Sécurité des applications backend

Création d'une Application de vente sécurisée

La société Aristote est une société de vente de livres. Elle possède déjà un site web e-commerce créé sur mesure en 2012 avec la technologie PHP5. Le code a été maintenu du mieux possible par les développeurs, mais les ressources en PHP se ratissent. Tout le monde a démissionné, et la technologie ne garantit pas tous les principes nécessaires au respect de certains fondamentaux de sécurité web, il a été conseillé par un expert indépendant qu'il reviendrait moins cher de recréer une nouvelle application.

Dès lors, deux ans plus tôt, une application frontend avait été redéveloppée par une société tierce afin de remettre le design plus à jour, avec des technologies JavaScript récentes.

Mais la direction d'Aristote aimerait recréer une application backend avec des technologies plus à jour afin de simplifier le code devenu inmaintenable par la société qui avait développé l'ancienne application. Cela lui permettrait d'ajouter des fonctionnalités plus facilement.
De plus, l'application en production n'étant plus maintenue par manque de ressources humaines, le respect des principes de sécurité OWASP vus au cours ne sont plus garantis.
Le directeur général d'Aristote ayant récemment participé à un séminaire sur la cybersécurité sait qu'il prend de grands risques en gardant sa partie backend de plus en plus sensible aux attaques de nouvelles génération.

Etant donné que vous êtes en dernière année de formation, vous avez été contacté pour un appel à projet par le directeur d'Aristote. Il avait entendu parler des technologies web Java, connues pour leurs performances, ainsi que leurs facilité à créer des applications web robustes, qualitatives et de grande envergure pour de longues années de maintenance.

Il vous demande donc de répondre à son appel à projet, en lui proposant une petite application backend sécurisée, à l'aide de la technologie Spring Boot, qui lui permettra de voir la mise en œuvre de la sécurité appliquée à son domaine. La partie visuelle ne doit pas être développée. L'API sera accessible via l'utilisation de l'outil Postman.

Ses demandes sont les suivantes :

- L'application sera créée avec une version de Spring Boot récente.
- L'outil de migration de base de données *Liquibase* adapté à la création de votre schéma de base de données relationnelle.
- L'intégration d'un système de gestion de base de données relationnelle de type PostgreSQL, ou MySQL (au choix).
- Un utilisateur doit pouvoir se connecter à l'application.
  Vous intégrerez un système d'inscription et de connexion avec *OAuth2.0*, ou *OpenID Connect*.
- Un utilisateur peut consulter ses données personnelles (nom, prénom, adresse de livraison, adresse de facturation, N° de TVA, N° de carte bancaire, etc.), les modifier, les supprimer.
  Ces informations sont cependant obligatoires pour valider une commande.
- Un article est constitué d'un identifiant unique inaccessible pour l'utilisateur, d'un titre, d'un code ISBN unique, d'une description, d'une photo, d'un prix, d'une quantité, et de toute autre information pertinente. Un article est lié à une ou plusieurs catégories.

