

BTS SERVICES INFORMATIQUES AUX ORGANISATIONS

E6 : Cybersécurité

Durée : 4 heures Coefficient : 4

Cas ReWebCaisse

Ce sujet comporte 22 pages dont 10 pages de documentation.
Le candidat doit vérifier que le sujet qui lui a été remis est complet.

Aucun matériel ni document autorisé

Dossier documentaire :

Document 1 : schéma complet de la base de données utilisée par l'application actuelle	
AchatWebCaisse	10
Document 2 : maquette du formulaire "Choix de formule pour un point de vente"	10
Document 3 : ébauche du code de vérification de l'authentification.....	11
Document 4 : maquette du formulaire "Recherche des consommateurs réguliers"	11
Document 5 : extrait du schéma relationnel pour la gestion des consommateurs fidèles	11
Document 6 : Rappels concernant les expressions régulières.....	12
Document 7 : extrait du diagramme de classes utilisé par le module GRC	13
Document 8 : extrait de la classe GRC contenant la méthode <i>listeConsoAFideliser</i>	13
Document 9 : exemple d'utilisation d'une collection <i>ArrayList</i>	14
Document 10 : classe de test de la méthode <i>addFidelite</i>	14
Document 11 : extrait du diagramme de classes utilisé par le module de statistiques.....	15
Document 12 : extrait de la classe Statistique utilisée par le module de statistiques.....	17
Document 13 : schéma de l'infrastructure envisagée.....	18
Document 14 : choix d'un hébergeur répondant à la certification NF525	18

Barème :

Mission 1	Souscription en ligne	30 points
Mission 2	Fidélisation des consommateurs	25 points
Mission 3	Statistiques de ventes	30 points
Mission 4	Mise en production	15 points
	Total	100 points

L'organisation cliente : Nasdy

La société Nasdy, implantée en Martinique et en Guadeloupe, est une entreprise de services du numérique créée en 2001 par deux ingénieurs. Elle accompagne les organisations dans leurs projets de dématérialisation et participe au développement local du numérique (animation d'ateliers sur le *big data*, les *Google Apps*, etc.), tout en menant une expansion à l'international.

Elle propose également la création de sites, la gestion de la communication numérique ou encore la réalisation d'études de faisabilité technique de projets web ou mobiles.

Nasdy assure une veille technologique sur les nouveaux besoins, ce qui amène la société à développer des applications en interne en vue d'une commercialisation.

Le logiciel *WebCaisse*

Pour rendre impossible la fraude, l'article 88 de la loi de finances pour 2016 instaure, à partir du 1er janvier 2018, l'obligation pour les commerçants et autres professionnels assujettis à la TVA d'enregistrer les paiements de leurs clients au moyen d'un logiciel de comptabilité ou d'un système de caisse sécurisé et certifié NF525.

Pour répondre à ce besoin réglementaire, Nasdy a décidé de développer un logiciel de caisse enregistreuse en ligne, nommé *WebCaisse*, qu'elle propose à ses clients.

Nasdy mise sur la proximité avec les commerçants des Antilles (épicier, coiffeurs, boulanger, etc.) qui sont nombreux à tenir leurs comptes sur support papier. Dans un second temps, la *WebCaisse* sera proposée au reste de la clientèle potentielle (Dom, Tom, France métropolitaine, étranger).

La *WebCaisse* proposée par Nasdy est un logiciel en mode *SaaS (Software as a Service)* destiné à l'encaissement et à la gestion de toute activité d'un commerce indépendant ou d'un réseau de magasins. Elle met à la disposition des points de vente des solutions d'encaissement sur divers supports : caisse, tablette, *smartphone*, etc.

Le logiciel *WebCaisse* et sa base de données sont hébergés pour chaque client chez un hébergeur spécialisé. Il propose différents modules : gestion des ventes, gestion des stocks, statistiques, campagne publicitaire, aide à la décision, gestion de la relation client et e-commerce. D'autres modules pourront, par la suite, être développés en fonction des besoins. Différentes formules de souscription seront proposées, chacune contenant différents modules paramétrables selon les besoins du client.

Une première version du logiciel *WebCaisse* a été développée ; elle contient certains modules proposés par le logiciel.

L'application *AchatWebCaisse*

Pour conserver les caractéristiques des formules de *WebCaisse* souscrites, une application *AchatWebCaisse* a été développée.

Utilisée uniquement par le service commercial de Nasdy, elle permet actuellement d'enregistrer et de consulter la formule de *WebCaisse* souscrite par chaque client et utilisée par tous ses points de vente. Le développement du logiciel *WebCaisse* et de l'application *AchatWebCaisse* ont été confiés à une équipe interne à Nasdy composée de trois personnes et dirigée par un chef de projet, M. Renaud Boileau.

Au sein de la société Nasdy, vous prenez part au projet de développement de l'application *AchatWebCaisse* et du logiciel *Webcaisse*.

Précisions sur les termes "clients" et "consommateurs" utilisés dans ce dossier :

- les **clients** sont les commerçants qui achètent le logiciel *WebCaisse* ;
- les **consommateurs** sont les personnes qui effectuent des achats chez les commerçants.

