BREVET DE TECHNICIEN SUPÉRIEUR

SERVICES INFORMATIQUES AUX ORGANISATIONS

Option : Solutions logicielles et applications métiers

U6 – CYBERSÉCURITÉ DES SERVICES INFORMATIQUES

SESSION 2022

_____

Durée: 4 heures
Coefficient: 4

_____

Matériel autorisé :
Aucun matériel ni document est autorisé.

Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le sujet comporte 21 pages, numérotées de 1/21 à 21/21
(sans compter la page de garde).

Cas Denas

Ce sujet comporte 21 pages dont un dossier documentaire de 12 pages.
La candidate ou le candidat est invité(e) à vérifier qu'il est en possession d'un sujet complet.

Barème

DOSSIER A	Participation à l'atelier d'analyse de risque	12 points
DOSSIER B	Amélioration de l'authentification	26 points
DOSSIER C	Validation des demandes client	24 points
DOSSIER D	Envoi des données de demande au progiciel de gestion intégrée (*PGI*)	18 points
	TOTAL	80 points

Dossier documentaire

Documents associés au dossier A....10
Document A1 : Diagramme de cas d'utilisation de l'application PartEdge....10
Document A2 : Besoins de sécurité liés aux cas d'utilisation....10
Document A3 : Événements redoutés et leurs impacts....11
Document A4 : Scénarios de risques et mesures à prévoir....11
Documents associés au dossier B....12
Document B1 : Clé OTP (One Time Password)....12
Document B2 : Formulaire contenu dans la vue loginView.php....13
Document B3 : Classe contrôleur LoginCtrl....13
Document B4 : Classe DAO ClientDao.php....13
Document B5 : Exemple d'utilisation de requête préparée en PHP....14
Document B6 : Utilisation de journaux syslog en PHP....14
Documents associés au dossier C....15
Document C1 : Extrait du schéma relationnel de la base de données PartEdge....15
Document C2 : Utilisation de la clause INTERVAL et de la fonction NOW() en SQL....15
Document C3 : Exemple de déclencheur (*trigger*)....16
Document C4 : Schémas de modélisation de la base de données PartEdge....16
Documents associés au dossier D....18
Document D1 : Interface REST en JAVA EE....18
Document D2 : Classe API REST DemandeRest.java....19
Document D3 : Classes DemandeJson et DemandeService....19
Document D4 : Tests unitaires Junit et classe DemandeServiceTest....21

Présentation du contexte

L'organisation du groupe Denas

Depuis plus de 20 ans, le groupe Denas s'est spécialisé en électronique dans les domaines du transport aérien civil.

Cette organisation a été créée en 1980 par trois ingénieurs chevronnés issus d'entreprises renommées de l'aéronautique française. Grâce à leur expérience, ils ont su proposer des solutions innovantes qui leur ont permis de s'imposer dans le domaine de l'aviation civile. Aujourd'hui, un aéronef sur trois dans le monde utilise les technologies du groupe Denas. Un aéronef est un appareil capable de se déplacer dans les airs (avion, hélicoptère, aérostat, etc.).

Actuellement, 15 000 personnes travaillent pour le groupe Denas dont 5 000 en France sur 10 sites. La faiblesse du dollar et la baisse des investissements des compagnies aériennes (-7 % sur le premier semestre 2019) incitent ce groupe à faire évoluer son système d'information et à développer l'activité maintenance. Le site principal est en charge des activités de conception et fabrication des systèmes de navigation, ainsi que de réparation de pièces électroniques et électromécaniques pour l'aéronautique civile uniquement.

Si un aéronef tombe en panne suite à une défaillance d'une pièce fabriquée par le groupe Denas, la personne en charge de la maintenance de l'aéronef sollicite le site principal pour obtenir, de façon urgente, une pièce de rechange. Dans l'industrie aéronautique, la traçabilité de chaque pièce d'aéronef est obligatoire dans le but d'assurer la sécurité des passagers et de l'équipage.

Le site principal utilise un progiciel de gestion intégré (PGI) qui lui permet de gérer les commandes, les stocks de pièces, la facturation et les clients.

La société a besoin d'être présente 24h/24 et 7j/7 auprès des compagnies aériennes pour assurer le bon fonctionnement de leurs aéronefs. Un aéronef cloué au sol coûte très cher et engendre des pertes importantes. C'est pourquoi les compagnies aériennes exigent de leurs fournisseurs une réactivité totale. Le groupe Denas doit donc mettre au point de nouveaux outils numériques.

Le service DigitaLab

Le site principal a inauguré fin septembre 2019 le service DigitaLab, un laboratoire de recherche et de développement de nouvelles solutions numériques. Ce service est constitué d'une équipe de 17 développeurs et d'un collaborateur en charge de l'intégration des développements.

DigitaLab s'appuie sur la méthode agile *Scrum* pour la gestion de ses projets.

DigitaLab produit et maintient un ensemble de logiciels hébergés en partie localement et pour l'autre partie dans le nuage informatique (*cloud*).

Vous rejoignez l'équipe DigitaLab et votre mission consiste à participer au développement d'une nouvelle application PartEdge permettant aux clients de gérer leurs demandes de pièce de rechange qui seront transmises au PGI grâce à une interface de programmation applicative (*API*) de type *REST*.

Développeuse ou développeur spécialiste en cybersécurité, vous devrez :

- participer à l'atelier d'analyse de risque ;
- sécuriser la connexion ;
- sécuriser le processus des demandes de pièces de rechange ;
- finaliser l'interface de programmation applicative (*API*) de type REST.

Vous vous appuierez sur le dossier documentaire mis à votre disposition.

Dossier A – Participation à l'atelier d'analyse de risque

L'objectif de l'application *Web* PartEdge est de permettre le dialogue entre les équipes internes du groupe Denas et les compagnies d'aviation clientes.

En cette période de pandémie, le stock de pièces est très faible car les délais de livraison des fournisseurs se sont allongés.

L'application PartEdge doit permettre la visualisation du stock des pièces disponibles et la gestion des échanges des documents contractuels nécessaires lors de la réparation, l'échange ou la vente de pièces au départ de l'Europe mais aussi des États-Unis et de l'Inde.

