Jalon 3	Création d'un client lourd pour la gestion du matériel

Annexe à utiliser :
Exemple de fichier XML à générer pour un client
Descriptif des classes pour le client lourd

La société CASHCASH a vendu et déployé de nombreux équipements à ses clients. Ces équipements sont fournis avec ou sans contrat de maintenance. Certains de ces clients, les hypermarchés par exemple, possèdent de nombreux terminaux. La maintenance du matériel acheté est donc assurée en interne ou par la société CASHCASH. Par soucis de rationalisation, ces clients souhaiteraient pouvoir intégrer les données concernant les matériels (date d'installation, date d'échéance du contrat de maintenance…) à leur propre système d'information. Le format de données privilégié est le XML.

Par ailleurs, l'expertise technique de la société CASHCASH montre qu'il est fréquemment intéressant pour un client d'externaliser la maintenance de son matériel. La société CASHCASH s'attend donc à couvrir de nombreux matériels déjà vendus par des contrats de maintenance. Les commerciaux transmettent en général par téléphone ou par mail les informations utiles.

Enfin, pour d'autres clients, les petits commerçants par exemple, une relance plus classique par courrier est envisagée. La durée des contrats est d'un an. Les contrats arrivant à expiration peuvent être renouvelés. La procédure de renouvellement d'un contrat se caractérise par l'enregistrement de la date du renouvellement et la modification de la date d'échéance du contrat ; on conserve toutefois, sans la modifier, la date à laquelle le contrat a été signé pour la première fois.

La société CASHCASH souhaiterait posséder une application de type Client lourd avec interface graphique assurant deux fonctionnalités :
Le gestionnaire doit pouvoir générer le fichier xml pour un client donné
Le gestionnaire doit pouvoir couvrir par un contrat de maintenance les matériels déjà vendus.
Le gestionnaire doit pouvoir générer des courriers automatiques de relance au format pdf

1) Organisation de VDEV

VDEV est une ESN (Entreprise de services du Numérique). Depuis sa création, développe des applicatifs pour divers clients. Elle a par exemple développé des applications de gestion de terrasses pour différentes villes ou alors fait évoluer le système d'information d'une coopérative de producteurs de noix.

Les applicatifs développés sont également de natures diverses. Cela peut être une application Web permettant de réserver des terrasses par des établissements d'une ville. Cette application permet également la consultation des tarifs pratiqués. Cela peut être également une application conçue à l'aide d'un langage objet permettant de positionner des terrasses sur la carte d'une ville donnée, en fonction d'un certain nombre de critères. Enfin, VDEV a fait évoluer ses technologies pour tenir compte des terminaux mobiles.

Pour diversifier ses activités, VDEV, décide de s'intéresser désormais au marché des entreprises. Développer des applications pour des acteurs privés peut être générateur de profit selon les dirigeants de VDEV. Après une longue campagne de communication et marketing, un premier contrat est en passe d'être finalisé avec l'entreprise CASHCASH.

Contraintes techniques et organisationnelles au sein de VDEV

Organisation du projet :
Cycle en V ou méthodes agiles (SCRUM)

Pour la couche données :
VDEV souhaite retrouver les spécifications et contraintes techniques suivantes ;

- Un dictionnaire de données détaillé
- Un modèle conceptuel de données comportant les extensions Merise 2
- Un modèle relationnel
- un SGBD ayant des fonctionnalités avancées (contraintes de domaine, implémentation de PL/SQL…).
- Un script de création de la base et de ses objets avec un jeu d'essai conséquent
- Une gestion très stricte au niveau de la couche donnée vous est demandée.
- Implémenter la BDD dans deux environnements différents :
  a- Windows / Mysql
  b- Linux / Postgre

Pour une application de type client lourd :

VDEV souhaite retrouver les spécifications et contraintes techniques suivantes ;

- Un diagramme de classe
- Une code commenté
- Une gestion fine des erreurs
- Une documentation technique au format HTML
- Un diagramme de cas d'utilisation
- Une description textuelle des cas d'utilisation
- Un diagramme de classe UML
- Un maquettage des IHM (les composants doivent être nommés)
- Le choix des Architecture logicielles retenues
- Rapport de tests
- Tests unitaires

.

