KATA RUDY

Objectif de l'exercice : Créer un endpoint d'API qui permet de télécharger des fichiers média (images) depuis un client.

Description de l'exercice :

Imagine que tu développes une API de média pour une application de galerie d'images. Tu dois créer un endpoint qui permet à un utilisateur de télécharger une image et de la stocker sur le serveur. Voici les étapes à suivre :

- Configure un projet Node.js avec TypeScript et Express.js si ce n'est pas déjà fait.
- Créer un endpoint POST /api/media/upload qui accepte les fichiers envoyés par les clients. Tu peux utiliser des bibliothèques telles que multer pour gérer le téléchargement de fichiers.
- Valide le type de fichier pour t'assurer qu'il s'agit bien d'une image. Tu peux utiliser une bibliothèque comme file-type pour cela.
- Génère un nom de fichier unique pour le média téléchargé et stocke-le sur le serveur. Tu peux utiliser le module uuid pour générer des noms de fichiers uniques.
- Enregistre le chemin du fichier téléchargé dans une base de données (MongoDB, par exemple) avec des métadonnées telles que le nom du fichier, la date de téléchargement, etc.
- Réponds au client avec un message de réussite et l'URL du média téléchargé.
- Assure-toi de gérer les erreurs de manière appropriée, notamment en cas de téléchargement de fichiers non autorisés ou de problèmes lors de l'enregistrement du média.

Conseils :

- Tu peux utiliser express-validator pour valider les types de fichiers.

- Utilise la bibliothèque multer pour gérer les fichiers envoyés par les clients.
- N'oublie pas de configurer la gestion des fichiers statiques dans Express.js pour permettre aux clients de télécharger les médias ultérieurement.
- Teste l'API en utilisant un outil tel que Postman ou des scripts de

test automatisés.

Cet exercice te permettra de mettre en pratique tes compétences en Node.js, TypeScript et Express.js, tout en te familiarisant avec le téléchargement et la gestion de fichiers média, ce qui est essentiel pour le développement d'une API de média.

Semaine 1 - Maîtrise de Node.js et TypeScript :

Jour 1 : Installation et Configuration Avancée

- Installation avancée de Node.js avec NVM (Node Version Manager) pour gérer différentes versions de Node.
- Configuration de l'éditeur de texte ou de l'IDE avec des plugins TypeScript pour une expérience de développement optimale.

Jour 2 : Concepts Avancés de Node.js

- Asynchronicité en Node.js : Utilisation de Promises, async/await pour gérer les opérations asynchrones de manière efficace.
- Exploration avancée du système de fichiers, des modules de base, et des événements Node.js.

Jour 3 : JavaScript Avancé pour TypeScript

- Fonctions fléchées avancées et gestion des contextes.
- Utilisation de TypeScript pour typer les fonctions et les objets.

Jour 4 : TypeScript Avancé

- Interfaces et types avancés en TypeScript.
- Généricité en TypeScript pour créer des structures de données flexibles.

Jour 5 : Gestion des Dépendances

- Utilisation d'outils avancés de gestion de dépendances tels que Yarn pour gérer les packages Node.js.
- Création de scripts personnalisés pour des tâches courantes, comme la construction ou les tests.

Jour 6 : Tests Unitaires et d'Intégration

- Introduction aux tests unitaires avec Jest.
- Mise en place de tests unitaires pour les fonctions Node.js.

Jour 7 : Pratique Intensive

- Exercices pratiques et création du projet-ci dessus en Node.js/TypeScript pour consolider les compétences acquises au cours de la semaine.

Semaine 2 - Développement Avancé avec Node.js et TypeScript :

Jour 1 : Mise en Place d'Express.js

- Configuration avancée d'Express.js pour créer une base solide pour l'API.
- Création d'un routeur principal pour gérer les routes de l'API.

Jour 2 : Middleware et Validation des Requêtes

- Utilisation de middleware pour la gestion des requêtes, par exemple, pour l'authentification et l'autorisation.
- Validation des données d'entrée à l'aide de bibliothèques comme express-validator.

Jour 3 : Gestion des Erreurs

- Création de gestionnaires d'erreurs avancés pour gérer les erreurs de manière propre et informative.
- Utilisation de la journalisation avancée pour le débogage.

Jour 4 : Sécurité Avancée des API

- Utilisation de Passport.js pour l'authentification avancée (stratégies multiples, OAuth, etc.).
- Protection contre les failles de sécurité courantes (injection SQL, XSS, CSRF).