L'application sera mise à disposition de 700 clients potentiels, allant de la petite compagnie aérienne régionale à la grande compagnie institutionnelle. Les réparateurs externes spécialisés dans la maintenance des aéronefs civils proposent leurs services pour les petites compagnies aériennes. Ils peuvent eux aussi réserver des pièces grâce à l'application PartEdge afin de réaliser des réparations pour le compte de leurs clients.

L'application PartEdge doit échanger des données avec le *PGI* de l'entreprise :

- le *PGI* actualise régulièrement la base de données utilisée par l'application PartEdge ;
- l'application PartEdge transmet les caractéristiques des commandes au PGI au fur et à mesure de leur saisie.

L'application est actuellement en début de développement et un atelier d'analyse de risque a été initié par votre équipe.

Mission A1 – Les enjeux de sécurité

Les différents cas d'utilisation ont été répertoriés dans le document "Diagramme de cas d'utilisation de l'application PartEdge".

Chaque cas d'utilisation engendre des besoins de sécurité spécifiques, ébauchés dans le document "Besoins de sécurité liés aux cas d'utilisation".

Question A1.1
Proposer une évaluation des quatre critères de sécurité pour les cas d'utilisation n°2 et 6.

Mission A2 – Les événements redoutés et les mesures à prendre

Le tableau "Événements redoutés et leurs impacts" présente le degré d'impact au niveau métier d'une malveillance anticipée.

Question A2.1
Évaluer l'impact métier correspondant aux événements redoutés n°4 et 5.

Le tableau "Scénarios de risques et mesures à prévoir" met en place des scénarios correspondants aux événements redoutés du tableau précédent.

Question A2.2
Proposer deux mesures à prévoir pour chacun des scénarios de risques n°2 et 5.

Question A2.3
Expliquer ce qui peut amener des clients peu scrupuleux à réaliser les scénarios n° 3 et 4.

Dossier B – Amélioration de l'authentification

L'application PartEdge nécessite que les utilisateurs s'authentifient pour toutes les opérations :

- consulter le catalogue de pièces ;
- opérer une demande de pièces.

Au vu de la confidentialité des données exposées, il est absolument indispensable que l'authentification soit fiable.

Mission B1 – Sécurisation des mots de passe

En l'état actuel du développement, l'authentification est rudimentaire : les mots de passe sont stockés en clair dans la base de données. Deux solutions sont envisagées :

- solution 1 : mots de passe chiffrés par chiffrement symétrique, par exemple avec *AES* ;
- solution 2 : mots de passe hachés, par exemple avec *SHA*.

Question B1.1
Choisir la solution qui semble la plus sécurisée en justifiant votre réponse.

Il est décidé d'opter pour un stockage des mots de passe hachés avec la fonction *Bcrypt*.

Question B1.2
Expliquer l'importance d'ajouter un grain de sel (salage) lors de l'opération de hachage.

Mission B2 – Prévention des injections de code *SQL*

La méthode d'interrogation de la base de données laisse la possibilité de pratiquer des injections de code *SQL*. La parade habituelle consiste à exécuter des requêtes préparées.

Question B2.1
Modifier le code de la méthode *readByLogin* de la classe ClientDao afin d'éviter les injections *SQL*.
Vous indiquerez les numéros de ligne concernées par chaque insertion ou modification du code.

Mission B3 – Authentification à deux facteurs et journalisation des connexions

Dans une application comme PartEdge, qui donne accès à des données d'une grande valeur commerciale et de sécurité, une authentification simple par un couple login / mot de passe semble trop peu sûre. Il a été décidé d'ajouter une authentification à deux facteurs utilisant un système de mot de passe à usage unique (clé *OTP, One Time Password*).

Question B3.1
Proposer à l'équipe de développement deux autres solutions d'authentification à deux facteurs, sans préciser la mise en œuvre.

Question B3.2
Modifier le code de la vue *loginView.php* pour permettre la saisie du mot de passe à usage unique fournie par la clé *OTP*.

Le code actuel ne prend pas en charge ni l'authentification à deux facteurs par clé *OTP* ni la journalisation des connexions indispensable au suivi des accès à l'application. Il sera nécessaire d'adapter la méthode *verifLogin* de la classe LoginCtrl afin :

- de vérifier le code fourni par la clé *OTP* saisi ;
- d'enregistrer dans un fichier journal (*logs*) toutes les tentatives de connexion à l'application :
  - Si la connexion réussit, l'enregistrement aura les caractéristiques suivantes :

Valeur du paramètre priority	Structure du message
LOG_INFO	"Connexion client <id du client>"

Par exemple : Connexion client 1287

  - Si la connexion échoue, le message est de type de la forme :

Valeur du paramètre priority	Structure du message
LOG_WARNING	"Erreur connexion client <login >, cause : <type>"

Par exemple : Erreur connexion client Az26Tk, cause : Erreur *OTP*

<login> est l'identifiant saisi sur le formulaire de connexion.

<type> peut être :
- "Erreur login/mot de passe" si le couple login/mot de passe n'est pas valide ;
- "Erreur OTP" si le mot de passe est valide mais que la vérification *OTP* a échoué.

Question B3.3
Écrire le code nécessaire dans la méthode *verifLogin* de la classe LoginCtrl pour vérifier la clé *OTP* saisie et pour écrire dans le fichier journal (*logs*).
Vous indiquerez les numéros de ligne où insérer le code.

Dossier C – Validation des demandes client

Lors d'une demande de pièces sur l'application PartEdge le client reçoit un ensemble de documents techniques ou administratifs dont le contrat qui devra être signé pour valider la demande.

Mission C1 – Envoi des documents au client et signature du contrat

Actuellement, l'envoi des documents est fait par courrier postal recommandé. L'objectif est de générer et d'envoyer automatiquement par messagerie électronique ces documents suite à l'enregistrement de la demande du client.

Question C1.1

Indiquer la condition nécessaire pour que cette solution d'échange électronique soit recevable devant les tribunaux en cas de contestation par le client.

Le service juridique du groupe Denas vous sollicite dans le cadre des échanges des documents, entre le groupe Denas et le client, et impose les quatre exigences suivantes :

1. une totale confidentialité des échanges ;
2. la preuve que les documents sont authentiques et non modifiés ;
3. la preuve que le client a bien reçu les documents ;
4. la signature du client qui l'engage légalement.

Après étude, il apparait qu'aucune de ces quatre exigences n'est gérée de façon satisfaisante par la messagerie électronique.