BTS Services informatiques aux organisations E6 : Cybersécurité	Code : E6SLAM	2022 Page 2/22
--	---------------	-------------------

Mission 1 - Souscription en ligne

Nasdy souhaite que chaque client puisse gérer la souscription au logiciel *WebCaisse* pour chacun de ses points de vente, sans être obligé de contacter le service commercial de Nasdy. Pour cela, l'application *AchatWebCaisse* doit être modifiée pour :

- mettre à disposition des clients des formulaires accessibles sur *internet* permettant de souscrire ou modifier une formule *WebCaisse* propre à chaque point de vente du client ;
- gérer les caractéristiques des formules, des modules et des programmes de fidélisation des consommateurs.

Toutes les informations collectées via l'application *AchatWebCaisse* permettront ensuite de réaliser automatiquement le déploiement du logiciel *WebCaisse* à partir des caractéristiques choisies par le client.

L'application *AchatWebCaisse* utilise une base de données dont le schéma complet vous est fourni dans le dossier documentaire.

Votre chef de projet vous demande de faire évoluer la structure de la base de données utilisée par l'application *AchatWebCaisse* pour qu'elle intègre les données nécessaires à la souscription en ligne présentée ci-après.

Partie 1 – Évolution de la base de données pour la souscription en ligne

Documents à utiliser : 1 et 2

La souscription en ligne sera réalisée en trois étapes : choix de la formule, choix du programme de fidélisation et choix des langues pour les modules.

Choix de la formule

Pour commander le logiciel *WebCaisse* en ligne, le client devra fournir différentes informations qui seront mémorisées : sa raison sociale, son adresse, son numéro de téléphone et son domaine d'activité (restauration, alimentation, coiffure, etc.).

L'identifiant et le mot de passe qui lui permettront de se connecter seront transmis par messagerie électronique.

Question 1.1

Quels sont les risques de transmettre les identifiants de cette façon ? Proposez des solutions pour limiter ces risques.

Une fois connecté, le client devra renseigner le nom et l'adresse précise de ses points de vente puis, pour chacun d'eux, il sera invité à sélectionner une formule de *WebCaisse* dans un formulaire lui

BTS Services informatiques aux organisations E6 : Cybersécurité	Code : E6SLAM	2022 Page 3/22
--	---------------	-------------------

affichant les différents modules compris dans chaque formule. Une maquette du formulaire est fournie dans le dossier documentaire.

Le taux de commission, qui concerne uniquement, pour le moment, le module e-commerce, sera affiché : il permettra au client de connaître le taux de commission qui sera appliqué au montant des ventes en ligne réalisées.

Choix du programme de fidélisation des consommateurs

Le logiciel *WebCaisse* propose trois programmes de fidélisation :

- le programme de type 1 "**Tampon**" ;
- le programme de type 2 "**Montant Achat**" ;
- le programme de type 3 "**Points**".

Le programme de fidélisation retenu pour chaque point de vente sera mémorisé dans la base de données.

Question 1.2

Modifier la structure de la base de données utilisée par l'application **AchatWebCaisse** afin de permettre la souscription en ligne du logiciel **WebCaisse**.

IMPORTANT : le candidat présentera les évolutions de la structure de la base de données en adoptant le formalisme conceptuel de son choix (schéma entité-association, diagramme de classes).

Partie 2 - Gestion de l'authentification

Documents à utiliser : 3

Un stagiaire a commencé à produire le code qui permettra de réaliser l'authentification :
Votre responsable vous demande votre avis sur le code produit :

Question 1.3

Indiquer si le code d'interrogation de la base de données est sûr ou alors expliquez quelles seraient les modifications à apporter dans le cas contraire.

Question 1.4

Conclure sur la façon de stocker les mots de passe au regard du code observé.

Partie 3 - Gestion du changement de formule

Un client doit pouvoir, à tout moment, changer la formule d'un de ses points de vente en fonction de ses besoins.

Dans un souci de traçabilité, il est nécessaire de garder **l'historique des formules souscrites** pour chaque point de vente, dont un changement de formule ne peut être effectué qu'une seule fois par jour. Bien entendu, rien n'interdit de souscrire, pour un point de vente, une formule déjà souscrite par le passé.

Pour gérer l'historique des formules souscrites, votre collègue Gilles Sabatier vous propose d'ajouter une table dans la base de données ayant la description suivante :

FormuleSouscrite(idPointDeVente, idFormule)

clé primaire : idPointDeVente, idFormule

*clés étrangères : idPointDeVente en référence à id de PointDeVente
idFormule en référence à id de Formule*

Question 1.5

Expliquer en quoi la structure de la table ne permettra pas de gérer l'historique des formules souscrites.

Question 1.6

Proposer une correction de la structure de la table qui réponde au besoin exprimé.

Mission 2 : Fidélisation des consommateurs

Partie 1 – Amélioration du module de gestion de la relation client (GRC)

Documents à utiliser : 3, 4, 5, et 6

Le logiciel *WebCaisse* propose, pour certaines formules, l'accès à un module de gestion de la relation client (GRC) disposant de fonctions de gestion des contacts, de rapports d'activité, etc.

Le tableau de bord du module GRC doit être complété pour permettre une analyse rapide des ventes réalisées par les consommateurs.

