48 heures.

Un déclencheur *trgDateRetourLAC*, fourni dans le dossier documentaire, existe déjà. Il permet de mettre à jour la date et heure de fin de location (colonne *dateHreRetourPrevu* de la table *Location*) pour une location avec chauffeur. Il est exécuté lors de l'ajout d'une location avec chauffeur.

Un déclencheur *trgDateRetourLSC* doit être ajouté pour mettre à jour la colonne date et heure de fin de location de la table *Location* pour une location **sans chauffeur**.

Question 1.1

Écrire le déclencheur *trgDateRetourLSC*. Pour cela, vous utiliserez la partie de la base de données entourée dans le document 1.

Statistiques sur les locations sans chauffeur

Le module "statistiques sur les locations" de l'application «*GestActivité*» doit être modifié pour obtenir des informations sur l'activité location sans chauffeur. De nouvelles statistiques vont permettre d'anticiper le renouvellement de la flotte automobile de **Localux**.

Pour les locations sans chauffeur, **Localux** veut obtenir les modèles les plus demandés.

Votre chef de projet vous demande de réaliser une requête qui permette d'obtenir le nombre de locations sans chauffeur pour chaque nom de modèle, avec les modèles les plus demandés en tête de classement.

Question 1.2

Écrire la requête répondant au besoin exprimé par votre chef de projet.

Évolution de la structure de la base de données

***IMPORTANT** : la candidate ou le candidat présentera les évolutions de la structure de la base de données en adoptant le formalisme de son choix (schéma entité-association, diagramme de classes ou encore schéma relationnel)*

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 Localux 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 Localux et signé par la personne cliente pour approbation.

Lors du retrait du véhicule, Localux 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 Localux 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 Localux 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 Localux et ce coût estimé est enregistré au niveau de la location. La gestion de la matrice des dommages sort du cadre de cette mission.

Question 1.3

Selon le formalisme de votre choix, adapter la structure de la base de données existante pour prendre en compte les données du nouveau module.

***Important** : La candidate ou le candidat ne présentera sur sa copie que les éléments nécessaires à la mission.*

Mission 2 : Gestion du dépassement du forfait kilométrique

Documents à utiliser : 4, 5, 6, 7 et 8

IMPORTANT : La candidate ou le candidat peut choisir de présenter les éléments de code à l'aide du langage de programmation de son choix ou de pseudo-code algorithmique.

La nouvelle activité de réservation de véhicule sans chauffeur nécessite une modification de l'application «*GestActivité*» pour permettre de gérer le dépassement du forfait kilométrique lors de la restitution du véhicule.

Ce nouveau module permettra à l'employé de Localux en charge de la restitution du véhicule de saisir le nombre total de kilomètres apparaissant au compteur afin d'obtenir le surplus à régler par la personne cliente en cas de dépassement du forfait kilométrique.

Un paquetage de classes techniques permet d'accéder à la base de données et l'application utilise des classes métiers présentées dans le dossier documentaire.

Pour l'instant les 3 formules ci-dessous sont proposées sur le site de réservation, mais d'autres formules sont à l'étude pour l'année 2019 :

- la formule "Forfait 4h" donne droit à 4h pour un forfait (nombre de kilomètres inclus) de 150 kilomètres ;
- la formule "Forfait 24h" donne droit à 24h pour un forfait de 300 kilomètres ;
- la formule "Forfait 48h" donne droit à 48h pour un forfait de 700 kilomètres.

Au-delà du forfait kilométrique, tout kilomètre supplémentaire effectué sera facturé à un tarif dépendant du modèle du véhicule loué. Par exemple, pour la location d'une Renault Espace, tout kilomètre supplémentaire sera facturé 3 euros. En revanche, pour une Peugeot 5800, le kilomètre supplémentaire sera facturé 4 euros.