Question C1.2

Indiquer les solutions techniques à mettre en œuvre pour répondre à chacune des quatre exigences. Chaque exigence devra être traitée de façon indépendante.

Mission C2 – Gestion des clients abusifs

Lorsqu'un client réalise une demande de pièce, il doit indiquer le type de pièce dont il a besoin et le modèle de l'aéronef concerné par la demande. Le service en charge de la livraison des pièces affecte immédiatement à cette demande une pièce présente dans le stock. Le client doit ensuite confirmer sa demande en signant un contrat. Une demande non confirmée dans les 24 heures est appelée demande abusive.

L'atelier d'analyse de risque a mis en évidence deux scénarios de risques qui nécessitent de prendre les mesures suivantes contre les clients abusifs pratiquant la cyber-influence :

- bloquer les clients qui ne confirment pas les demandes de pièces dans les 24 heures ;
- vérifier la compatibilité d'une demande de pièce avec le modèle de l'aéronef concerné.

Dans un premier temps, on souhaite connaître les clients potentiellement en position d'abus.

Question C2.1
Écrire la requête qui permet d'obtenir la liste des clients ayant des demandes de plus de 24 heures non confirmées. Pour chaque client on souhaite afficher l'identifiant, le nom et le nombre total de pièces demandées.

Pour contrer ces clients peu scrupuleux, il a été décidé d'automatiser la procédure de surveillance.

Pour protéger le stock, avant qu'une demande ne soit enregistrée dans la base de données, on doit vérifier que le client n'est pas en position d'abus. La fonction *nb_abus(idClient)* qui retourne le nombre de demandes abusives pour un client donné a été codée et est utilisable au sein du *SGBD*. Le traitement prévu est de bloquer le client, en passant l'attribut booléen *estBloque* à la valeur *true*, et d'empêcher la création de la nouvelle demande. Un client est bloqué si le nombre d'abus est égal à 2.

Question C2.2
Écrire le code du déclencheur (*trigger*) qui permet d'obtenir cette fonctionnalité.

On constate que certains clients ont tendance à demander davantage de pièces de certains types de façon à gêner des concurrents. En effet l'immobilisation d'un aéronef engendre des pertes financières pouvant aller jusqu'à 25 000 € par jour pour un modèle 777-300ER du constructeur *Boeing*, sans compter l'annulation des billets.

Actuellement, un client peut demander un type de pièce sans relation avec le modèle de l'aéronef concerné par la demande. Le service en charge de la livraison des pièces affecte une pièce à cette demande mais n'a, à ce jour, aucune possibilité de vérifier dans la base de données qu'un type de pièce est bien compatible avec un modèle d'aéronef.
Un type de pièce est en général prévu pour être monté sur plusieurs modèles d'aéronef.
Un commentaire sur chaque compatibilité d'un type de pièce avec un modèle d'aéronef sera utile pour affiner les recherches.

Question C2.3
Adapter la modélisation des données existante pour intégrer la compatibilité des types de pièce avec les modèles d'aéronef.

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

Il faudra donc vérifier, au moment de la saisie de la demande, que le type de pièce demandé est bien compatible avec le modèle d'aéronef concerné.

Question C2.4
Proposer une solution pour implémenter cette contrainte métier, sans la mettre en œuvre.

Dossier D – Envoi des données de demande au progiciel de gestion intégré (*PGI)*

Les utilisateurs internes du groupe Denas utilisent principalement un progiciel de gestion intégré (*PGI*) pour la gestion des demandes et toutes les fonctions traditionnelles de ce type de progiciel.

Actuellement, le processus de demande de type de pièces est le suivant :

1. Le client demande la pièce dont il a besoin.
2. Les données sont saisies manuellement dans le *PGI*.

Afin de gagner en temps de traitement et en fiabilité, il est prévu que la remontée de demande s'effectue automatiquement, mais sans compromettre la sécurité du *PGI* qui est un élément primordial pour l'entreprise.

Le *PGI* est localisé dans le réseau interne de l'entreprise et l'application PartEdge sera hébergée en ligne (*cloud*) grâce à un service de l'entreprise AWS. Une interface de programmation (*API*) de type *REST* permettra à l'application PartEdge d'envoyer les données de demande dans le *PGI* après vérification et sera localisée dans une zone démilitarisée (*DMZ*).

Mission D1 – Finalisation d'une interface de programmation (*API*) de type *REST*

Cette interface contient actuellement deux méthodes :

- La première traite une requête de type *GET* pour obtenir une demande à partir de son identifiant.
- La deuxième traite une requête de type *PUT* pour mettre à jour les caractéristiques d'une demande dans la base de données.

Pour l'instant, l'interface *REST* ne permet pas de traiter une requête de type *POST* pour créer une demande, aussi, elle doit être modifiée pour le permettre. Si cette requête *HTTP* n'aboutit pas, cela doit se traduire par un retour *HTTP* indiquant que la requête n'a pas pu être réalisée.

Question D1.1
Réaliser la méthode de l'interface de programmation (API) *REST* située dans la classe DemandeRest qui permettra d'ajouter une demande.

Mission D2 – Contrôle des données

La sécurité de cette interface est essentielle, car il s'agit d'une porte ouverte pour d'éventuels attaquants. Les données suivantes transmises doivent être vérifiées :

- Le mode de transport ne peut contenir que les valeurs suivantes : "INTERNE" ou "EXTERNE".
- Le type d'échange des pièces ne peut contenir que les valeurs suivantes : "STANDARD", "SILVER" ou "GOLD".
- La date de la demande doit être au format *ISO* : YYYY-MM-DD.