Vdev souhaite retrouver un certes nombres de règles de développement

Régles de développement pour un client lourd

- Toute méthode publique d'une classe sera précédée d'une documentation qui comprendra au minimum le résumé, la description des paramètres et du résultat suivant le format JavaDoc.
- Le nom (identificateur) d'une classe respectera la notation Pascal : la première lettre du nom de méthode et la première lettre de chaque mot présent dans l'identificateur sont en majuscules. Par exemple, LigneCommande respecte la notation Pascal.
- Le nom d'une méthode est un verbe, ou un groupe verbal. Les méthodes qui permettent de lire (resp. écrire) directement une variable privée d'instance sont préfixées par get (resp. set), suivi du nom de la variable.
- Le nom (identificateur) des méthodes, paramètres formels et des variables locales respectera la notation Camel : la première lettre du nom est en minuscules et la première lettre de chaque mot présent dans l'identificateur est en majuscules.Par exemple, uneQte respecte la notation Camel.
- Le nom des méthodes, paramètres et variables doit être le plus explicite possible et informer de leur rôle. Il faut privilégier la lisibilité à la concision.
- Le nom des méthodes, paramètres et variables ne contient que des lettres non accentuées ou des chiffres : le tiret bas, trait d'union ou tout autre caractère non alphanumérique sont interdits.

2) Annexes utiles

Annexe - Exemple de fichier XML à générer pour un client (materielClientcli15432.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<listeMateriel>
<materiels idClient="cli15432" >
      <sousContrat>
      <materiel numSerie="00213">
            <type refInterne="A506" libelle=" souris sans fil logitech S   />
            <famille codeFamille="SE"  libelle="serveur"  />
            <quantite>50</quantite>
            <date_vente>10-03-21</date_vente>
            <date_installation>12-05-21</date_installation>
            <prix_vente>32.5</prix_vente>
            <emplacement>"Boulangerie" </emplacement>
            <nbJourAvantEcheance>94</nbJourAvantEcheance >
      </materiel>
      <materiel numSerie="00214">
            <type refInterne="B106" libelle=" clavier sans fil logitech S   />
            <famille codeFamille="SE"  libelle="serveur"  />
            <date_vente>10-04-21</date_vente>
            <date_installation>17-05-21</date_installation>
            <prix_vente>18.5</prix_vente>
            <emplacement>"Boulangerie" </emplacement>
            <nbJourAvantEcheance>94</nbJourAvantEcheance >
      </materiel>
      </sousContrat>
      <horsContrat>
      <materiel numSerie="00215">
            <type  refInterne="A507"  libelle="  Datalogic  QW2100  Datalogic
QuickScan Lite/>
            <famille codeFamille="PC"  libelle="petite caisse"  />
            <date_vente>10-03-21</date_vente>
            <date_installation>16-06-21</date_installation>
            <prix_vente>68.26</prix_vente>
            <emplacement>"Boulangerie" </emplacement>
      </materiel>
      </horsContrat>
</listeMateriel>
```

Annexe – Descriptifs des classes pour le client lourd

Classe PersistanceSQL

```
Public
   // Constructeur
   PersistanceSQL (ipBase : chaîne, port : entier, nomBaseDonnee : chaîne)
   // Construit un objet PersistanceSql. Cet objet permettra de charger les
données depuis une base
   // de données ou de les sauvegarder dans la base.

   procédure RangerDansBase (unObjet : Objet)
   // Stocke les données de l'objet dans la base de données.