Si une personne cliente loue une Renault Espace avec la formule 24h, elle devra régler 240 euros lors de la réservation sur le site *«www.localuxhubert.com».* Si elle effectue 400 kilomètres, elle devra donc régler 300 euros au titre du dépassement du forfait kilométrique lorsqu'elle restituera le véhicule. En revanche, si elle n'a pas dépassé le forfait kilométrique, elle n'aura rien à régler lors de la restitution.

Question 2.1

Réaliser le code du constructeur de la classe métier LocationSansChauffeur. Le kilométrage au compteur lors de la restitution du véhicule sera initialisé à 0.

Question 2.2

Réaliser le code de la méthode *GetMontantDepasForfait()* de la classe métier LocationSansChauffeur qui permet d'obtenir le montant à régler par la personne cliente en cas de dépassement du forfait kilométrique.

Des tests unitaires sont nécessaires pour valider chaque méthode. Ceux-ci doivent donc être réalisés pour tester la méthode *GetMontantDepasForfait().*
Une ébauche d'un test, fourni dans le dossier documentaire, a été réalisée pour tester cette méthode. Il porte sur une location sans chauffeur, réalisée le 15 février 2018 et concerne une Renault Espace immatriculée LA-039-LP, achetée le 28 décembre 2016. La personne cliente a choisi la formule "Forfait 24h" et a réglé 240 euros. Lors du retrait du véhicule, le compteur indique 25000 kilomètres, et lors de sa restitution, il comptabilise 25400 kilomètres.

Question 2.3

Compléter le code du test unitaire de la méthode *GetMontantDepasForfait().*

Dans le cadre de la mise en place de la nouvelle activité de réservation de véhicule sans chauffeur, monsieur Berthu, directeur du service commercial de Localux, souhaite obtenir le montant total des sommes perçues suite à dépassement du forfait kilométrique pour chaque véhicule. En analysant ces informations, il pourra revoir à la hausse ou à la baisse le prix du kilomètre supplémentaire de certains véhicules.

Votre chef de projet a rédigé un courriel, fourni dans le dossier documentaire, qui présente les modifications à apporter pour répondre au besoin de monsieur Berthu.

Question 2.4

Afin de répondre au besoin de monsieur Berthu :

a) apporter les modifications nécessaires au diagramme de classes (*seules les classes concernées par les modifications devront être représentées sur votre copie*) ;

b) coder les modifications à apporter à la classe Vehicule.

Mission 3 : Choix de solution pour le développement d'une application mobile

Document à utiliser : 9

Localux, qui a actuellement recours aux applications mobiles de ses concurrents moyennant le versement d'une commission pour chaque prestation, envisage de créer une version mobile de son site de réservation pour que les utilisateurs puissent passer de la même façon leur réservation à partir de leur *smartphone* ou tablette.

Les objectifs de Localux sont :

- faire évoluer l'application *web* vers un site mobile disponible 24h/24 adapté à une clientèle cible plutôt haut de gamme ;
- augmenter ses parts de marché.

Localux se pose la question de la technologie à utiliser. En effet, trois solutions sont possibles : application native, application hybride ou application *web* (spécifique mobile ou adaptative -*RWD responsive web design*-).

L'entreprise a demandé à DevApp de l'aider à faire un choix entre ces différentes possibilités.

Votre chef de projet vous a remis un article, trouvé sur un blog, qui explique les différences. Le contenu technique de cet article fourni dans le dossier documentaire a été vérifié auprès de sources fiables.

Pour comparer des solutions, DevApp utilise une matrice de compatibilité. Il s'agit d'un tableau ayant en entête de **colonne** les solutions à étudier et en entête de **ligne** les critères de comparaison. Chaque cellule du tableau contient un **avis** sous la forme d'étoiles (1 étoile : «peu satisfaisant», 2 étoiles : «satisfaisant» et 3 étoiles : «très satisfaisant»).