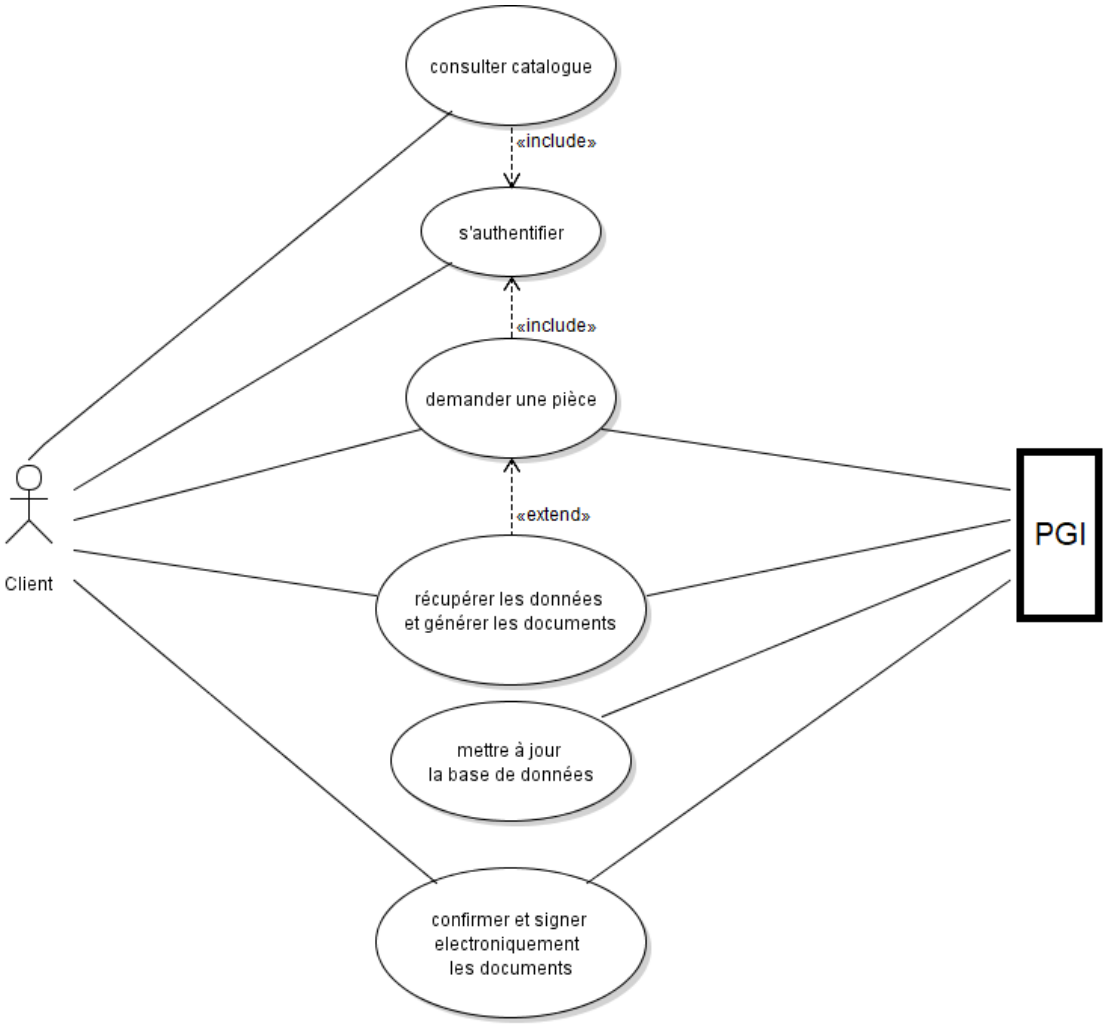
Question D2.1
a) Compléter la méthode *verifDonneesCreate* de la classe DemandeService pour qu'elle contienne les contrôles demandés, en utilisant la grammaire des expressions régulières.
b) Proposer un contrôle supplémentaire permettant de sécuriser davantage la date de la demande. Le code n'est pas demandé.

La réalisation de vérifications automatisées de tests de non-régression lors de la mise à jour du code semble absolument nécessaire pour sécuriser le bon fonctionnement de l'application.

Question D2.2
Compléter le test unitaire manquant de la classe DemandeServiceTest permettant de s'assurer du bon fonctionnement de la méthode *create* de la classe DemandeService.

Documents associés au dossier A

Document A1 : Diagramme de cas d'utilisation de l'application PartEdge



Document A2 : Besoins de sécurité liés aux cas d'utilisation

Cas N°	Cas d'utilisation	Critères de sécurité
1	Un client s'authentifie.	Confidentialité : ++ Intégrité : ++ Disponibilité : ++ Preuve : +
2	Un client consulte le catalogue des pièces.	**** à compléter sur votre copie ****
3	Un client demande une pièce.	Confidentialité : ++ Intégrité : ++ Disponibilité : ++ Preuve : ++
4	Le *PGI* récupère les données de la demande et génère les documents.	Confidentialité : ++ Intégrité : ++ Disponibilité : ++ Preuve : 0
5	Le *PGI* met à jour la base de données de PartEdge.	Confidentialité : ++ Intégrité : ++ Disponibilité : 0 Preuve : 0
6	Le client confirme en signant électroniquement les documents.	**** à compléter sur votre copie ****

0 : pas de besoin + : besoin important ++ : besoin très important

Confidentialité : les informations ne doivent pas être divulguées.
Intégrité : les données doivent être exactes et complètes.
Disponibilité : la fonctionnalité doit être utilisée au moment voulu.
Preuve : les traces de l'activité du système sont opposables en cas de contestation.

Document A3 : Événements redoutés et leurs impacts

Événement N°	Événement redouté	Impact métier	Gravité
1	Un attaquant vole les mots de passe stockés dans la base de données.	La fiabilité des demandes est gravement compromise.	++
2	Un attaquant "devine" un mot de passe.	La fiabilité des demandes est gravement compromise.	++
3	Un utilisateur/client peu scrupuleux saisit des demandes de pièces qu'il ne validera pas ensuite.	Immobilisation temporaire inutile de certaines pièces > coûts élevés pour les avions immobilisés > perte d'image	+
4	Un utilisateur/client peu scrupuleux saisit et valide des demandes de pièces non compatibles avec l'aéronef déclaré en panne.	**** à compléter sur votre copie ****	++
5	Un attaquant accède aux données du *PGI* en utilisant de façon inappropriée l'interface d'une demande de pièce.	**** à compléter sur votre copie ****	++

+ : gravité modérée ++ : gravité très élevée

Document A4 : Scénarios de risques et mesures à prévoir

Événement N°	Scénarios de risques	Mesures à prévoir
1	En tant qu'attaquant externe, je réalise une injection *SQL* sur le formulaire d'authentification.	- Filtrer les données saisies dans le formulaire - Modifier le code d'interrogation de la base de données : requêtes préparées
2	En tant qu'attaquant externe, j'utilise un outil d'attaque des mots de passe.	**** à compléter sur votre copie ****
3	En tant que client peu scrupuleux, j'effectue des demandes de pièces que je ne valide pas.	Bloquer les clients qui ne confirment pas les demandes de pièce dans le temps imparti.
4	En tant que client peu scrupuleux, j'effectue des demandes de pièces non compatibles avec l'aéronef concerné.	Vérifier la compatibilité d'une demande de pièce avec l'aéronef concerné.
5	En tant qu'attaquant, j'essaie de corrompre le *PGI* par injection de code dans les demandes de pièce.	**** à compléter sur votre copie ****

Documents associés au dossier B

Document B1 : Clé OTP (One Time Password)

Un mot de passe à usage unique (*OTP*, de l'anglais *one time password*) est un mot de passe qui n'est valable que pour une session ou une transaction. Ces mots de passe permettent de combler certaines lacunes associées aux traditionnels mots de passe statiques, comme la vulnérabilité aux attaques par rejeu (ou attaques par relecture).