Les éléments d'information suivants seront ajoutés dans ce tableau de bord :

- a la liste des consommateurs (nom, prénom, adresse de courriel) pour lesquels au moins une vente a été réalisée en 2020 ;
- b le nombre de consommateurs ayant souscrit au programme de fidélité et appartenant à la tranche d'âge 18-30 ans ;
- c la liste des consommateurs (nom, prénom, adresse de courriel) avec le montant total des ventes réalisées pour chacun.

Question 2.1

Écrire les requêtes permettant d'extraire les informations nécessaires de la base de données fournie dans le dossier documentaire.

Le tableau de bord du module GRC est personnalisable : d'autres indicateurs, prédefinis dans le logiciel, peuvent être ajoutés sur ce tableau de bord et certains d'entre eux sont paramétrables par l'utilisateur.

L'un d'eux, l'indicateur "Consommateurs réguliers" permet d'obtenir la liste des consommateurs non encore fidélisés pour lesquels au moins une vente a été réalisée lors des **trente derniers jours**.

La méthode *listeConsoAFideliser* de la classe GRC établit la liste de ces consommateurs.

La fonctionnalité de recherche des consommateurs réguliers existante doit évoluer vers davantage de souplesse en permettant au client de définir une période de recherche, en utilisant un formulaire de saisie des critères de recherche. Une maquette de ce formulaire est fournie dans le dossier documentaire.

Ce formulaire n'est pas sécurisé pour le moment, dans la mesure où aucun contrôle des saisies utilisateur n'est réalisé pour le moment.

Vous décidez de contrôler les saisies grâce à des expressions régulières (rationnelles)

Question 2.2

Proposer les expressions régulières nécessaires au contrôle des différentes saisies du formulaire.

BTS Services informatiques aux organisations E6 : Cybersécurité	Code : E6SLAM	2022 Page 6/22
--	---------------	-------------------

Partie 2 – Réalisation de tests unitaires pour la méthode *AddFidelite()*

Documents à utiliser : 7 et 10

Le module de gestion des ventes de la *WebCaisse* gère le programme de fidélisation pour chaque point de vente :

- Pour le programme de type 1 (tampon), le module de gestion des ventes permet de compter le nombre de visites d'un consommateur. À chaque achat, le consommateur gagne un tampon. Il peut bénéficier d'une réduction à partir d'un nombre de tampons paramétré dans *WebCaisse* pour le point de vente.
- Pour le programme de type 2 (montant achat), le module de gestion des ventes totalise le total dépensé par chaque consommateur. Lorsque le montant total des achats atteint un seuil paramétré pour le point de vente dans *WebCaisse*, le consommateur peut bénéficier d'une réduction.
- Pour le programme de type 3 (points), le module de gestion des ventes attribue un nombre de points fidélité lors de chaque achat du consommateur en fonction de la tranche d'achat :
 - 10 points (montant d'achat compris entre 100 € et 200 € inclus) ;
 - 20 points (montant d'achat supérieur à 200 € jusqu'à 500 €) ;
 - 50 points (montant d'achat supérieur à 500 €).Lorsque le nombre de points atteint un seuil paramétré pour le point de vente dans *WebCaisse*, le consommateur peut bénéficier d'une réduction.

La réduction associée au programme de fidélisation est également paramétrée dans *WebCaisse* pour chaque point de vente.

La méthode *addFidelite()* de la classe *ConsoFidele*, fournie dans le dossier documentaire, est appelée à l'issue de chaque achat d'un consommateur inscrit au programme de fidélisation. Elle permet de mettre à jour l'attribut *pointsFidelite* de la classe *ConsoFidele* en fonction du programme de fidélisation choisi pour le point de vente.

La classe de test *TestPointsFidelite*, fournie dans le dossier documentaire, a été créée afin de valider la méthode *addFidelite*.

Certains tests proposés dans cette classe ne sont pas terminés. Le test *testInitConso()* ne traite pas le cas de la valeur initiale des points de fidélité, qui devrait être fixée à zéro lors de la création d'un consommateur fidèle.

Question 2.3

Compléter la méthode *testInitConso* permettant de combler ce manque.

Votre chef de projet vous demande de compléter le test qui traite le cas du programme de fidélisation par points (type 3).

Question 2.4

Compléter la méthode *testAddMontant* permettant de valider les points de fidélité obtenus dans le cas d'un de programme de fidélisation par points.

BTS Services informatiques aux organisations	2022
E6 : Cybersécurité	Code : E6SLAM

Mission 3 : Statistiques de ventes

Documents à utiliser : 9, 11 et 12

Le module de statistiques du logiciel *WebCaisse* permet d'obtenir des statistiques sur les ventes par l'intermédiaire d'une classe *Statistique* qui contient plusieurs méthodes.

Une méthode de cette classe nommée *statVente* n'a pas été documentée lors de son écriture et votre chef de projet souhaite que vous fassiez le nécessaire.

Question 3.1

Rédiger le commentaire de la méthode *statVente* de la classe *Statistique* expliquant ce qu'elle retourne.

Le responsable de Nasdy souhaite ajouter des statistiques permettant de comparer les ventes en magasin aux ventes en ligne, réalisées sur le site e-commerce.

Dans un premier temps, on vous demande d'implémenter et compléter certaines méthodes.