Question 3.1

Dresser un tableau comparatif, similaire à ceux utilisés chez DevApp, afin de comparer les différentes solutions de développement envisagées. Vous utiliserez les critères de comparaison suivants : temps de développement, ergonomie, rapidité d'exécution hors connexion réseau.

Localux a décidé de demander à DevApp un devis concernant la conception, la mise en production et la maintenance d'une application native.

DevApp a conclu qu'il fallait 3 développeurs à temps plein sur ce projet pour une durée de 2 mois. Un développeur est facturé 400 € TTC par jour. Le mois contient en moyenne 20 jours ouvrables.

Le montant total facturé à Localux pour le développement et la production de l'application mobile sera de 48 000 euros.

Question 3.2

Détailler le calcul du montant total facturé à Localux.

La comptabilité de Localux a montré que cette dernière reverse à Uber et Chauffeur Privé en moyenne 50 000 € TTC/an pour l'utilisation de leurs applications mobiles.

La maintenance de l'application développée par DevApp va coûter 20 000 € TTC par an dès la première année.

Localux s'interroge sur le choix à effectuer en matière de stratégie informatique : faut-il continuer à utiliser les applications mobiles de Uber et Chauffeur privé (solution 1) ou faire développer et maintenir sa propre application mobile (solution 2) ?

Question 3.3

À partir de calculs chiffrés, déterminer la solution la moins coûteuse pour la première année.

Question 3.4

Conclure sur l'évolution du coût de chaque solution au bout de trois ans.

Question 3.5

Fournir d'autres éléments que le coût permettant d'opter pour l'une ou l'autre de ces deux solutions.

Dossier documentaire

Document 1 : Structure de la base de données

Modélisation conceptuelle

Schéma entité-association :

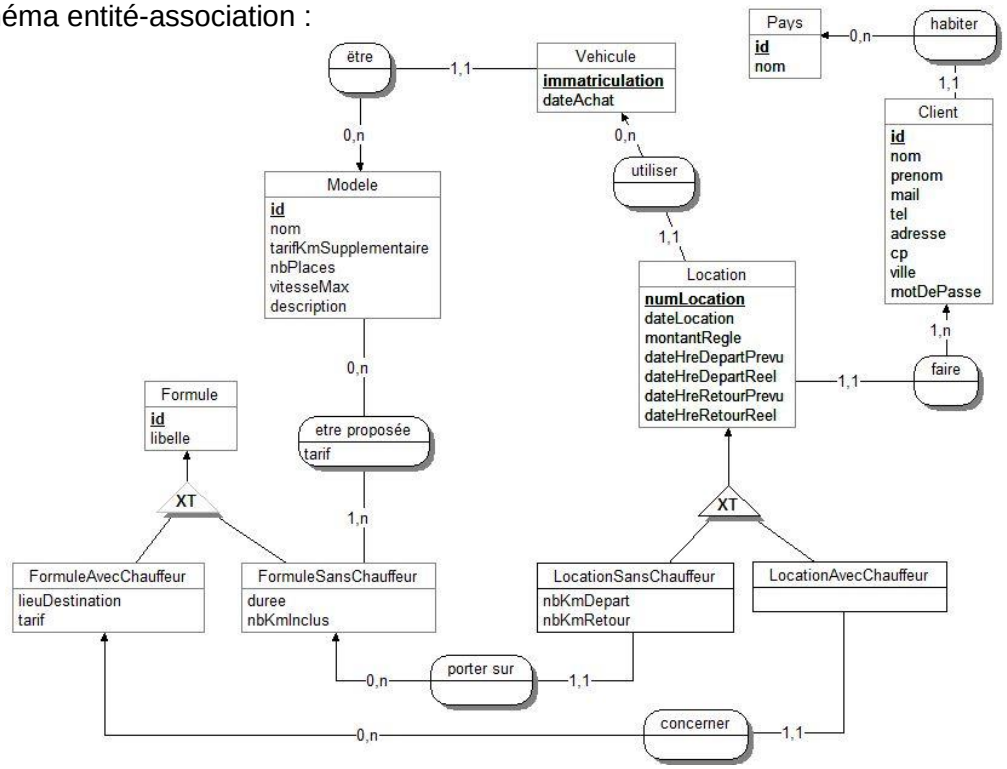


Diagramme de classes :