Le principe *OTP* retenu est basé sur un algorithme qui fournit un code en fonction d'une clé secrète immuable (code *PIN* enregistré dans la base de données pour chaque client) et de l'heure courante.

L'utilisation est simple : la clé *OTP*, qui a l'aspect d'un petit boîtier avec un bouton et un afficheur, génère un code à 6 chiffres à chaque appui sur le bouton.

Ce code doit être renvoyé à l'application dans la minute qui suit.

L'application est en mesure de produire le même code afin de le comparer au code reçu par l'utilisateur grâce aux fonctionnalités offertes par la classe OTP :

- Constructeur **OTP(Client $client)**
- Méthode publique **getCode(DateTime $date) : int**
  // Cette méthode renvoie le code de validation (à comparer à celui généré par l'utilisateur avec sa clé OTP)

  Remarque : DateTime est une classe standard du langage *PHP*, dont l'instanciation fournit la date courante.

Exemple d'utilisation de la classe OTP :

```
$uneDateTime = new DateTime();
$unUtilisateur = new Utilisateur(…);
$unOtp = new OTP($unUtilisateur);
$code = $unOtp->getCode($uneDateTime);
```

Document B2 : Formulaire contenu dans la vue loginView.php

```
<form method="post" action="/login/val" enctype="multipart/form-data">
  <label for="login">Login :</label><input type="text" id="login" name="login"/>
  <label for="pass">Mot de passe :</label><input type="password" id="pass" name="pass"/>
  <input type="submit" value="Valider">
</form>
```

Document B3 : Classe contrôleur LoginCtrl

```
class LoginCtrl
{    ...
    //méthode appelée pour afficher le formulaire d'authentification
    public function afficheLoginView() {...}
    //méthode appelée pour vérifier le login d'un utilisateur
    public function verifLogin():void
    {
        // on récupère les valeurs passées en post en les filtrant
        $loginForm = filter_input(INPUT_POST, 'login', FILTER_SANITIZE STRING);
        $passForm = filter_input(INPUT_POST, 'pass', FILTER_SANITIZE STRING);
        $daoC = new ClientDao();
        $erreur = false;
        $client = $daoC->readbyLogin($loginForm);
        if($client == null){
            $erreur = true;
        }
        else {
         // le mot de passe saisi est-il égal au mot de passe haché stocké dans la BDD
            if (password_verify($passForm, $client->getPass() ) {
                new AccueilView();
            }
            else{
                $erreur = true;
            }
        }
        if($erreur){
             new LoginView("Authentification invalide");
        }
    }
}
```

Document B4 : Classe DAO ClientDao.php

```
class ClientDao extends Dao
{
    public function readByLogin($login):?Client
    {   // renvoie un Client ou null si aucun login ne correspond
        $connex = DB::getPdo();
        $req = "SELECT id, nom, login, pass, courriel, telephone, adresse,
             pays, estBloque, codePinOtp FROM client WHERE login = '" . $login . "'";
        $res = $connex->query($req);
        $enreg = $res->fetch(PDO::FETCH_OBJ);
        if ($enreg != false) {
           $unClient = new Client($enreg->id, $enreg->nom, $enreg->login, $enreg->pass,
               $enreg->courriel, $enreg->telephone, $enreg->adresse, $enreg->pays,
               $enreg->estBloque, $enreg->codePinOtp);
        } else {
            return null;
        }
        $res->closeCursor();
        return $unClient;
    }   //autres méthodes du DAO...
}
```

Remarque importante : la classe Client a la même structure que la table Client et tous les accesseurs ont été créés : getId(), getNom(), etc.

Document B5 : Exemple d'utilisation de requête préparée en PHP

```
1.  //Connexion à la base Mysql
2.  $connex = DB::getPdo();
3.  //Préparation d'une requête
4.  $sql = "SELECT idAvion FROM avion WHERE constructeur = ? AND anneeFab < ?";
5.  $prep = $connex->prepare($sql);
6.  //envoi des données et de la requête
7.  $prep->bindValue(1,'Airbus', PDO::PARAM_STR);
8.  $prep->bindValue(2, 2002, PDO::PARAM_INT);
9.  //exécution
10. $prep->execute();
11. while ($row = $prep->fetch(PDO::FETCH_OBJ)) {
12.      //Traitement des résultats
13. }
14. // on ferme le curseur des résultats
15. $prep->closeCursor();
```

Document B6 : Utilisation de journaux syslog en PHP

syslog (int $priority, string $message) : bool

La fonction *syslog* génère un message qui sera inscrit dans l'historique par le système.

Le paramètre *priority* contient une constante représentant le niveau du message. Les valeurs possibles sont (par ordre de gravité décroissant) :

Constante	Description
LOG_EMERG	système inutilisable
LOG_ALERT	une décision doit être prise immédiatement
LOG_CRIT	condition critique
LOG_ERR	condition d'erreur
LOG_WARNING	condition d'alerte
LOG_NOTICE	condition normale, mais significative
LOG_INFO	message d'information
LOG_DEBUG	message de débogage

Le paramètre *message* contient le message à envoyer.

Cette fonction retourne la valeur **true** en cas de succès ou la valeur **false** si une erreur survient.