Question 3.2

Écrire la méthode *getNbVentes* de la classe *Conso* qui retourne le nombre de ventes enregistrées dans la collection des ventes du consommateur.

Question 3.3

Écrire le constructeur de la classe *VenteEcommerce* qui permet d'initialiser tous les attributs d'une instance de la classe.

Une des statistiques devra permettre de visualiser rapidement quel canal de vente est le plus utilisé par les clients inscrits au programme de fidélisation, appelés clients fidèles, en comparant le montant total des ventes réalisées en ligne au montant total des ventes réalisées en magasin. Pour cela, l'application utilisera la méthode *compareLieuVente* de la classe *Statistique* qui retournera le montant total des ventes réalisées en magasin divisé par le montant total des ventes réalisées en ligne.

Question 3.4

Compléter le code de la méthode *compareLieuVente*.

Question 3.5

Expliquer en quoi la dernière instruction "**return totalMag/totalEcom**" de la méthode *compareLieuVente* peut poser problème.

On souhaite ajouter à la classe *Conso* une méthode qui retourne, pour le consommateur, la liste des ventes d'un montant supérieur à un montant passé en paramètre.

Question 3.6

Écrire la méthode répondant à ce besoin.

Mission 4 : Mise en production

Partie 1 - Sécurisation de l'accès au SGBD

Document à utiliser : 13

Le document XX montre l'infrastructure qui a été imaginée pour déployer le service Web en interne.

Question 4.1 :

Où devrait être placé le SGBD pour un maximum de sécurité ? Justifiez.

Les commandes SQL qui permettent de créer le compte d'accès à la base de données par l'application ont été préparées par un stagiaire. Il vous montre ce code :

```
CREATE USER acces_webcaisse WITH PASSWORD 'xxxx';
GRANT ALL ON webcaisse_db TO acces_webcaisse;
```

Question 4.2 :

Critiquer le code et proposez les modifications qui permettraient d'améliorer la sécurité.

Partie 2 - Choix de l'hébergeur

Document à utiliser : 14

Finalement le DSI décide que l'application sera déployée chez un hébergeur.

Le chef de projet vous a adressé un courriel contenant ses attentes en termes de protection des données et les spécifications de trois hébergeurs. Il vous confie l'étude des solutions d'hébergement.

Question 4.3

Rédiger une courte note à destination du chef de projet en justifiant le choix d'un hébergeur parmi les trois propositions.

BTS Services informatiques aux organisations E6 : Cybersécurité	Code : E6SLAM	2022 Page 9/22
--	---------------	-------------------

Dossier documentaire

Document 1 : schéma complet de la base de données utilisée par l'application actuelle AchatWebCaisse

Schéma conceptuel des données

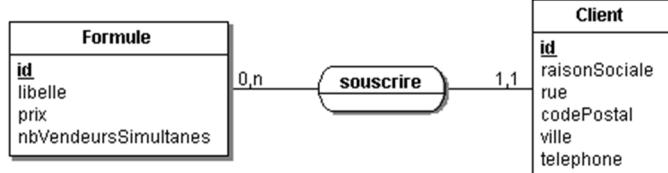


Diagramme de classes

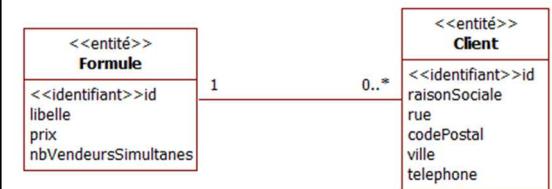


Schéma relationnel :

Formule (id, libelle, prix, nbVendeursSimultanées)

clé primaire : id

Client (id, raisonSociale, rue, codePostal, ville, telephone, idFormule)

clé primaire : id

clé étrangère : idFormule en référence à id de Formule

Document 2 : maquette du formulaire "Choix de formule pour un point de vente"

Nom du client : Société A. Boumbe
 Choix de formule pour le point de vente "Chez Faly" situé à Le Marin :

FORMULE ->	Silver	Gold	Platine
Nombre maximum de vendeurs simultanés	5	15	500
Modules proposés dans chaque formule			
Module gestion des ventes	✓	✓	✓
Module gestion des stocks	✓	✓	✓
Module statistique	✓	✓	✓
Module campagne publicitaire		✓	✓
Module aide à la décision			✓
Module GRC (gestion de la relation client)			✓
Module e-commerce(**)			✓
Prix de la formule	50€/mois	90€ /mois	120 €/mois
Cochez la case de la formule retenue pour le point de vente	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

(**) : + 2,8% du montant des ventes en ligne au titre du module e-commerce

Vos points de vente et leurs formules :

Point de vente	Adresse du point de vente	Formule
Salon Hair Styl	31, rue de la sérénité – 97224 Ducos	Gold
CoupCoif	3, rue du clou - 97200 Fort de France	Silver
Chez Faly	6, rue du flambeau – 97217 Le Marin	En attente de souscription

Document 3 : ébauche du code de vérification de l'authentification

```
Connection con = DriverManager.getConnection(url, loginDB, passDB);
String req = "SELECT COUNT(*) FROM client WHERE identifiant="
+ identifiantATester
+ "" AND motPasse=\"" + Hash.SHA256(passATester) + "\" ;

ResultSet resultat;
try {
    Statement stmt = con.createStatement();
    resultat = stmt.executeQuery(req);
    if (resultat.getInt(1) == 1) {
        //Authentification réussie > aller vers page demandée
    } else {
        //Erreur d'authentification > aller vers page erreur
    }
}
catch(Exception e){
    //gestion des exceptions
}
```

Remarque : La méthode statique Hash.SHA256("mot") produit un hash de "mot" en utilisant l'algorithme cryptographique SHA256.