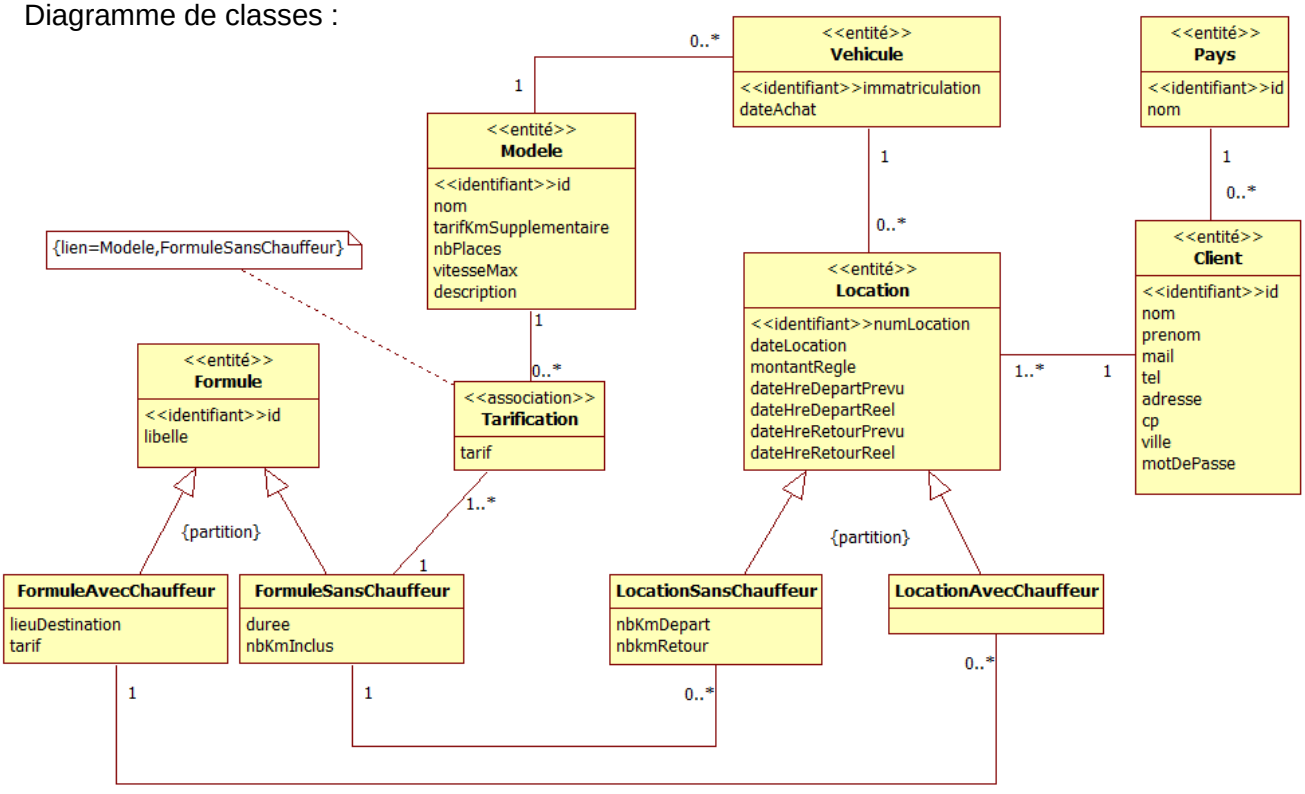


Schéma relationnel

Vehicule (immatriculation, dateAchat, idModele)
Clé primaire : immatriculation
Clé étrangère : idModele en référence à id de Modele