Exemple

```
<?php
...
    syslog(LOG_ERR, "Message à journaliser");
}
closelog();
?>
```

Documents associés au dossier C

Document C1 : Extrait du schéma relationnel de la base de données PartEdge

```
TypePiece(id, libelle)
      clé primaire : id

Piece(numSerie, dateFabrication, prix, etat, siteStockage, idTypePiece, idDemande)
      clé primaire : numSerie
      clé étrangère :
            idTypePiece en référence à id de TypePiece
            idDemande en référence à id de Demande        (null autorisé)

Client(id, nom, login, pass, courriel, telephone, adresse, pays, estBloque,
codePinOtp)
      clé primaire : id

Demande(id, dateDemande, dateConfirmation, typeEchange, fraisTransport,
modeTransport, numSerieAeronef, modeleAeronef, idClient, idTypePiece, numPiece)
      clé primaire : id
      clés étrangères :
            idClient en référence à id de Client,
            idTypePiece en référence à id de TypePiece,
            numPiece en référence à numSerie de Piece     (null autorisé)
```

Document C2 : Utilisation de la clause INTERVAL et de la fonction NOW() en SQL

- La clause INTERVAL permet d'ajouter ou retirer un espace de temps à une date.

Exemple de requête :

```
SELECT dateFabrication, dateFabrication + INTERVAL 2 YEAR
FROM Piece
WHERE numSerie = 15
```

Résultat possible :

```
| dateFabrication | dateFabrication + INTERVAL 2 YEAR|
+-----------------|----------------------------------+
| 2021-12-01      | 2023-12-01                       |
+ -------------------------------------------------+
```

De la forme : **SELECT unTemps +/- INTERVAL nombre unité**

Expression	Unité
Année	YEAR
Mois	MONTH
Jour	DAY
Heure	HOUR
Minute	MINUTE

- La fonction NOW() renvoie la date et l'heure du système.

Exemples d'utilisation de la fonction NOW() :

```
SELECT id from Demande where dateDemande = NOW();
SELECT NOW();
```

Document C3 : Exemple de déclencheur (trigger)

Le déclencheur suivant permet d'empêcher la réutilisation d'un mot de passe déjà utilisé pour un client. Ce déclencheur exécute son code avant chaque exécution d'instruction UPDATE sur la table client.
Il existe une table d'historisation des données des clients client_histo, dont la structure est quasi-identique à la table Client et qui permet donc de stocker tous les mots de passe successifs utilisés pour chaque client.

```
CREATE OR REPLACE TRIGGER verif_mot_de_passe
BEFORE UPDATE ON Client
FOR EACH ROW
BEGIN
      DECLARE nb int;
      --on recherche dans la table d'historisation des clients
      SELECT COUNT(*)   INTO nb;
      FROM client_histo
      WHERE id = NEW.id
      AND pass = NEW.pass

      IF nb > 0
      --Tentative de réutilisation d'un ancien mot de passe, on l'empêche
            CALL EXIT;
      ENDIF
END
```

Document C4 : Schémas de modélisation de la base de données PartEdge

Diagramme de classes

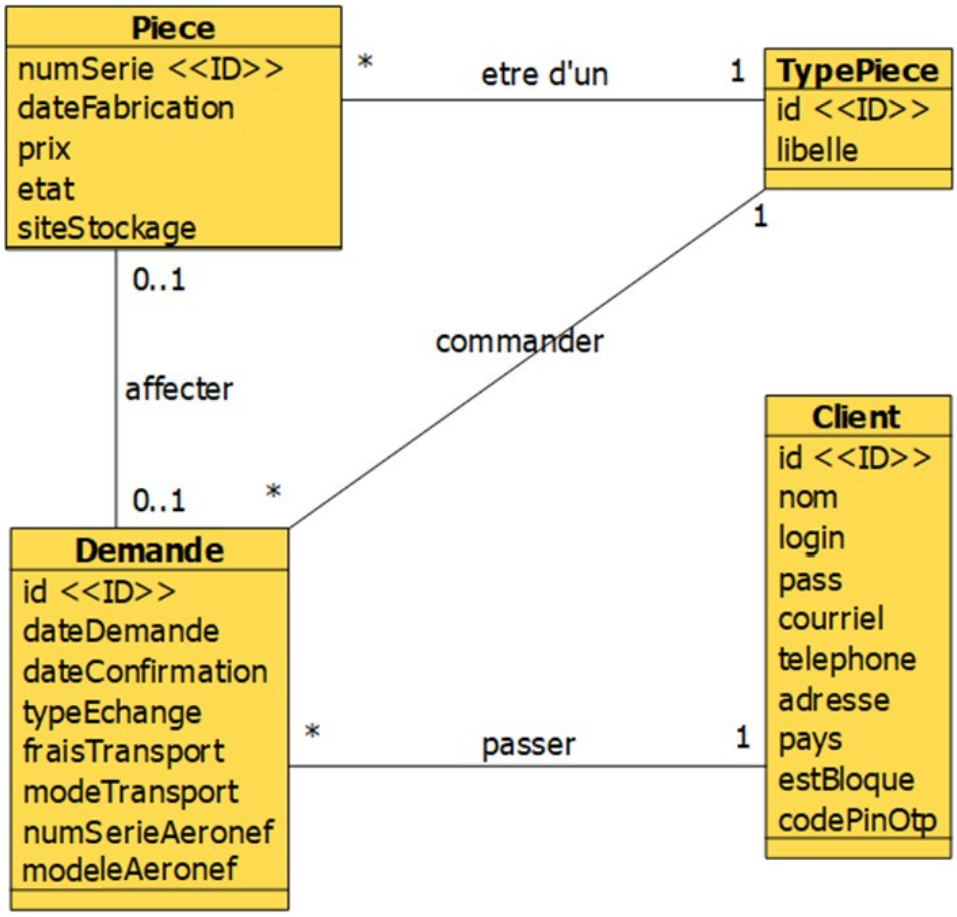
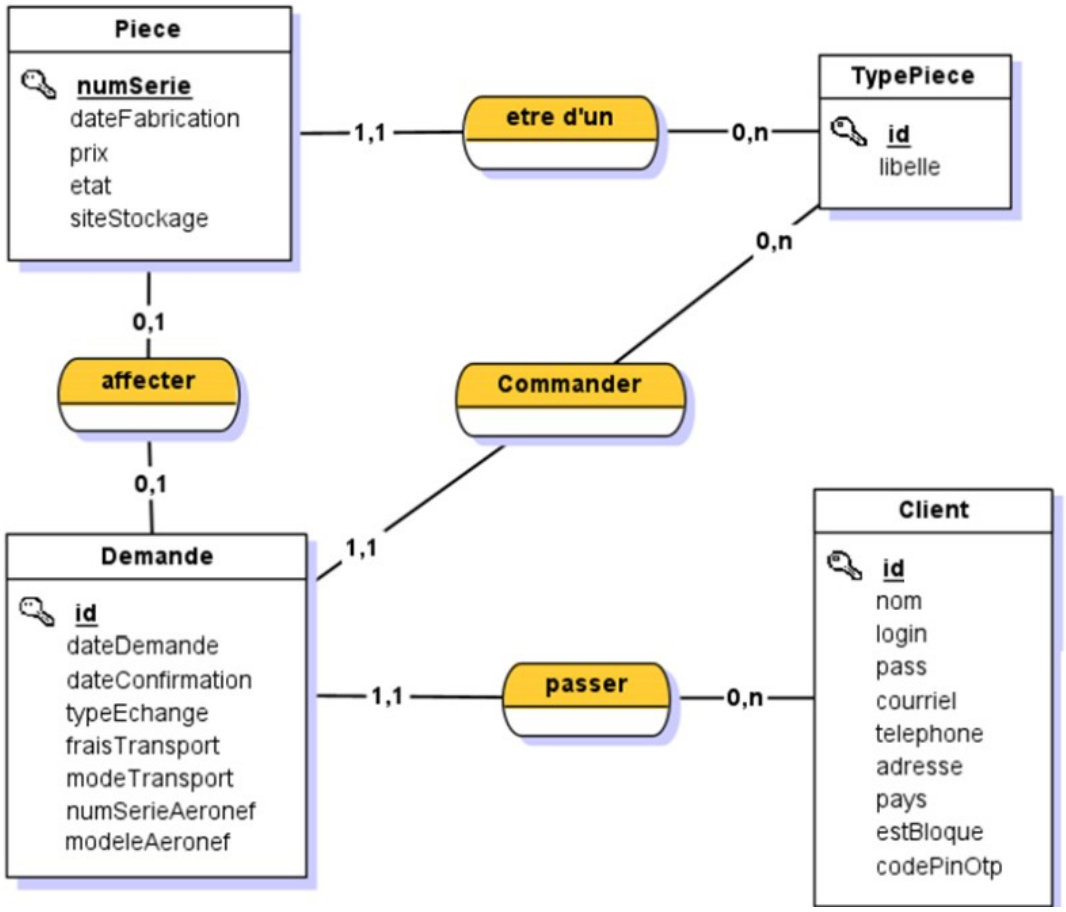