Document 4 : maquette du formulaire "Recherche des consommateurs réguliers"

Paramétriser les critères de recherche des consommateurs réguliers pour le point de vente *CoupCoif*

Vos critères

Nombre de ventes minimum

du (jj/mm/aa) au (jj/mm/aa)

Document 5 : extrait du schéma relationnel pour la gestion des consommateurs fidèles

Conso(id, nom, prenom, mail, tel,...)

clé primaire : id

ConsoFidele(id, dateNaiss, nbPoints, dateInscription)

clé primaire : id

clé étrangère : id en référence à id de Conso

Vente(id, dateVente, montantVente, idConso,...)

clé primaire : id

clé étrangère : idConso en référence à id de Conso

Remarques :

- La table ConsoFidele ne contient que les consommateurs ayant adhéré au programme de fidélité.
- La fonction *TIMESTAMPDIFF(year, uneDate1, uneDate2)* retourne la différence en valeur absolue et en nombre d'années entre uneDate1 et uneDate2.
- La fonction *CURDATE()* retourne la date du jour.
- La fonction *YEAR(uneDate)* retourne l'année de la date passée en paramètre.

Document 6 : Indications relatives aux expressions régulières

Opérateur	Description
[]	Indique une classe de caractères et fait correspondre un seul des caractères listés entre les crochets [et].
-	Placé dans une classe de caractère, ce signe définit un intervalle de caractères valides.
[^]	Un ^ au début d'une classe de caractères entre crochets signifie qu'on considère le complément de cette classe (l'ensemble des caractères qui ne sont pas listés dans la classe). Ailleurs dans la classe de caractères, il est considéré comme un caractère normal.
^	Ce caractère, placé au début d'une expression rationnelle, impose que l'expression cherchée commence en début d'une ligne . Ailleurs il est considéré comme un caractère normal.
\$	Ce caractère, placé à la fin d'une expression rationnelle, impose que l'expression cherchée termine en fin d'une ligne . Ailleurs il est considéré comme un caractère normal.
.	Ce <i>joker</i> , qui n'est reconnu comme tel qu'en dehors d'une classe de caractères, est une classe de caractères particulière et correspond à un caractère unique quelconque (sauf les caractères de fin de ligne, et les caractères de contrôle en dehors de la tabulation).
	L'opérateur de choix (aussi appelé opérateur d'alternation ou d'union d'ensembles) fait correspondre l'expression avant ou après l'opérateur.
()	Les parenthèses de groupement permettent de délimiter un ensemble d'alternatives ou toute sous-expression rationnelle (à l'exception des conditions de début et fin de ligne) pour leur appliquer un quantificateur.
*	Ce quantificateur fait correspondre zéro, une ou plusieurs fois ce qui précède.
+	Ce quantificateur (dans une classe il désigne littéralement le caractère correspondant), fait correspondre une ou plusieurs fois l'élément qui le précède. Contrairement à l'opérateur standard, l'élément qui précède peut être aussi un groupe.
?	Ce quantificateur fait correspondre zéro ou une seule fois ce qui le précède. L'élément précédent peut être un caractère littéral, ou une séquence d'échappement, ou une classe, ou un joker.
{m, n}	Cela crée un quantificateur borné, permettant de faire correspondre exactement de <i>m</i> à <i>n</i> occurrences de ce qui précède, <i>m</i> et <i>n</i> étant deux entiers tels que <i>m</i> < <i>n</i> . Chacun des deux paramètres peut être omis : si <i>m</i> est omis, il prend la valeur par défaut 0 ; si <i>n</i> est omis, mais la virgule est présente, il est considéré comme infini ; si le second paramètre <i>n</i> est omis ainsi que la virgule séparatrice, il prend la valeur par défaut égale au premier paramètre <i>m</i> .

Document 7 : extrait du diagramme de classes utilisé par le module GRC

```

class Conso {
    private String nom;
    private String prenom;
    private String mail;
    private String tel;
    public Conso (String nom, String prenom,
                  String mail, String tel) {...}
    public boolean estFidele() {
        return false;
    }
}
class ConsoFidele extends Conso {
    private Date dateNaiss;
    private double pointsFidelite; // contient un nombre de points ou un montant
    private Date dateInscription;
    public ConsoFidele(String nom, String prenom,
                      String mail, String tel, Date dateNaiss, Date dateInsc) {...}

    @override
    public boolean estFidele() {
        return true;
    }
    public void addFidelite(int typeFidelite, double montant) {
        // typeFidelite contient le type de programme de fidélisation choisi (1, 2 ou 3)
        // montant contient la valeur du montant d'achat réalisé
        switch (typeFidelite) {
            case 1 : this.pointsFidelite++;
            break;
            case 2 : this.pointsFidelite += montant;
            break;
            case 3 : if (montant >=100 && montant <=200)
                this.pointsFidelite += 10;
                else if (montant >200 && montant <=500)
                    this.pointsFidelite += 20;
                else if (montant > 500)
                    this.pointsFidelite += 50;
            break;
        }
    }
    public double getPointsFidelite() {...}
}