Modele (id, nom, tarifKmSupplementaire, nbPlaces, vitesseMax, description)
Clé primaire : id

EtreProposee (idModele, idFormule, tarif)
Clé primaire : idModele, idFormule
Clés étrangères : idModele en référence à id de Modele
idFormule en référence à id de Formule

Formule (id, libelle)
Clé primaire : id

FormuleAvecChauffeur (id, lieuDestination, tarif)
Clé primaire : id
Clé étrangère : id en référence à id de Formule

FormuleSansChauffeur (id, duree, nbKmInclus)
Clé primaire : id
Clé étrangère : id en référence à id de Formule

Location (numLocation, dateLocation, montantRegle, dateHreDepartPrevu, dateHreDepartReel, dateHreRetourPrevu, dateHreRetourReel, idClient, immatriculation)
Clé primaire : numLocation
Clés étrangères : idClient en référence à id de Client
immatriculation en référence à immatriculation de Vehicule

LocationSansChauffeur (numLocation, nbKmDepart, nbKmRetour, idFormule)
Clé primaire : numLocation
Clés étrangères : numLocation en référence à numLocation de Location
idFormule en référence à id de FormuleSansChauffeur

LocationAvecChauffeur (numLocation, idFormule)
Clé primaire : numLocation
Clés étrangères : numLocation en référence à numLocation de Location
idFormule en référence à id de FormuleAvecChauffeur

Pays (id, nom)
Clé primaire : id

Client (id, nom, prenom, mail, tel, adresse, cp, ville, motDePasse, idPays)
Clé primaire : id
Clé étrangère : idPays en référence à id de Pays

Remarques :

- Toutes les colonnes contenant des dates (dont le nom commence par "date") sont de type horodatage ;
- La colonne duree dans la table FormuleSansChauffeur est exprimée en heures entières

Document 2 : Déclencheur trgDateRetourLAC

```
CREATE TRIGGER trgDateRetourLAC BEFORE INSERT ON LocationAvecChauffeur
FOR EACH ROW
BEGIN
   DECLARE duree INTEGER ;
   DECLARE dateDepart DATETIME;
   DECLARE dateRetour DATETIME;
   /* la durée est de 24h pour une  location avec chauffeur */
   SET duree = 24;
   /* récupération de la date de départ dans une variable */
   SET dateDepart = (SELECT dateHreDepartPrevu FROM Location
                        WHERE numLocation = NEW. numLocation) ;
   SET dateRetour  = DATE_ADD(dateDepart, INTERVAL duree HOUR) ;

   UPDATE Location set dateHreRetourPrevu = dateRetour
                    where numLocation = NEW. numLocation ;
END ;
```

Remarque : la fonction **DATE_ADD(**date**, INTERVAL** *nb* **unitéDeTemps)** ajoute *nb* unités de temps à la date passée en paramètre (*nb* pouvant être une valeur ou le nom d'une variable)

Exemples de valeurs possibles pour **unitéDeTemps** : SECOND, HOUR, DAY, etc.

Document 3 : Rapport établi lors de la restitution d'un véhicule

Numéro de location : ***240***
Modèle : ***Lamborghini Huracán LP 610-4 Spyder***
Immatriculation : ***LA – 038 – SP***

Numéro de client : ***56***
Nom et prénom du client : ***Agathe MORIN***

Retrait du véhicule

Contrôle n° 320 réalisé par ***Marc BOTUR***

Kilométrage	Date - Heure
12 480	13/04/2018 08:30

Dommages constatés sur le véhicule

	RS	RP	EC		RS	RP	EC
Aile AV G.	☐	☐	☐	Aile AV D.	☐	☐	☐
Aile AR G.	☐	☐	☐	Aile AR D.	☐	☐	☐
Calandre	☑	☐	☐	Phare AV G.	☐	☐	☐
Phare AV D.	☐	☐	☐	Siège cond.	☐	☐	☐
Siège pass.	☐	☐	☐	Tableau de b.	☐	☐	☐
Porte AV G.	☐	☐	☐	Porte AV D.	☐	☐	☐
...				...			