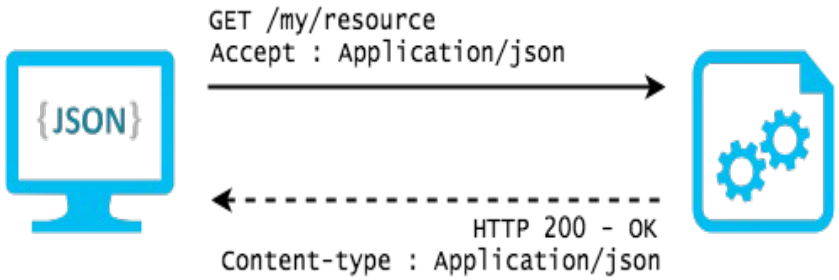
Schéma entité-association


Piece
numSerie
dateFabrication
prix
etat
siteStockage
1,1
etre d'un
0,n
TypePiece
id
libelle
0,n
0,1
affecter
Commander
0,1
Demande
id
dateDemande
dateConfirmation
typeEchange
fraisTransport
modeTransport
numSerieAeronef
modeleAeronef
1,1
1,1
passer
0,n
Client
id
nom
login
pass
courriel
telephone
adresse
pays
estBloque
codePinOtp

Documents associés au dossier D

Document D1 : Interface REST en JAVA EE

L'approche *REST* se base sur les méthodes HTTP pour déterminer l'opération à réaliser.



Les verbes HTTP correspondant aux opérations de type CRUD (Create, Read, Update, Delete) :

- Créer (Create) : @POST
- Afficher (Read) : @GET
- Mettre à jour (Update) : @PUT
- Mettre à jour partiellement (Update) : @PATCH
- Supprimer (Delete) : @DELETE

Les types de médias (*MIME types*) consommés en entrée et délivrés en sortie sont définis respectivement par les annotations @Consumes et @Produces (*JSON, XML*, ...).

La méthode *REST* se base sur les codes retour HTTP pour préciser le statut d'une ressource suite à l'exécution d'une requête.

Les codes retour HTTP les plus couramment utilisés dans les *API REST* sont fournis dans le tableau suivant. Au lieu d'utiliser le code HTTP au format numérique, il est possible d'utiliser une constante :

Code HTTP	Constante	Commentaire
200	OK	indique la réussite d'une requête
201	CREATED	indique que la requête a réussi avec la création d'une ressource (avec renvoi ou non d'une entité)
204	NO_CONTENT	indique que la requête a réussi et que le serveur ne renvoie pas de contenu (requête PUT)
400	BAD_REQUEST	indique que le serveur ne peut pas comprendre la requête en raison d'une syntaxe invalide
404	NOT_FOUND	indique qu'un serveur ne peut pas trouver la ressource demandée

Document D2 : Classe API REST DemandeRest.java

```
@Path("/demande")
public class DemandeRest {
	// méthode qui retourne la demande correspond à l'id demandé
	@GET
	@Path("{id}")
	// l'annotation @Produces indique que les caractéristiques de la demande
	// demandée sont retournées au format JSON
	@Produces(MediaType.APPLICATION_JSON)
	public Response getDemandeParId(@PathParam("id") int id) {
		// le service retourne une demande et un code HTTP 200
		return Response.ok((DemandeService.findById(id)).build();
	}
	// méthode qui met à jour une demande
	@PUT
	// l'annotation @Consumes indique que les caractéristiques de la demande
	// à créer sont fournies au format JSON
	@Consumes(MediaType.APPLICATION_JSON)
	public Response update(DemandeJson laDemandeJson) {
		// met à jour la demande et retourne un code HTTP 204 ou 400
		boolean ret = DemandeService.update(laDemandeJson);
		if (ret == true){
			return Response.status(Status.NO_CONTENT).build();
		} else {
			return Response.status(Status.BAD_REQUEST).build();
		}
	}
	// méthode qui crée une demande
	@POST
	@Consumes(MediaType.APPLICATION_JSON)
	public Response createDemande(DemandeJson laDemandeJson)
	{
		//	** À compléter sur votre copie **
	}
}
```