```

```

classDiagram
    class Conso {
        -nom
        -prenom
        -mail
        -tel
        +estFidele(): bool
    }
    class ConsoFidele {
        -dateNaiss
        -pointsFidelite
        -dateInscription
        +estFidele(): bool
        +getPointsFidelite(): double
        +addFidelite(typeFidelite:int, montant:double)
    }
    Conso <|-- ConsoFidele

```

Document 8 : extrait de la classe GRC contenant la méthode listeConsoAFideliser

```
class GRC {  
    public static ArrayList<Conso> listeConsoAFideliser() {  
        ArrayList <Conso> lesConsos = new ArrayList <Conso>();  
        //Connexion à la base de données  
        Connection dbConnect = DriverManager.getConnection("jdbc:mysql:"  
                + dbURL, adminUser, adminPwd);  
        Statement dbStatement = this.dbConnect.createStatement();  
  
        //Création de la requête  
        String requete="select distinct nom, prenom, tel, mail"  
                + " from Conso "  
                + "join Vente on idConso = Conso.id"  
                + " where Conso.id not in (select id from Consommateurs)"  
                + " and datediff(now(), dateVente)<30" ;  
  
        //Exécution de la requête  
        ResultSet res = this.dbStatement.executeQuery(requete);  
        //Pour chaque enregistrement résultat de la requête  
        while (res.next()) {  
            //Création d'un objet de la classe Conso avec les données  
            Conso leConso = new Conso(res.getString(1), res.getString(2) ,  
                    res.getString(3) , res.getString(4)) ;  
            //ajout du Consommateur à la liste des Consommateurs  
            lesConsos.add(leConso) ;  
        }  
        // fermeture de la connexion à la BD  
        res.close();  
        dbStatement.close() ;  
        dbConnect.close() ;  
        return lesConsos ;  
    }  
}
```

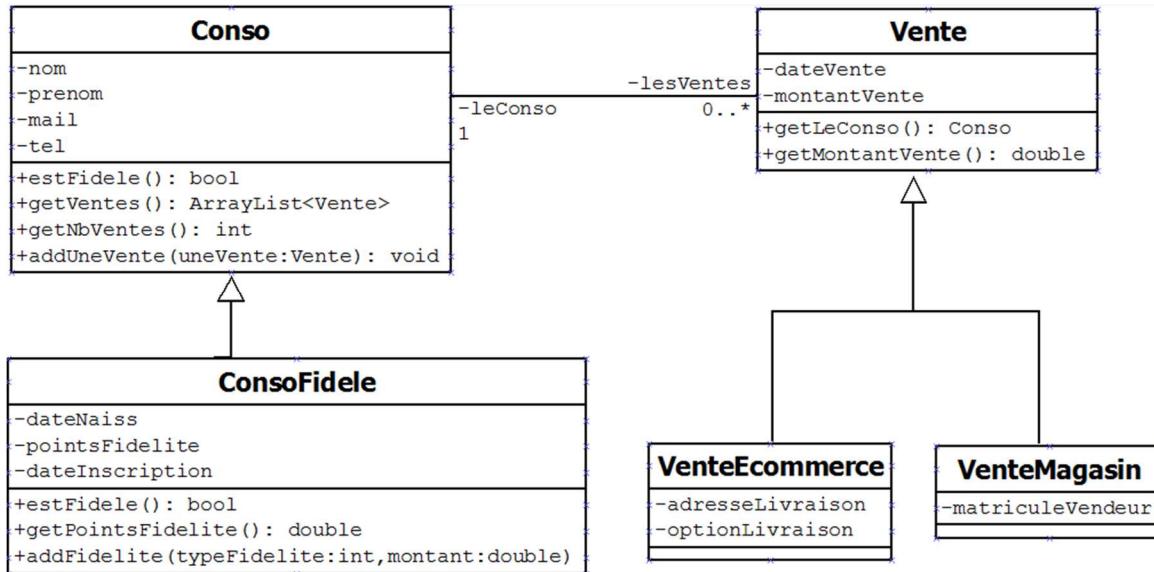
Document 9 : exemple d'utilisation d'une collection ArrayList

```
ArrayList<Vente> lesVentes ; // Déclaration d'une collection d'instances de la classe Vente  
lesVentes = new ArrayList<Vente>() ; // Instanciation d'une collection  
Vente nouvVente = new Vente (...);  
lesVentes.add(nouvVente) ; // Ajout d'un élément dans la collection  
lesVentes.get( 0 ) ; // accès au premier élément  
lesVentes.size() ; // accès au nombre d'éléments enregistrés dans la collection  
foreach( Vente uneVente : lesVentes) {  
    // Parcours de la collection  
}
```

