TEST TECHNIQUE USER ENGAGEMENT
NOTIFICATION CENTER

Analyse des problématiques techniques :

Développer une API REST impose des conventions HTTP.
Le format JSON implique un processus de sérialisation et désérialisation des datas à fournir.
L'ampleur d'un tel projet oblige un besoin de maintenabilité et une standardisation du code pour faciliter le travail d'équipe.

Outils choisis pour réaliser cet exercice :

- Symfony (version light), avec 2 bundles spécifiques pour la création d'api : JMS Serializer bundle et FOS REST bundle.
- L'ORM doctrine pour faciliter la construction de l'arbre des données à renvoyer, la mise en cache, et jouir de son approche POO pour construire des Entity claires et faciles à maintenir.
- L'extension Postman sur google chrome pour tester les appels à l'API.

Difficultés rencontrées :

La principale difficulté de l'exercice est la relation entre une notification et un éventuel contenu partagé, chaque contenu disposant de sa table spécifique en BDD.
Une première approche consiste à inclure la clé étrangère de chacune des relations possibles dans la table -notification-. Nous ne choisirons pas cette solution.
A la place, nous utiliserons une relation polymorphique, avec une stratégie de récupération : ref_table, ref_id en clés étrangères de la table -notification-. Cette approche nous permettra par la suite de rajouter n'importe quel contenu à partager, sans avoir à toucher à la table -notification-.

Le problème pourrait aussi se poser avec l'auteur de la notification, deezer user ou administrateur, si ces 2 types se trouvent dans 2 tables distinctes. N'ayant pas plus d'informations sur cette possibilité lors de cet exercice, nous partirons du principe que deezer user et administrateur sont dans la même table -user-, avec un champ -role- pour les distinguer.

Evolution à apporter au travail proposé :

- Le système choisit pour récupérer la liste des notifications n'est pas optimal :
  - 1 requète pour avoir le nombre total de notifications
  - 1 requète pour avoir le nombre de total de notifications non vu par l'utilisateur
  - 1 requète pour récupérer les notifications (à coupler avec un système de pagination)
  - Autant de requètes supplémentaires pour récupérer le contenu partagé en "lazy loading".

Pour afficher 10 notifications, nous pouvons donc avoir jusqu'à 13 requètes SQL (connu comme étant le problème N+1).
Symfony est capable de gérer le polymorphisme à travers un système de superclass et de discriminator mapping, mais je n'ai pas approfondi la question pour cet exercice.
Sinon, il faut repenser tout le système avec une approche NoSql.

- Ajout d'un composant "Pagination" pour récupérer les notifications, avec un paramètre dans l'url

lors de l'appel.

- Ajout d'un système d'authentification avec Token.

- Ajout de la suppression automatique des notifications expirées, Soit exécuté avant chaque récupération des notifications par l'utilisateur, impliquant une requête en plus, soit avec une tâche Cron sur le serveur.

- Ajouter plus de tests unitaires.

- Documenter l'API.

- Versionner l'API.