Cahier de Recette - Projet NestJS

Ce cahier de recette décrit les tests fonctionnels mis en place dans un projet NestJS back Auth, en se concentrant particulièrement sur les tests **end-to-end (e2e)**. Ce type de tests vérifie que le système fonctionne correctement dans son ensemble, en simulant des interactions avec l'API de manière proche des situations réelles.

Recette

Contexte

Le projet consiste en une API NestJS qui gère des utilisateurs et propose un système d'authentification basé sur des jetons JWT. Le but de cette recette est de s'assurer que toutes les fonctionnalités critiques, telles que l'authentification, la gestion des utilisateurs, et le traitement des erreurs, fonctionnent correctement.

Objectifs de Test

Les objectifs de cette recette sont de valider les fonctionnalités suivantes :

1. **Authentification :**
    - Authentification réussie avec des informations valides et réception d'un token JWT.
    - Échec de l'authentification en cas d'informations incorrectes ou d'utilisateur inexistant.
2. **Gestion des utilisateurs :**
    - Création d'un utilisateur client.
    - Vérification de la gestion des erreurs lors de la création d'un utilisateur avec des données incorrectes (ex. email invalide).
    - Consultation d'un utilisateur par son ID.
    - Liste de tous les utilisateurs.
    - Mise à jour des informations d'un utilisateur.
    - Suppression d'un utilisateur existant et gestion des erreurs en cas d'ID invalide.

Scénarios de Test End-to-End (e2e)

Les tests e2e permettent de vérifier l'application dans son ensemble, en simulant des requêtes HTTP sur le serveur complet. Voici un résumé des parties du code qui implémentent ces tests.

1. Authentification

Les tests dans la section **Authentication** sont clairement des tests e2e. Voici les scénarios principaux :

- **Scénario 1.1 : Authentification réussie**
    - **Action :** Une requête POST est envoyée à l'endpoint `/auth/login` avec des identifiants valides (email et mot de passe).

    - **Attendu :** Un jeton JWT valide est reçu en réponse (testé avec une expression régulière pour valider le format du token).
    - **Test e2e :** Ce test simule une authentification complète, en passant par les couches d'authentification, de validation et de génération du token JWT dans le backend.
- **Scénario 1.2 : Authentification échouée (mot de passe incorrect)**
    - **Action :** Une requête POST est envoyée avec un email correct mais un mot de passe incorrect.
    - **Attendu :** Le serveur doit répondre avec un code HTTP 401 (Unauthorized) et aucune information sur le token JWT.
    - **Test e2e :** Ici, le test vérifie que les mécanismes de sécurité et de gestion des erreurs fonctionnent correctement, en traitant un utilisateur avec des informations incorrectes.
- **Scénario 1.3 : Authentification échouée (utilisateur inexistant)**
    - **Action :** Une requête POST est envoyée avec des informations pour un utilisateur inexistant.
    - **Attendu :** Le serveur renvoie un code HTTP 401.
    - **Test e2e :** Ce test s'assure que le système ne permet pas d'authentifier des utilisateurs inexistants.

2. Gestion des utilisateurs

La section **Users** du test est également un ensemble de tests e2e. Voici les scénarios clés :

- **Scénario 2.1 : Création d'un utilisateur**
    - **Action :** Une requête POST est envoyée à l'endpoint `/users` avec les informations d'un nouvel utilisateur (email, mot de passe, prénom, nom).
    - **Attendu :** Un nouvel utilisateur est créé, et le serveur retourne un statut 201 avec les détails de l'utilisateur créé.
    - **Test e2e :** Ce test vérifie la création complète d'un utilisateur via toutes les couches de l'application, y compris la validation des données, l'enregistrement en base de données, et la réponse correcte de l'API.
- **Scénario 2.2 : Création avec un email invalide**
    - **Action :** Une requête POST est envoyée avec un email au format incorrect.
    - **Attendu :** Le serveur renvoie une erreur 400, indiquant que les données sont invalides.
    - **Test e2e :** Ce test simule l'injection de données invalides et vérifie que le système applique correctement les règles de validation définies dans l'application (via les pipes de validation NestJS).
- **Scénario 2.3 : Consultation d'un utilisateur par ID**
    - **Action :** Une requête GET est envoyée à l'endpoint `/users/{id}` avec l'ID de l'utilisateur.
    - **Attendu :** Le serveur retourne les informations de l'utilisateur.
    - **Test e2e :** Ce test vérifie la bonne récupération des données d'un utilisateur à travers toute la pile de l'application (contrôleur, service, base de données).
- **Scénario 2.4 : Liste des utilisateurs**
    - **Action :** Une requête GET est envoyée à l'endpoint `/users` pour lister tous les utilisateurs.
    - **Attendu :** Le serveur retourne une liste d'utilisateurs avec un statut HTTP 200.

    - **Test e2e :** Ce test évalue la capacité du système à gérer les requêtes de liste, incluant la pagination si nécessaire.
- **Scénario 2.5 : Mise à jour des informations d'un utilisateur**
    - **Action :** Une requête PATCH est envoyée à `/users/{id}` pour mettre à jour les informations de l'utilisateur.
    - **Attendu :** Les informations de l'utilisateur sont mises à jour et un statut 200 est retourné.
    - **Test e2e :** Ce test end-to-end vérifie la bonne transmission des modifications à la base de données et la validation des entrées.
- **Scénario 2.6 : Suppression d'un utilisateur**
    - **Action :** Une requête DELETE est envoyée à `/users/{id}` pour supprimer un utilisateur.
    - **Attendu :** L'utilisateur est supprimé avec succès et un statut 200 est retourné.
    - **Test e2e :** Ce test simule la suppression d'un utilisateur dans le système, vérifiant que toutes les étapes, depuis l'accès au serveur jusqu'à la suppression en base de données, fonctionnent correctement.

Validation et Critères d'Acceptation

- **Pour chaque scénario**, les tests doivent vérifier les réponses HTTP appropriées (200, 201, 400, 401, 404, etc.), ainsi que la structure et le contenu des réponses JSON.
- **Critères d'acceptation :**
    - Toutes les fonctionnalités décrites doivent être testées et répondre correctement aux attentes.
    - L'application doit gérer correctement les erreurs (données invalides, authentification échouée, etc.).