Document 10 : classe de test de la méthode *addFidelite*

```
class TestPointsFidelite extends TestCase {  
    public void testAddFideliteTampon() {  
        ConsoFidele consoTest = new ConsoFidele("Lifo", "Paul",  
            "lifo.paul@gmail.com", "0600000000",  
            new SimpleDateFormat("yyyy-MM-dd").parse("1961-01-03"),  
            new SimpleDateFormat("yyyy-MM-dd").parse("2017-01-05"));  
        consoTest.addFidelite(1, 50);  
        assertEquals("erreur calcul 1er tampon", 1, consoTest.getPointsFidelite());  
        consoTest.addFidelite(1, 20);  
        assertEquals("erreur calcul 2ème tampon ", 2, consoTest.getPointsFidelite());  
    }  
  
    public void testAddFideliteMontant() {  
        ConsoFidele consoTest = new ConsoFidele("Lifo", "Paul",  
            "lifo.paul@gmail.com", "0600000000",  
            new SimpleDateFormat("yyyy-MM-dd").parse("1961-01-03"),  
            new SimpleDateFormat("yyyy-MM-dd").parse("2017-01-05"));  
        consoTest.addFidelite(2, 150);  
        assertEquals("erreur calcul 1er achat", 150, consoTest.getPointsFidelite());  
        consoTest.addFidelite(2, 250);  
        assertEquals("erreur calcul 2ème achat", 400, consoTest.getPointsFidelite());  
    }  
  
    public void testInitConso() {  
        ConsoFidele consoTest = new ConsoFidele("Lifo", "Paul",  
            "lifo.paul@gmail.com", "0600000000",  
            new SimpleDateFormat("yyyy-MM-dd").parse("1961-01-03"),  
            new SimpleDateFormat("yyyy-MM-dd").parse("2017-01-05"));  
        // À compléter sur votre copie  
    }  
  
    public void testAddMontant() {  
        ConsoFidele consoTest = new ConsoFidele("Lifo", "Paul",  
            "lifo.paul@gmail.com", "0600000000",  
            new SimpleDateFormat("yyyy-MM-dd").parse("1961-01-03"),  
            new SimpleDateFormat("yyyy-MM-dd").parse("2017-01-05"));  
        // À compléter sur votre copie  
    }  
}
```

Document 11 : extrait du diagramme de classes utilisé par le module de statistiques



```

class Conso {
    private String nom;
    private String prenom;
    private String mail;
    private String tel;
    private ArrayList<Vente> lesVentes ;
    public Conso(String nom, String prenom, String mail, String tel){...}
    public boolean estFidele() { return false; }
    public void addUneVente(Vente uneVente) {...}
    // retourne le nombre de ventes enregistrées dans la collection des ventes du consommateur
    public int getNbVentes() { ... À compléter sur votre copie ...}
    // retourne la collection lesVentes qui contient des instances des
    // classe VenteEcommerce et VenteMagasin
    public ArrayList<Vente> getVentes() { ... }
}

class ConsoFidele extends Conso { // extends met en oeuvre le mécanisme d'héritage en Java
    private Date dateNaiss;
    private double pointsFidelite;
    private Date dateInscription;
    public ConsoFidele(String nom, String prenom, String mail, String tel,
                       Date dateNaiss, Date dateInsc){...}
    @override
    public boolean estFidele() { return true; }
    public void addFidelite(int typeFidelite, double montant) { ... }
    public double getPointsFidelite() { ... }
}
  
```

```

class Vente {
    private Date dateVente ;
    private Conso leConso ;
    private double montant ;

    // constructeur qui valorise les attributs d'un objet de la classe Vente
    public Vente (Date uneDateVente, Conso unConso, double unMontant) {...}
    public double getMontantVente(){...}
    public Conso getLeConso(){...}

}

// extends met en oeuvre le mécanisme d'héritage en Java
class VenteEcommerce extends Vente {
    private String adresseLivraison ;
    private String optionLivraison ;
    public VenteEcommerce(...À compléter sur votre copie ...){
        ...À compléter sur votre copie ...
    }
}

// extends met en oeuvre le mécanisme d'héritage en Java
class VenteMagasin extends Vente {
    private String matriculeVendeur;
    ....
}

```

Document 12 : extrait de la classe Statistique utilisée par le module de statistiques