- Un utilisateur doit pouvoir créer, mettre à jour et supprimer des données, selon ses autorisations et ses accès (rôles) dédiés.
  Trois rôles sont alors disponibles :
    - Visiteur → lecture seule des articles, mais ne peut pas commander
    - Client → lecture seule des articles, peut en ajouter/supprimer à un panier
    - Administrateur → lecture, création, modification, suppression d'articles via des APIs dédiées
- Permettre d'accéder aux URLs (routes) développées, via l'outil Postman, tout en s'authentifiant à l'application backend lorsque cela est requis.
- L'application doit comprendre un système de transactions, afin que les utilisateurs ne puissent pas commander plusieurs articles lorsqu'il n'en reste pas suffisamment, ni annuler un paiement lorsqu'il est en cours.
  De la même manière, en cas de modification d'un article, il est nécessaire que les modification appliquées par l'administrateur ne puissent pas interagir avec le paiement en cours par un client. De cette manière, si une transaction d'un client est en cours au même moment qu'une modification par un administrateur, il sera nécessaire que la société assume la commande passée avant la modification de la transaction.
- Une API pour chaque requête susceptible d'être utile (et uniquement). Entre autres, l'application doit comprendre les éléments suivants :
    - L'accès à une liste d'Articles
        - service GET /article/all : renvoie une liste d'articles, paginée. Tous les types d'utilisateurs auront accès.
        - service GET /article : renvoie la fiche d'un article donné selon un ISBN. Tous les types d'utilisateurs auront accès. Utilisez les *query parameters*.
        - L'utilisateur doit pouvoir filtrer les articles par Catégorie
    - L'accès au panier d'un Utilisateur
        - Tout utilisateur peut accéder à son panier, ajouter, supprimer des articles,
        - Seul un utilisateur connecté peut valider sa commande avec ses informations de facturation.
        - Un administrateur peut consulter la liste des paniers non-commandés des utilisateurs.
    - L'accès à une liste des Commandes par utilisateur
        - service GET /command/all : renvoie la liste des commandes pour tous les utilisateur. Ce service ne sera accessible qu'à l'administrateur.
        - service GET /command/{id} : renvoie une commande pour un utilisateur donné, s'il a l'autorisation d'y accéder. Seul l'utilisateur connecté dédié a accès à ses commandes.
        - Une commande possède un identifiant, un code fonctionnel, un statut (NEW, CONFIRMED, TO_PAY, PAID), et de toute autre information pertinente.
    - Une liste des Utilisateurs accessible uniquement par un Administrateur

- L'application doit intégrer un certificat de sécurité à l'aide du protocole *SSL/TLS*.
- L'application doit être résiliente face aux différents types d'attaques listées par OWASP.
- Le paiement du site web est effectué à la validation du panier, à l'aide de l'API *Stripe*. En cas d succès et redirige vers une page de confirmation en cas de succès, où l'état de la commande passe de l'état « CONFIRMED » à « PAID », sinon une redirection est faite vers une page d'erreur et le statut de la commande reste à « TO_PAY ».
- Les erreurs serveur et rediriger vers des pages dédiées.

- L'application doit être fonctionnelle et la plus robuste possible. Il pourra vous être demandé de faire une démonstration de vos API's à l'aide l'outil Postman, ainsi que d'expliquer comment vous avez pensé la protection des fonctionnalités de l'application.
- Toutes les vérifications de sécurité jugées nécessaires sont à faire côté backend. Etant donné qu'une application web ne doit faire confiance à personne, vous validerez les données en entrée dans l'application.
- Vous utiliserez la bibliothèque *Spring Security* pour rendre votre application sécurisée.
- Vous inclurez un dossier *ressources* contenant les fichiers JSON permettant d'appeler vos APIs avec l'outil Postman.

Vous pouvez consulter le formateur en cas de besoin d'informations. Celui-ci vous répondra en guise d'intermédiaire de la société Aristote.

Consignes de remise du travail :

- Votre travail sera déposé sur un dépôt *Github* personnel, avec un fichier *README.md* à sa racine, contenant les informations nécessaires pour être capable d'installer et de démarrer l'application.
- Vous référencerez le dépôt dans le devoir prévu à cet effet sur la plateforme *ItsLearning*, et vous vous assurerez de fournir un accès du code produit au formateur (pierre-yves.crutzen@formateur.ifapme.be).
- Le code de l'application sera également remis dans le même devoir dans un fichier **.zip** .
- Vous rendrez un rapport expliquant les aspects techniques de votre application pour gestion de la sécurité (connexion, des autorisations et rôles, sécurisation des API's, configurations de sécurité, certificats, etc.), ainsi que la manière dont vous respectez les principes de sécurité demandés. Vous y expliquerez comment il est possible de résister au différents types d'attaques OWASP Top 10 selon le contexte d'un environnement client.