Observations :
L'intérieur de la voiture est poussiéreux.

Restitution du véhicule

Contrôle n°436 réalisé par ***Aline LIDBO***

Kilométrage	Date - Heure
12 760	14/04/2018 9:15

Dommages constatés sur le véhicule

	RS	RP	EC		RS	RP	E
Aile AV G.	☐	☐	☐	Aile AV D.	☐	☐	☐
Aile AR G.	☐	☐	☐	Aile AR D.	☐	☐	☐
Calandre	☑	☐	☐	Phare AV G.	☐	☐	☐
Phare AV D.	☐	☐	☐	Siège cond.	☐	☐	☐
Siège pass.	☐	☐	☐	Tableau de b.	☐	☐	☐
Porte AV G.	☐	☑	☐	Porte AV D.	☐	☐	☐
...							

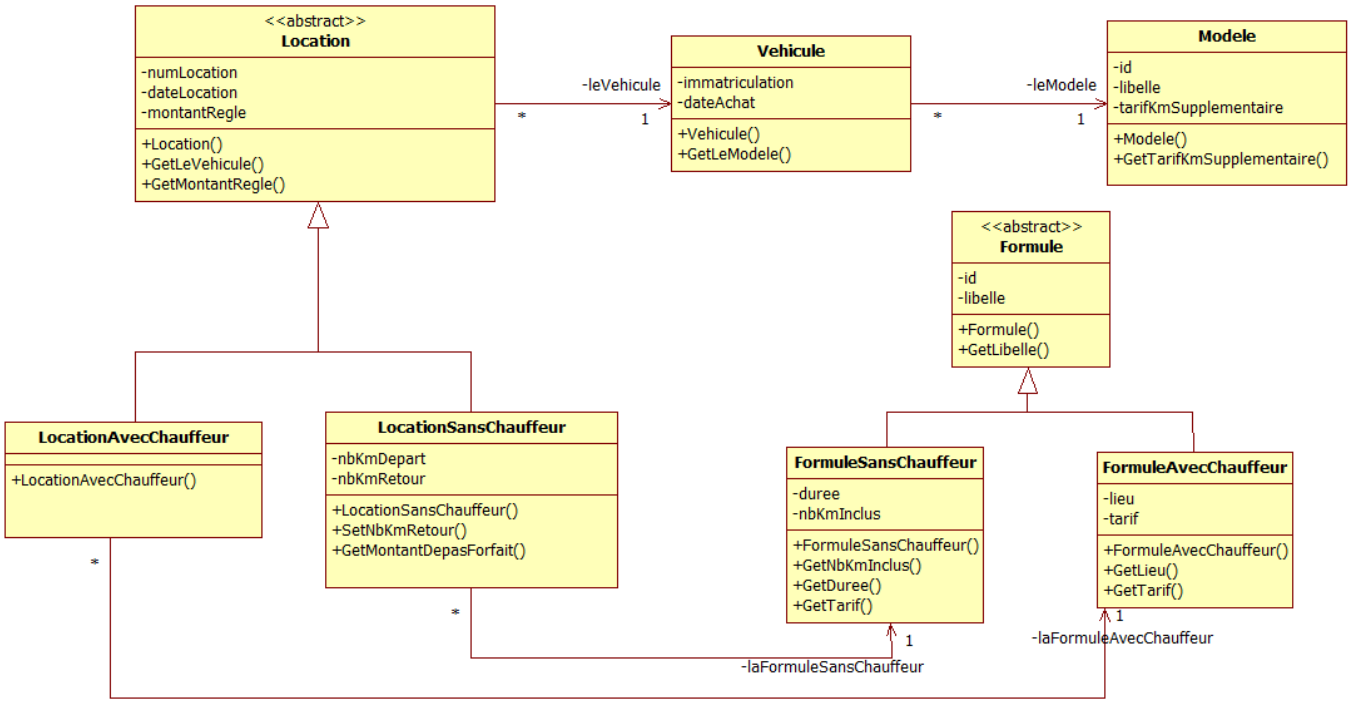
Observations :
À plusieurs reprises, nous avons entendu des bruits anormaux lors du freinage.