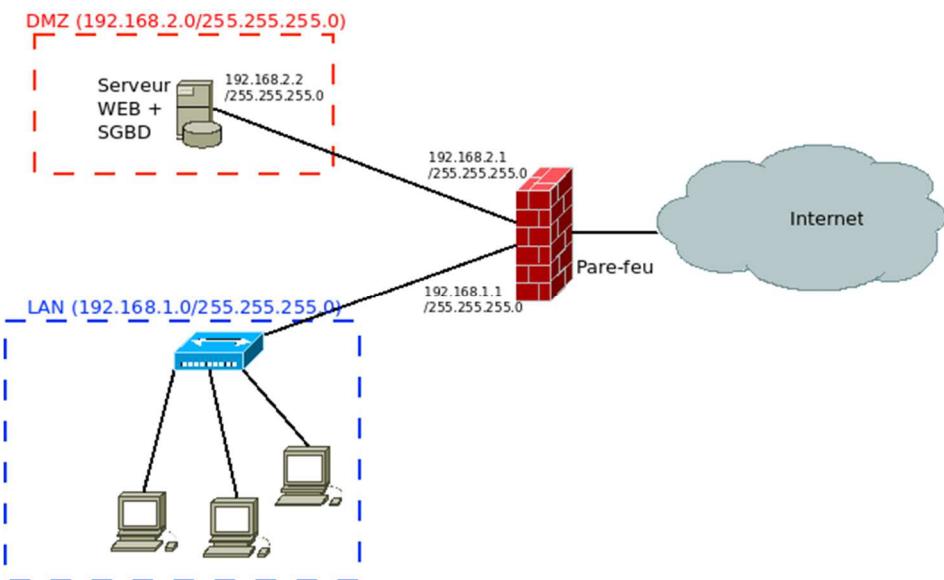
```
public class Statistique {  
  
    // méthode statVente à commenter  
  
    public static double statVente(ArrayList<Vente> lesVentesDuJour) {  
        int nbVenteFidele=0;  
        foreach(Vente uneVente : lesVentesDuJour) {  
            Conso c = uneVente.getLeConso();  
            if (c.estFidele())  
                nbVenteFidele++;  
        }  
        return nbVenteFidele*100/lesVentesDuJour.size();  
    }  
  
    // méthode compareLieuVente  
  
    public static double compareLieuVente(ArrayList<ConsoFidele> lesConsos) {  
        double totalEcom=0; //cumul des montants des ventes ecommerce  
        double totalMag=0; // cumul des montants des ventes en magasin  
        //parcours de la liste des consommateurs fidèles  
        foreach(ConsoFidele cf : lesConsos) {  
  
            ... À compléter sur votre copie ...  
  
        }  
        return totalMag/totalEcom ; //calcul de l'indice et retour du résultat  
    }  
    ...  
}
```

Remarque :

instanceof permet de connaître le type d'instance d'un objet.

```
if (uneVente instanceof Vente) {  
  
    // traitement effectué si uneVente est instance de la classe Vente  
}
```

Document 13 : schéma de l'infrastructure envisagée



Document 14 : choix d'un hébergeur répondant à la certification NF525

Contenu du courriel qui vous est adressé par Renaud Boileau

Afin de permettre à l'application **WebCaisse** d'obtenir la certification NF 525, nous devons être vigilants à propos de la sécurité des données.

Nous avons besoin d'assurer à nos clients :

- une sauvegarde quotidienne des données du portail ;
- de la haute disponibilité (réplication de l'application pour éviter un déni de service) ;
- une sécurisation des échanges et des données par chiffrement.

Voici en pièce jointe les spécifications de trois hébergeurs qui entrent dans notre budget et extraites de leurs sites internet. Merci de me transmettre ton analyse au plus tôt.

Cordialement,
Renaud Boileau

Pièce jointe n°1 : hébergeur A

Tous les serveurs sont localisés à Roubaix, dans le Nord de la France. Le site client bénéficie donc d'une adresse IP française dédiée et d'une bande passante garantie.

Le support technique est disponible 24h/24 via quatre moyens de communication : courriel, ticket, aide en ligne ou *Skype*.

La disponibilité du service est garantie à 99,9% : en cas d'indisponibilité supérieure à 43 minutes sur un mois, une assurance indemnisera le client à hauteur des préjudices subis.

Un délai d'intervention maximum de 4 heures est garanti.

Les données du client bénéficient de deux sauvegardes : une sauvegarde en temps réel et une sauvegarde effectuée tous les lundis soir.

Différents tutoriels sont disponibles afin de faciliter la prise en main de l'outil d'administration *cPanel* qui permet de gérer l'hébergement web.

L'acquisition d'un certificat *SSL* (*Secure Sockets Layer*) est possible afin de garantir des échanges sécurisés.

Pièce jointe n°2 : hébergeur *B*

Le client bénéficie d'un serveur dédié physique dont les ressources *CPU* et mémoire lui sont propres. Il a un contrôle total sur son serveur qui lui permet d'installer les logiciels de son choix.

La haute-disponibilité est garantie à hauteur de 99,9% dans un *datacenter Tier IV* ultra sécurisé et un réseau redondant.

283 extensions de nom de domaine sont proposées. Le nom de domaine est inclus dans l'offre.

Un outil d'administration convivial est disponible pour gérer l'hébergement internet.

La sauvegarde des données est réalisée quotidiennement. La restauration est possible d'un seul clic.

Pièce jointe n°3 : hébergeur *C*

L'hébergement propose un espace disque et une bande passante illimités.

Les supports de l'hébergement sont alimentés à 100% par des sources d'énergie certifiées vertes et renouvelables.

De nombreux scripts sont mis à disposition pour faciliter l'administration de l'hébergement. Des outils de gestion de contenu (*CMS*) tels que *WordPress*, *Joomla!*, *Drupal* et bien d'autres sont disponibles sur la plateforme.

La disponibilité est garantie à 99,9 %.

Des systèmes RAID créent des doubles de chaque serveur pour assurer leur disponibilité. Par ailleurs, le client a accès à des sauvegardes manuelles gratuites avec *cPanel*.

BTS Services informatiques aux organisations E6 : Cybersécurité	Code : E6SLAM	2022 Page 22/22
--	---------------	--------------------