Okan Emre KARADAG
Dorian BUYAT
Axel HASER AGHAJARI
Nathan BOUQET

Document de Sécurisation de l’application

Le périmètre de cette analyse se concentre sur la couche applicative de l’application (code, services, endpoints REST, sécurité des données, configuration Spring Security). L’analyse n’intègre pas, dans ce document, la sécurité réseau, la configuration des serveurs systèmes, ni la sécurité front-end côté navigateur. Ces éléments pourraient faire l’objet d’un audit complémentaire.

Nous constatons de multiples expositions des API REST via des points d'entrée tels que UtilisateurRestController, AuthRestController, CarteRestController et MarketController, qui assurent l'authentification et la gestion des transactions sans appliquer de contrôles d'accès différenciés ni de segmentation des privilèges.

Le système actuel d'authentification centralisée, basé sur une API dédiée, ne met pas en œuvre des méthodes modernes comme la génération et la gestion de tokens ou le renouvellement sécurisé des sessions. De plus, l'accès aux données sensibles n'est pas suffisamment restreint, ce qui favorise d'éventuelles élévations de privilèges en cas de faille. Par ailleurs, la gestion des mots de passe (restés en clair ou de manière insuffisamment sécurisée) représente un risque important.

L'application est aussi exposée à des vulnérabilités comme le Cross-Site Scripting (XSS) et Cross-Site Request Forgery (CSRF), en raison d'une validation insuffisante et d'une absence de mise en œuvre de tokens CSRF. L'absence de journalisation centralisée et de système de surveillance complique la détection des intrusions ainsi que les analyses post-incidents.

Pour remédier à ces failles, l’intégration de Spring Security est une solution incontournable pour sécuriser les processus d’authentification et d’autorisation. En intégrant notamment le mécanisme JSON Web Token (JWT), l’application bénéficiera d’une authentification sans état, évitant ainsi le stockage de session côté serveur et permettant une gestion plus fine des accès à chaque endpoint. Le SecurityContext de Spring Security sera utilisé pour définir et vérifier les rôles de manière détaillée. Il sera aussi essentiel de chiffrer les mots de passe avec des solutions robustes comme Bcrypt, afin de garantir la sécurité des données sensibles.

Pour sécuriser les communications, toutes les interactions client-serveur devront se faire via le protocole **HTTPS**, assurant un chiffrement de bout en bout des données. Il faudra également mettre en place des mesures de prévention contre les attaques **CSRF** et **XSS**, notamment par l'utilisation de tokens CSRF et la validation des entrées utilisateurs.

Okan Emre KARADAG
Dorian BUYAT
Axel HASER AGHAJARI
Nathan BOUQET

Pour garantir l'intégrité des changements, une stratégie de test renforcée sera mise en place. L'utilisation d'outils comme JUnit et Mockito permettra de réaliser des tests unitaires et fonctionnels. Sonar sera utilisé en complément pour surveiller la couverture de tests et détecter les anomalies dans le code.

Le passage vers une **architecture MicroServices** représente une autre opportunité d'améliorer la sécurité globale de l'application. En adoptant cette approche, l'application sera décomposée en services isolés et autonomes, chacun gérant une fonction spécifique (authentification, utilisateurs, cartes, market, etc.). Cela réduit la surface d'attaque et facilite la mise en œuvre de contrôles d'accès spécifiques à chaque composant. La mise en place d'une **API Gateway** jouera un rôle central, agissant comme point d'entrée unique pour les requêtes, gérant l'authentification, appliquant des limites de débit et centralisant la journalisation des accès. Chaque microservice disposera de sa propre base de données, réduisant les risques de propagation en cas de faille.

Le plan de mise en œuvre sera divisé en plusieurs phases. Dans un premier temps, Spring Security sera déployé pour sécuriser les endpoints critiques, chiffrer les communications via HTTPS et utiliser Bcrypt pour les mots de passe. Ensuite, des tests unitaires et fonctionnels valideront chaque changement et garantiront que l'intégration des nouvelles mesures ne perturbe pas le fonctionnement existant de l'application. En deuxième phase, l'architecture sera migrée progressivement vers une structure MicroServices. Une analyse détaillée permettra de redéfinir le découpage fonctionnel de l'application, et l'API Gateway sera déployée pour centraliser la gestion des requêtes et appliquer les mécanismes de sécurité.