RS : Rayure superficielle ; RP : Rayure profonde ; EC : Enfoncement/Choc

Coût estimé des réparations : 2 000 euros

Signature du client :

Document 4 : Extrait du diagramme des classes métier



Remarque *: les paramètres des méthodes et le type de valeur retourné n'apparaissent pas sur le diagramme de classes*

Document 5 : 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()
        {
                // À compléter sur votre copie
        }
}
```

Document 6 : Méthode de test unitaire à compléter

```
[TestMethod]
public void TestGetMontantDepasForfait()
{
    // création d'une instance de modèle
    Modele leModele = new Modele(1, "Renault Espace", 3);
    // création d'une instance de Vehicule
    Vehicule leVehicule = new Vehicule("LA-039-LP", new DateTime(2016, 12, 28),
                                                    leModele);
   // création d'une instance de FormuleSansChauffeur
    FormuleSansChauffeur laFormule = new FormuleSansChauffeur(1, "Forfait 24h", 24,300);
    // création d'une instance de LocationAvecChauffeur
    LocationSansChauffeur laLocation = new LocationSansChauffeur(1,
                            new DateTime(2018, 02, 15), 240, 25000,
                            leVehicule, laFormule);

        // À compléter sur votre copie
}
```

La classe technique **Assert** contient différentes méthodes statiques permettant de savoir si le test unitaire a réussi ou non. Elle contient la méthode ***AreEqual*** dont voici la signature :

public **static** void AreEqual(double expected, double actual, string message)

expected : valeur de type double contenant la valeur attendue
actual : valeur de type double contenant la valeur obtenue
message : message à afficher si l'assertion échoue, c'est-à-dire lorsque la valeur attendue est différente de la valeur obtenue.

Exemple : Assert.AreEqual(val1, val2, "erreur") ;

Document 7 : Mail de la chef de projet

From Sophie.Louvin@devapp.net
To: Alix.Pilou@devapp.net
Subject : modification de GestActivité pour répondre aux besoins de M. Berthu

Bonjour Alix,
Tu dois modifier la classe ***Vehicule*** pour y ajouter une méthode nommée *GetTotalDepasForfait* qui retournera le montant total des sommes versées suite à dépassement du forfait kilométrique pour toutes les locations sans chauffeur réalisées avec le véhicule courant. Tu devras ajouter un nouvel attribut dans cette classe qui permettra d'accéder à toutes les instances **LocationSansChauffeur** réalisées avec le véhicule et donc modifier le constructeur de la classe. Evidemment, tu ajouteras également dans la classe ***Vehicule*** une méthode qui permettra d'ajouter une nouvelle location sans chauffeur au véhicule.
Bon courage et n'hésite pas à me contacter si tu as besoin d'un complément d'information.
Sophie Louvin

Document 8 : Exemple d'utilisation d'une collection

En C#, la collection de type List<T> permet d'enregistrer des éléments de type T où T peut être n'importe quel type d'éléments (entier, float, nom de classe,…)

L'exemple ci-dessous permet de manipuler une collection d'entiers.

Le principe est le même, quel que soit le type des éléments.