Jour 5 : Documentation de l'API

- Utilisation de Swagger/OpenAPI pour documenter l'API de manière automatique.
- Création de spécifications OpenAPI pour décrire les endpoints et les schémas de données.

Jour 6 : Pratique Intensive avec des Cas Réels

- Création d'endpoints de l'API avec Express.js, en utilisant des validations, de la sécurité, et de la gestion des erreurs.
- Création de tests d'intégration pour valider les fonctionnalités.

Jour 7 : Optimisation des Performances

- Examen des meilleures pratiques pour l'optimisation des performances en Node.js.
- Mise en cache des données fréquemment utilisées, gestion des requêtes asynchrones, et équilibrage de charge.

Semaine 3 - Base de Données, Sécurité et Performances :

Jour 1 : Approfondissement de MongoDB

- Exploration des fonctionnalités avancées de MongoDB, notamment les

index, les agrégations, et la géo-localisation.
- Mise en place de la réplication pour la haute disponibilité.

Jour 2 : Utilisation Avancée de Mongoose
- Création de schémas de données avancés avec Mongoose, y compris les références, les sous-documents, et les indexes personnalisés.
- Gestion des transactions avec Mongoose.

Jour 3 : Sécurité Avancée des API (suite)
- Mise en œuvre de la sécurité au niveau de la base de données, notamment la gestion fine des privilèges et des rôles dans MongoDB.
- Intégration de stratégies de sécurité avancées, telles que l'authentification à deux facteurs.

Jour 4 : Optimisation des Performances (suite)
- Mise en place de la mise en cache avancée, y compris la gestion du cache Redis pour améliorer les performances.
- Configuration de la journalisation des performances pour identifier les goulots d'étranglement.

Jour 5 : Sécurité contre les Attaques Web
- Protection contre les attaques courantes, telles que l'injection SQL, les attaques par déni de service, et les attaques par script intersite (XSS). ● Mise en place de politiques de sécurité HTTP avancées, y compris CSP (Content Security Policy).

Jour 6 : Pratique Intensive avec des Cas Réels
- Application des connaissances acquises en sécurisant et en optimisant l'API de média développée précédemment.
- Test de la sécurité et des performances de l'API.

Jour 7 : Planification de la Scalabilité
- Évaluation des besoins de scalabilité pour l'API de média.
- Mise en place de stratégies de scalabilité automatique basées sur la charge.

Semaine 4 - AWS et Déploiement Avancé avec Intégration de Tests :

Jour 1 : Introduction à AWS
- Exploration des services AWS pertinents pour le déploiement d'applications, tels qu'EC2, RDS, S3, Lambda, API Gateway, et Elastic Beanstalk.
- Configuration du compte AWS et gestion des clés d'accès.

Jour 2 : Création d'une Infrastructure AWS Avancée

- Mise en place d'une architecture hautement disponible en utilisant des services tels que Amazon EC2 Auto Scaling et Amazon RDS Multi-AZ. ● Configuration d'une instance EC2 pour héberger l'API.

Jour 3 : Déploiement Avancé de l'API sur AWS

- Utilisation d'Amazon Elastic Beanstalk pour déployer l'API Node.js/TypeScript.
- Configuration des environnements de développement et de production.

Jour 4 : Gestion de la Scalabilité et de la Sécurité sur AWS

- Mise en place d'un équilibrage de charge avec Amazon Elastic Load Balancer (ELB).
- Configuration de la sécurité AWS, y compris les groupes de sécurité, les stratégies IAM et la gestion des clés KMS.

Jour 5 : Intégration de Tests Continus (CI/CD)

- Configuration de pipelines de déploiement continu (CI/CD) avec des outils tels que Jenkins, Travis CI, ou AWS CodePipeline.
- Automatisation des tests unitaires, d'intégration et de bout en bout dans le pipeline CI/CD.

Jour 6 : Déploiement en Production

- Planification et exécution du premier déploiement en production sur AWS.
- Surveillance des performances, de la disponibilité et de la sécurité de l'API.

Jour 7 : Gestion de Base de Données sur AWS

- Configuration d'une base de données Amazon RDS pour l'API, avec des sauvegardes automatiques et des mises à jour.
- Utilisation d'Amazon CloudWatch pour la surveillance des performances de la base de données.