En résumé, le renforcement de la sécurité applicative repose sur un ensemble de mesures techniques et organisationnelles : intégration de Spring Security avec JWT et Bcrypt, sécurisation des communications via HTTPS, prévention des attaques XSS et CSRF. La transition vers une architecture MicroServices permettra une isolation des services, une gestion centralisée des accès via une API Gateway et une surveillance renforcée par l'analyse continue des logs. Ces évolutions, appuyées par des tests unitaires et des audits réguliers, constituent une solution complète et robuste pour faire face aux enjeux de sécurité actuels et futurs, tout en assurant la continuité et la qualité du service pour les utilisateurs.

Okan Emre KARADAG
Dorian BUYAT
Axel HASER AGHAJARI
Nathan BOUQET

Catégorie	Vulnérabilité	Risques	Mesures de remédiation
Contrôle d'accès	Absence de contrôle d'accès différencié sur les contrôleurs REST	Accès non autorisé à certaines fonctionnalités ou données	Intégration de Spring Security avec contrôle détaillé via `SecurityContext`, gestion des rôles et autorisations.
Authentification	Authentification centralisée obsolète, pas de gestion de tokens ni de renouvellement sécurisé	Sessions vulnérables au vol d'identifiants, réutilisation de sessions	Implémentation de JWT.
Stockage des mots de passe	Mots de passe stockés en clair ou de manière peu robuste	Fuite de données sensibles en cas de compromission de la base de données	Utilisation de Bcrypt pour le hachage sécurisé des mots de passe.
Forces brutes	Mécanisme de protection contre les attaques par force brute absent	Possibilité de compromission de comptes par attaque dictionnaire	Mise en place de limites de tentatives, captchas, temporisation ou blocage temporaire.
XSS (Cross-Site Scripting)	Aucune validation/échappement des entrées utilisateurs	Injection de scripts malveillants, vol de session ou redirections frauduleuses	Validation côté serveur et client, échappement des caractères spéciaux, Content Security Policy (CSP).

Okan Emre KARADAG
Dorian BUYAT
Axel HASER AGHAJARI
Nathan BOUQET

CSRF (Cross-Site Request Forgery)	Absence de token CSRF, mauvaise implémentation	Exécution non autorisée d'actions par un utilisateur authentifié	Génération et vérification de tokens CSRF sur les requêtes sensibles.
Journalisation & surveillance	Absence de système centralisé de journalisation et de détection des intrusions	Détection tardive des attaques, analyses post-incident impossibles	Mise en place de logs centralisés, détection d'anomalies, filtrage par ip et pages visitées.
Communication	Utilisation potentielle de Basic Auth sans canal sécurisé	Interception des identifiants via attaques type Man-In-The-Middle	Obligation de passer par HTTPS, abandon de Basic Auth au profit de tokens (ex : JWT).
Architecture monolithique	Surface d'attaque large, absence d'isolation entre services	Compromission d'un composant met en danger l'ensemble de l'application	Migration vers une architecture MicroServices avec API Gateway et segmentation des services et des bases de données.
Tests & audit	Absence de tests de sécurité automatisés et d'audits réguliers	Déploiement de code vulnérable, faible résilience aux attaques nouvelles	Tests automatisés (JUnit, Mockito), couverture de code (Sonar), audits de sécurité.
Failles Spring Security	Mauvaise config. de rôles, CSRF/CORS, mots de passe en dur	Accès non autorisé, attaque CSRF, fuite d'identifiants	Bonnes pratiques Spring Sec., Bcrypt, MAJ régulières, audit config