Document D3 : Classes DemandeJson et DemandeService

```
// classe permettant d'obtenir une demande à partir de données au format JSON
public class DemandeJson implements Serializable {
	private String dateDemande, dateConfirmation, typeEchange, modeTransport,
modeleAeronef;
	private int  id, numPiece, numSerieAeronef, fraisTransport, idClient, idTypePiece;
	public DemandeJson(int id, String dateDemande, String dateConfirmation, String
typeEchange, int fraisTransport, String modeTransport, int numSerieAeronef, String
modeleAeronef, int idClient, int idTypePiece, Integer numPiece) {
		this.id = id;
		this.dateDemande = dateDemande;
		this.dateConfirmation = dateConfirmation;
		this.typeEchange = typeEchange;
		this.fraisTransport = fraisTransport;
		this.modeTransport = modeTransport;
		this.numSerieAeronef = numSerieAeronef;
		this.modeleAeronef = modeleAeronef;
		this.idClient = idClient;
		this.idTypePiece = idTypePiece;
		this.numPiece = numPiece;
	}
// Tous les « Getter » et « Setter » ont été créés dans la classe DemandeJson
// Le paramètre Integer numPiece dans le constructeur permet d'accepter la valeur null
}
public class DemandeService {
```

```
   // méthode qui contrôle les valeurs de la demande passée et crée une demande
   public static boolean create(DemandeJson laDemande){
       boolean res = true;
       // contrôle des valeurs
       res = verifDonneesCreate(laDemande);
       if (res == true) {
              Date dateDemande;
              SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
              try {
                  // conversion de la date reçue en string au format Date
                  dateDemande = dateFormat.parse(laDemande.getdateDemande());
                  // on crée une instance de Demande
                  Demande cmd = new Demande(laDemande.getId(), dateDemande, null,
                           laDemande.getTypeEchange(),laDemande.getFraisTransport(),
                           laDemande.getModeTransport(), laDemande.getNumSerieAeronef(),
                           laDemande.getModeleAeronef(),laDemande.getIdClient(),
                           laDemande.getIdTypePiece(), laDemande.getNumPiece());

                  // on appelle la méthode create de la classe DaoDemande
                  // chargée d'ajouter laDemande
                  DaoDemande dao = new DaoDemande();
                  res = dao.create(cmd);
              } catch (ParseException e) {
                     e.printStackTrace();
                     res = false;
              }
          }
          return res;
       }
       // méthode qui retourne la demande correspondant à un id passé en paramètre
       public static DemandeJson findById(int id) {...}

       // méthode qui contrôle la demande passée et la met à jour
       public static boolean update(DemandeJson laDemande) {...}

       // méthode qui supprime la demande passée
       public static boolean delete(DemandeJson laDemande) {...}

       // méthode qui contrôle les données transmises
       public static boolean verifDonneesCreate(DemandeJson laDem)
       {
          boolean res = true;

          // La méthode statique booléenne matches() de la classe Pattern compare une
          // expression régulière à une chaîne fournie

          // l'expression ^[0-9]4-[0-9]2-[0-9]2$ correspond au format de date AAAA-MM--JJ
          if (Pattern.matches("^[0-9]4-[0-9]2-[0-9]2$", laDem.getDateDemande()) == false){
              res = false;
          }

           // l'expression INTERNE|EXTERNE signifie INTERNE ou EXTERNE
          if (Pattern.matches("INTERNE|EXTERNE", laDem.getModeTransport()) == false) {
              res = false;
          }
              //     ** À compléter sur votre copie **

          return res;
       }
}
```

Document D4 : Tests unitaires Junit et classe DemandeServiceTest

L'outil *JUnit* utilise les annotations pour définir les méthodes de test et les configurer. Les annotations suivantes sont utilisées dans le projet :

@Test : marque une méthode comme étant une méthode de test.
@Before : exécutée avant chaque test, elle est utilisée pour préparer l'environnement de test.

Assertions

L'outil *JUnit* fournit des méthodes statiques pour tester certaines conditions via la classe Assert. Elles permettent de spécifier le message d'erreur, le résultat attendu et le résultat effectif.

```
1.  public class DemandeServiceTest {
2.      private DemandeJson dem1;
3.      private DemandeJson dem2;
4.      private DemandeJson dem3;
5.
6.      // ** À compléter sur votre copie **
7.
8.      @Before
9.      public void setUp(){
10.         this.dem1 = new DemandeJson(1, "2022-03-20", " ",
11.                   "STANDARD", 0, "PERSO", 22, "B747", 2, 3, null);
12.         this.dem2 = new DemandeJson(2, "2022-03-21", " ",
13.                   "AUTRE", 0, "INTERNE", 11, "A320",2, 3, null);
14.         this.dem3 = new DemandeJson(3, "2022-03-33", " ",
15.                   "STANDARD", 0, "INTERNE", 33, "A319", 2, 3, null);
16.         //  ** À compléter sur votre copie **
17.     }
18.      // test d'une demande avec un type de transport incorrect
19.      @Test
20.      public void testCreateTypeTransportInvalide(){
21.           // exécution méthode
22.           boolean result = DemandeService.create(dem1);
23.           // test du retour de la méthode
24.           assertEquals("Create avec un type transport invalide", false, result);
25.      }
26.
27.      // test d'une demande avec un type de d'échange de pièce incorrect
28.      @Test
29.      public void testCreateTypeEchangeInvalide(){
30.           // exécution méthode
31.           boolean result = DemandeService.create(dem2);
32.           // test du retour de la méthode
33.           assertEquals("Create avec un type échange invalide", false, result);
34.      }
35.
36.      // test d'une demande avec une date de demande invalide
37.      @Test
38.      public void testCreateDateDemandeInvalide(){
39.           // exécution méthode
40.           boolean result = DemandeService.create(dem3);
41.           // test du retour de la méthode
42.           assertEquals("Create avec une date de demande invalide", false, result);
43.      }
44.      //      ** À compléter sur votre copie **
45. }
```