```
List<int> mesNombres;                    // déclaration d'une collection d'entiers
mesNombres = new List<int>();            // instanciation de la collection
mesNombres.Add(250);                     // ajout de deux entiers à la collection
mesNombres.Add(10);

// parcours de la collection
foreach (int unNombre in mesNombres)     // parcours de la collection
{
        Console.WriteLine(unNombre);     // affichage de chaque élément
}

mesNombres.RemoveAt(1);                  // suppression du 2e élément (indice 1)
Console.WriteLine(mesNombres[0]);        // affichage du 1er élément (indice 0)
```

Document 9 : Technologies de développement mobile

Inspiré d'un blog microsoft.com

1. Le natif

C'est l'approche la plus naturelle. Un développement est effectué pour chacune des plateformes cibles dans le langage prévu pour elles. Cette technique permet d'obtenir un résultat au plus proche du système ciblé. Les outils dédiés à chacun des environnements sont utilisés ce qui permet de disposer des meilleures conditions pour la réalisation et les tests de l'application.

Faire du natif, c'est donc réaliser plusieurs développements distincts (pour Android, iOS, Windows,etc.). Très peu de choses sont mutualisées et chaque mise à jour ou correctif devra être réalisé trois fois. Cependant, cela offre :

- la souplesse nécessaire à la réalisation d'une application qui s'adapte parfaitement à l'écosystème de chaque téléphone ;
- des performances qui peuvent se révéler excellentes car relatives à la qualité du développement.

Faire du natif reste néanmoins l'approche la plus chère : elle nécessite le plus de compétences et de connaissances, mais c'est celle qui donne souvent la meilleure garantie de qualité.

2. Le quasi-natif (application hybride)

Si on écarte la solution native pour des raisons de coût et de réactivité lors des mises à jour et des correctifs, il faut se diriger vers une approche qui permet de partager le maximum de code possible.

Une des voies permettant d'arriver à ce résultat est le ''quasi-natif''. Un seul langage de programmation est utilisé et un seul développement est effectué, mais un ensemble d'outils qui nécessite un certain temps de formation, est utilisé pour compiler et packager une version de l'application adaptée à chaque plateforme cible. Le code ''générique'' est traduit dans le langage compris par la plateforme puis compilé nativement ou exécuté par une machine virtuelle, ce qui peut avoir un impact sur la performance.

Les applications hybrides proposent une ergonomie très proche de celles des solutions natives.

Des frameworks/outils tels que Cordova ou encore Ionic permettent de développer des applications hybrides.

3. Le *web* (application *web*-mobile ou adaptative)

La troisième approche est une extension naturelle d'une des fonctions premières des *smartphones* depuis leur apparition : l'accès au *world wide web* depuis un navigateur sur lequel s'exécute de l'*HTML* et du *JavaScript*.
Depuis toujours, la tendance est de rendre les sites accessibles sur les mobiles en développant une version mobile distincte puis en appliquant tous les principes d'adaptabilité (*RWD responsive web design*).

Les langages HTML5, CSS3 et Javascript sont utilisés pour développer ces applications qui sont exécutables sur tous les smartphones, via un navigateur web.

Grâce à un développement unique et une utilisation multiplateforme, ces applications permettent un gain de temps et d'argent. Mais, n'utilisant pas la mémoire embarquée du téléphone, elles perdent en fluidité (temps de chargement plus long) et en performance, surtout si l'application doit réaliser des tâches complexes.

Par ailleurs, ces applications peuvent ne pas suivre les recommandations graphiques d'une application native (Android, iOS, Windows Phone, etc.), et elles ont souvent tendance à créer un design standard pour toutes les plateformes : rivaliser avec une application native est, de ce fait, assez difficile…

Etant donné que le *web* fonctionne sur tous les mobiles et qu'il est possible de développer des sites qui s'affichent correctement sur un mobile, on peut se demander pourquoi ne pas emballer (*packager*) ces sites dans une application ? Ce choix permettrait de rendre l'application disponible sur les plateformes de téléchargement d'applications destinées à des applications mobiles (*stores*), de donner l'impression qu'il s'agit d'une application classique et multiplateforme. Le seul point négatif serait lié à l'usage difficile des fonctionnalités du téléphone comme la sauvegarde de fichier en local, l'utilisation de l'appareil photo ou encore l'accès aux contacts.