Okan Emre KARADAG
Dorian BUYAT
Axel HASER AGHAJARI
Nathan BOUQET

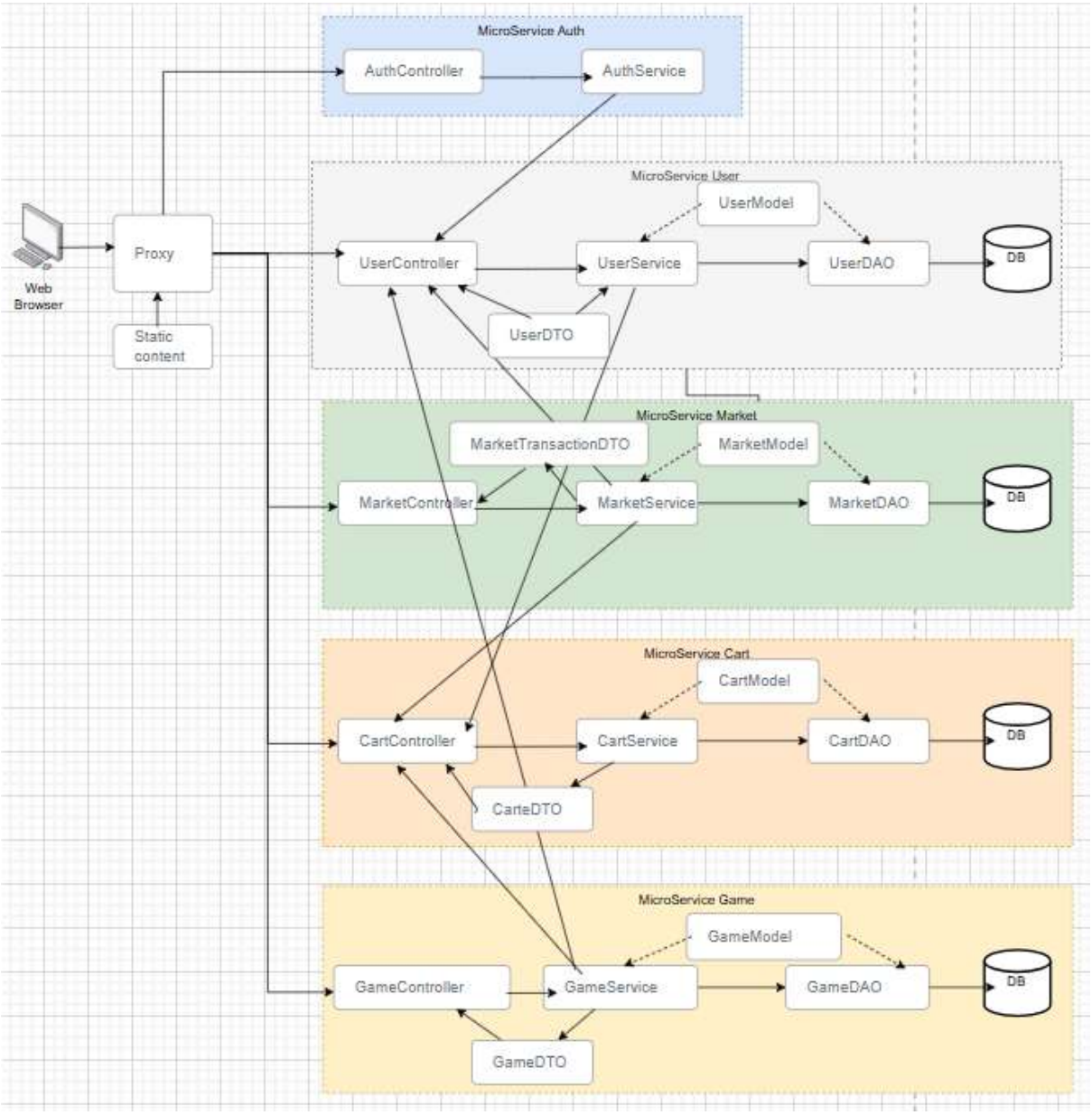

SCHÉMA D'ARCHITECTURE MIS À JOUR