Recrutement Poisson Soluble 2025

Test Symfony Backend

Contexte

Vous devez développer une application backend qui permet d'importer des destinataires depuis un fichier CSV et d'envoyer des alertes météo par SMS à des particuliers. Pour simplifier l'envoi, l'API SMS sera simulée via un service qui se contentera de logger chaque envoi. La file d'envoi sera gérée par Symfony Messenger.

Contraintes Techniques

- **Langage & Version** : PHP 8.4 minimum
- **Framework** : Symfony 6.4
- **Backend uniquement** : Aucune interface frontend n'est nécessaire.
- **Persistance** : Aucune utilisation d'ORM. Pour la gestion des migrations SQL, vous utiliserez le bundle Doelia/sql-migrations-bundle.

Fonctionnalités à Implémenter

1. Commande d'Import CSV

- **Description** : Créer une commande console (via `bin/console`) permettant d'importer un fichier CSV.
- **CSV** : Le fichier contiendra au minimum deux colonnes : `insee` et `telephone`.
- **Traitement** :
    - Chaque ligne doit être validée (ex. : vérification du format du code INSEE et du numéro de téléphone).
    - Les enregistrements valides doivent être sauvegardés en base de données (par exemple, dans une table dédiée aux destinataires).
    - Les enregistrements non valides doivent être ignorés, et comptabilisés comme erreurs.
- **Rapport** : À la fin de l'import, la commande devra générer un rapport texte indiquant le nombre de succès et le nombre d'erreurs.

2. Service d'envoi de SMS : `SmsService`

- **Description** : Créer un service `SmsService` dont la responsabilité sera de simuler l'envoi d'un SMS.
- **Comportement** :

    - Le service ne doit pas appeler une API réelle. Il se contentera de logger les informations relatives à l'envoi (numéro de téléphone, message, date, …).
    - Ce service pourra être utilisé par d'autres parties de l'application (par exemple, via le bus Messenger).

3. Endpoint `/alerter`

- **Description** : Exposer un endpoint HTTP (GET ou POST) accessible sur `/alerter` qui prend en paramètre un code INSEE.
- **Traitement** :
    - À la réception d'un appel, le système doit rechercher en base de données les destinataires dont le code INSEE correspond au paramètre fourni.
    - Pour chaque destinataire trouvé, un SMS d'alerte météo sera envoyé en utilisant le `SmsService`.
    - L'envoi des SMS doit être réalisé de manière asynchrone via Symfony Messenger (création d'un message/handler).
    - La réponse HTTP JSON peut être simple (par exemple, un statut 200 et un message de confirmation).

4. Utilisation de Symfony Messenger

- **Objectif** : Gérer l'envoi asynchrone des SMS.
- **Implémentation** :
    - Créez un message (ex. : `AlertMessage`) contenant les informations nécessaires pour envoyer l'alerte (numéro de téléphone, contenu du message, …).
    - Créez un handler qui utilisera le `SmsService` pour simuler l'envoi.
    - La commande `/alerter` devra dispatcher un message pour chaque destinataire.
- **Utilisation** : Les messages doivent être consommés de manière asynchrone, par la commande php bin/console messenger:consume

5. Gestion des Migrations SQL

- **Objectif** : Créer et gérer la structure de la base de données sans ORM.
- **Bundle** : Utiliser Doelia/sql-migrations-bundle pour créer vos migrations SQL.
- **Table** : Une table pour stocker les destinataires avec les champs `insee`, `telephone`

Facultatif : Si vous avez le temps

Sécurisation de l'Endpoint `/alerter` via Clé d'API

- **Objectif** : Ajouter une couche de sécurité à l'endpoint `/alerter`.
- **Mécanisme** :
    - La sécurisation doit se faire via une clé d'API.
    - La clé d'API devra être transmise via un header HTTP (ex. : `X-API-KEY`) ou en paramètre de requête.

- La clé attendue sera définie dans la configuration (par exemple, dans le fichier `.env` ou via les paramètres de Symfony).
- Si la clé n'est pas présente ou ne correspond pas à celle attendue, la requête doit être rejetée avec un statut HTTP 403 (Forbidden) ou 401 (Unauthorized).

Attentes et Qualités du Code

- **Code soigné et lisible** : Respectez les standards PSR, structurez votre code de façon modulaire et testable.
- **Simplicité** : La solution ne doit pas être sur-engineered. Le but est de montrer votre capacité à structurer un projet Symfony professionnel.
- **Logs** : Utilisez le système de logging de Symfony pour tracer les actions importantes (import, envoi de SMS, erreurs, etc.).
- **Documentation** : Fournir un README clair expliquant :
  - Les prérequis pour exécuter le projet en local.
  - Les commandes à utiliser pour installer les dépendances.
  - La configuration (ex. : paramètres de connexion à la base de données, clé d'API, etc.).
  - Comment exécuter les migrations SQL.
  - Comment importer un fichier CSV via la commande.
  - Comment lancer l'endpoint `/alerter` et vérifier les logs d'envoi des SMS.

Livrables

- **Code Source** : L'intégralité du projet, idéalement sous forme d'un repository Git.
- **README.md** : Un fichier de documentation qui explique clairement comment installer et utiliser le projet en local.

Bonne chance pour la réalisation de ce test technique ! Nous attendons une solution qui reflète à la fois vos compétences techniques et votre rigueur dans la mise en œuvre d'un projet professionnel.