   fonction ChargerDepuisBase (id : chaîne, nomClasse : chaîne) : Objet de la
classe NomClasse.
   // Retourne l'objet de la classe NomClasse dont l'identifiant est "id".
Cet objet est chargé
   // depuis la base de données, ainsi que l'ensemble de ses objets liés (voir
l'exemple d'utilisation
   // ci-dessous). Retourne NULL si aucun objet de cette classe ne possède
cet identifiant.
```

Fin classe

Exemple d'utilisation :

// *persist* est une instance de PersistanceSQL
Client leClient ← persist.chargerDepuisBase ("2", "Client")
// *leClient* est l'instance de la classe Client dont l'identifiant est 2.

Toutes les commandes du distributeur sont automatiquement chargées dans la collection *leClient.lesMateriels*. Toutes les données utiles sont également chargées.

Les constructeurs, accesseurs et mutateurs n'ont pas été décrits. Ils sont bien évidement à ajouter.
Vous pourrez ajouter toute méthode que vous jugerez utile.

Classe Client

```
Privé
   numClient : chaine
   raisonSociale : chaîne
   siren : chaîne
   codeApe : chaîne
   adresse : chaîne
   telClient : chaîne
   email : chaîne
   dureeDeplacement : entier
   distanceKm : entier
   lesMateriels : Collection de <Materiel> // Tous les matériels du client.
   leContrat : ContratMaintenance // peut être nul si le client ne possède
pas de contrat


Public

   fonction getMateriels() : Collection de <Materiel>
   // Retourne l'ensemble des matériels du client

   fonction getMaterielsSousContrat() : Collection de <Materiel>
   // Retourne l'ensemble des matériels pour lesquels le client a souscrit un
contrat                de                maintenance                qui
   // est encore valide (la date du jour est entre la date de signature et la
date d'échéance)

   fonction estAssure() : booléen
   // Retourne vrai si le client est assuré, faux sinon
```

Fin classe

Classe ContratMaintenance

```
Privé
   numContrat : chaîne
   dateSignature : date
   dateEcheance : date
   lesMaterielsAssures : Collection de <Materiel> // Tous les matériels sous
contrat de maintenance

Public

   fonction getJoursRestants () : entier
   // Renvoie le nombre de jours avant que le contrat arrive à échéance

   fonction estValide () : booléen
   // indique si le contrat est valide (la date du jour est entre la date de
signature et la date d'échéance)

   procédure ajouteMateriel (Materiel unMateriel)
   // ajoute unMatériel à la collection lesMaterielsAssures si la date de
signature du contrat est     //  antérieure  à  la  date  d'installation  du
matériel
```

Fin classe

Classe Materiel

```
Privé
   numSerie : entier                                    .
   dateVente : Produit
   prixVente : réel
   emplacement : chaîne
   leType : TypeMateriel

Public

   fonction xmlMateriel () : chaîne
   // Retourne la chaîne correspondant au code XML représentant le matériel
(voir annexe).
```

Fin classe

Classe TypeMateriel

```
Privé
   referenceInterne : chaine
   libelleTypeMateriel : chaine
   laFamille : Famille


Public

// libre
```

Fin classe

Classe Famille

```
Privé
   codeFamille : chaine
   libelleFamille : chaine

Public

//libre
```

Fin classe

Classe GestionMateriels

```
Privé
   donnees : PersistanceSQL
   // Attribut qui permet de rendre les objets métiers accessibles.

Public
   //Constructeur
   GestionMateriels (lesDonnees : PersistanceSQL)
   // Construit un objet GestionMateriels avec un modèle de persistance
associé.

   fonction getClient (idClient : entier) : Client
   // Retourne l'objet Distributeur qui possède l'identifiant idDistributeur
passé en paramètre,
   // retourne null si aucun Distributeur ne possède cet identifiant.

   fonction XmlClient (unClient :  Client) : chaîne
   // Retourne une chaîne de caractères qui représente le document XML de la
liste des matériels
   // du client passé en paramètre comme le montre l'exemple de l'annexe.

   fonction statique XmlClientValide (xml :  chaine) : booleen
      // Retourne un booléen Vrai si le fichier xml respecte la DTD référencée
   dans le fichier XML, Faux sinon
```

Fin classe