Le Projet CodeAlpha

Contexte

L'entreprise **CodeAlpha** s'est spécialisée dans la production de produits à destination des centrales nucléaires, des laboratoires de physique, des universités, etc...

Elle doit gérer les présences de son personnel, techniciens, ingénieurs, responsables, etc. dans des salles sous contraintes liées à la radioactivité.

Actuellement, elle utilise des lecteurs de cartes magnétiques à l'entrée de ses salles, permettant une remontée des informations dans une application tiers avec enregistrement dans un fichier de type tableur.

Elle désire remplacer, pour une meilleure gestion des accès, les lecteurs de cartes magnétiques par des lecteurs de QR Code à l'entrée des salles avec un traitement par une application NodeJS interne.

*Vous êtes chargé, en tant que tout nouveau salarié - et en fonction de vos compétences dans le développement **Back** - de participer à la création de ce projet.*

Cahier des charges du projet

Un précédent travail (partiel) a déjà été effectué par un développeur sur l'application NodeJS, à savoir une application nommée **QRIdent**.

Vous disposez des sources de cette application qu'il va falloir transformer en **CodeAlpha** avec des améliorations à effectuer.

Principe de fonctionnement de l'application

→ Le personnel habilité est enregistré dans l'application à l'aide du SGBD MongoDB ; l'enregistrement comprend son identité (nom et prénom), un code, une adresse email et son poste (technicien, ingénieur, responsable).

→ Un opérateur, en fonction des demandes d'interventions validées (par écrit), génère un QR Code qui est automatiquement envoyé sur la boîte mail sécurisée de la personne concernée.

→ La génération du QRCode implique aussi l'inscription d'une ligne dans une collection pour les interventions ; l'enregistrement comprend le code de l'intervenant, la salle, la date et heure prévisionnelles de début, celles de l'entrée réelle et celles de fin.

→ L'intervenant scanne son QR Code sur le lecteur en début et en fin d'intervention, ce qui a pour effet d'envoyer, outre son code personnel, les heures à destination d'une API gérée par CodeAlpha.

→ L'API modifie dans la collection pour les interventions la ligne existante en rapport pour y inscrire la date et heure de l'entrée réelle et celles de fin.

Lexique :

Code : code unique du personnel, format laissé à votre appréciation.
Salle : représentée par une identification, format laissé à votre appréciation.
Heure : représentation sous forme de chaîne de caractères de l'objet Date (en JavaScript).

Périmètre technique de l'application

→ L'application dispose d'une page permettant l'affichage des intervenants.

Note : *Vous devez mettre en place un action de création automatique d'un jeu d'essai d'intervenants (collection* `intervenants`*). Vous avez le choix de la méthode (données reçues via HTTP, création automatique dans un contrôleur, fichier JSon, API de type create...).*

→ L'application dispose d'une page permettant à l'opérateur de générer un passe d'accès à une salle (QR Code) en fonction des informations suivantes : code de l'intervenant, code de la salle, jour et heure de début (prévisionnelle), de début et de fin (vides au départ) de l'intervention.

→ L'application dispose d'une page permettant l'affichage des interventions, en cours ou terminées, pour l'ensemble des interventions ou pour un intervenant en particulier.

→ L'application dispose d'une partie API pour la mise à jour de la donnée temporelle réelle de début et de fin d'intervention.

Note : *l'envoi des informations par le lecteur QR Code de la salle à l'API sera simulé par une requête POST d'un client HTTP comme* ***Insomnia*** *par exemple.*

Note générale sur la sécurité : *il n'y a pas d'identification ni d'authentification pour l'opérateur sur l'application CodeAlpha ; on suppose le poste client et le serveur (le serveur contient l'application NodeJS et le SGBD MongoDB) dans une salle à accès restreint et sécurisé.* ***Cependant, le responsable de la sécurité du site souhaite qu'un filtrage des adresses IP soit mis en place dans l'application. Seuls les opérateurs disposant d'une adresse IP acceptées (ou les lecteurs des salles) pourront accéder aux fonctionnalités de l'application.***

Travail à faire

Phase 1 : **Analyse**

Vous devez réaliser pour cette partie :

- Un diagramme des cas d'utilisation
- Un tableau des actions malveillantes redoutées
- Un diagramme UML des entités
- Un tableau présentant les différentes URI supportées par votre API
- Le code NodeJS du modèle pour la collection ***intervenants***
- Le code NodeJS du modèle pour la collection ***interventions***

On vous demande de présenter cette analyse sous la forme d'un `README.adoc` (ou `.md`) situé dans un projet sur `github.com` (donc consultable en ligne). Elle contiendra bien sûr la date, le nom de l'équipe et les participants au projet (si pas individuel).

Phase 2 : **Réalisation**

Vous devez réaliser l'application en lien avec un dépôt distant (Github).

Outre les éléments d'analyse, votre rapport présentera le travail réalisé en lien avec les cas d'utilisation (ou *user storie*s) et les cas de défense (***evil user stories)*** : Pour chaque scénario, présentation du scénario lié à un cas d'utilisation (user story ou evil user story), copie(s) d'écran de l'UI, extrait(s) de code associé…

Une conclusion.

ANNEXES

Pour l'envoi du QR Code après sa génération, vous devrez utiliser le module NPM NodeMailer :
https://nodemailer.com/about/

Un exemple NodeJS assez complet avec l'utilisation de MailTrap pour les tests :
https://mailtrap.io/blog/sending-emails-with-nodemailer/

Un exemple plus précis sur l'envoi d'une image :
https://community.nodemailer.com/using-embedded-images/

Un rappel pour la construction d'une API avec NodeJS :
https://slam-vinci-melun.github.io/sio22/phase1/Construire_API_NodeJS.pdf
https://slam-vinci-melun.github.io/sio22/phase1/Construire_API_NodeJS_v2.pdf