```
**Objectif :**
Créer une application permettant aux utilisateurs de générer automatiquement
des adresses et des numéros de téléphone en fonction des pays choisis,
facilitant ainsi les achats en ligne. L'application doit fournir des
fonctionnalités de gestion des comptes et garantir la sécurité des
informations.

**Fonctionnalités Principales :**

1. **Authentification et Gestion de Compte :**
   - `POST /api/auth/register` : Créer un nouveau compte utilisateur avec
validation par e-mail ou SMS.
   - `POST /api/auth/login` : Authentifier un utilisateur existant.
   - `GET /api/auth/logout` : Déconnecter un utilisateur.
   - `POST /api/auth/verify-email` : Vérifier l'e-mail de l'utilisateur.
   - `POST /api/auth/verify-phone` : Vérifier le numéro de téléphone de
l'utilisateur.
   - `POST /api/auth/forgot-password` : Envoyer un lien de réinitialisation
de mot de passe par e-mail ou SMS.
   - `POST /api/auth/reset-password` : Réinitialiser le mot de passe après
validation du lien.

2. **Génération d'Adresse et Numéro de Téléphone :**
   - `GET /api/address/:country` : Générer une adresse aléatoire en fonction
du pays spécifié.
   - `GET /api/phone/:country` : Générer un numéro de téléphone aléatoire
dans la liste de contacts prédéfinie pour le pays spécifié.
   - `GET /api/user/addresses` : Obtenir la liste des adresses uniques de
l'utilisateur.
   - `GET /api/user/phones` : Obtenir la liste des numéros de téléphone
uniques de l'utilisateur.



**Sécurité :**
```

```
   - Utilisation de JWT (JSON Web Tokens) pour l'authentification.
   - Validation de l'e-mail et du numéro de téléphone pour éviter les abus.
   - Protection contre les attaques par force brute, les injections SQL,
etc.

**Technologies :**
   - Backend : Django.
   - Base de données : SQLite (pour le développement, peut être migré vers
un autre RDBMS en production).
   - Authentification : JWT avec vérification par e-mail ou SMS.
   - Communication en temps réel : Channels (Django Channels, pour les
notifications en temps réel).

**Remarques :**
   - Les adresses générées doivent être valides pour le pays spécifié.
   - Les numéros de téléphone générés doivent respecter le format du pays
spécifié.
   - Les fonctionnalités métier doivent être testées minutieusement pour
garantir une génération correcte des adresses et des numéros de téléphone.

**Diagramme de Classe :**

```plaintext
+----------------+       +-------------------+       +---------------------+
|   User         |       |   Address         |       |   PhoneNumber       |
+----------------+       +-------------------+       +---------------------+
| id             |1----* | id                |       | id                  |
| username       |       | street            |1----* | number              |
| email          |1----* | city              |       | country             |
| password       |       | state/province    |       | user_id             |
| phone_number   |1----* | postal_code       |       +---------------------+
| is_email_verified |     | country           |
| is_phone_verified |     | user_id           |
+----------------+       +-------------------+
```
```

#Admin Section

```
```

Cahier des Charges pour l'Application de Génération d'Adresses et de Numéros de Téléphone

1. Objectif du Projet :

- Développer une application web permettant aux utilisateurs de générer des adresses et des numéros de téléphone pour faciliter les achats en ligne.

2. Fonctionnalités Principales :

Génération d'Informations :

- Les utilisateurs peuvent générer automatiquement des adresses pour différents pays.
- Les utilisateurs peuvent générer des numéros de téléphone pour différents pays.

Authentification et Gestion de Compte :

- Inscription avec vérification par e-mail ou SMS.
- Connexion avec validation de deux facteurs (2FA).
- Réinitialisation du mot de passe via e-mail ou SMS.

Gestion de Compte Utilisateur :

- Profil utilisateur avec informations de base.
- Historique des adresses générées.
- Historique des numéros de téléphone générés.
- Possibilité de mettre à jour les informations du compte.

Administration :

- Interface d'administration sécurisée.
- Visualisation des comptes utilisateurs.
- Possibilité de bloquer/débloquer des comptes.
- Journal d'activité pour suivre les actions administratives.

Sécurité :

- Protection contre les attaques par force brute.
- Stockage sécurisé des informations sensibles.
- Transmission sécurisée des données via HTTPS.

3. Endpoints pour la Partie Admin :

Endpoints pour la Gestion de Comptes Utilisateurs (Côté Administrateur) :

1. **Liste des Utilisateurs :**
    - **Endpoint :** `/admin/users`
    - **Méthode :** GET
    - **Description :** Récupère la liste complète des utilisateurs enregistrés.
2. **Détails d'un Utilisateur Spécifique :**
    - **Endpoint :** `/admin/users/:userId`
    - **Méthode :** GET
    - **Description :** Récupère les détails d'un utilisateur spécifique.
3. **Mise à Jour des Informations Utilisateur :**
    - **Endpoint :** `/admin/users/:userId/update`
    - **Méthode :** PUT
    - **Description :** Permet à l'administrateur de mettre à jour les informations d'un utilisateur.
4. **Blocage d'un Compte Utilisateur :**
    - **Endpoint :** `/admin/users/:userId/block`
    - **Méthode :** POST
    - **Description :** Permet à l'administrateur de bloquer le compte d'un utilisateur.
5. **Déblocage d'un Compte Utilisateur :**
    - **Endpoint :** `/admin/users/:userId/unblock`
    - **Méthode :** POST
    - **Description :** Permet à l'administrateur de débloquer le compte d'un utilisateur.
6. **Suppression d'un Compte Utilisateur :**
    - **Endpoint :** `/admin/users/:userId/delete`
    - **Méthode :** DELETE
    - **Description :** Permet à l'administrateur de supprimer définitivement le compte d'un utilisateur.
7. **Historique des Activités Utilisateur :**
    - **Endpoint :** `/admin/users/:userId/activity-log`

- **Méthode :** GET
- **Description :** Récupère l'historique des activités d'un utilisateur, telles que les connexions, les modifications de profil, etc.

8. **Réinitialisation du Mot de Passe Utilisateur :**
   - **Endpoint :** `/admin/users/:userId/reset-password`
   - **Méthode :** POST
   - **Description :** Permet à l'administrateur de réinitialiser le mot de passe d'un utilisateur.

9. **Liste des Utilisateurs Bloqués :**
   - **Endpoint :** `/admin/blocked-users`
   - **Méthode :** GET
   - **Description :** Récupère la liste des utilisateurs actuellement bloqués.

10. **Statistiques Utilisateurs :**
    - **Endpoint :** `/admin/users/statistics`
    - **Méthode :** GET
    - **Description :** Récupère des statistiques sur les utilisateurs, telles que le nombre total, le nombre actif, le nombre bloqué, etc.

Note : Les noms des endpoints peuvent être ajustés en fonction des conventions et des besoins spécifiques du projet.

4. Sécurité :

- Utilisation de JWT (JSON Web Tokens) pour l'authentification.
- Utilisation de HTTPS pour sécuriser la transmission des données.
- Protection contre les injections SQL.

5. Technologies :

- Backend : Django
- Frontend : Vue 3 avec TypeScript
- Base de données : PostgreSQL
- Authentification : JWT

6. Tests :

- Tests unitaires pour chaque fonctionnalité.
- Tests d'intégration pour vérifier les interactions entre les composants.

7. Déploiement :

- Déploiement sur une infrastructure sécurisée.
- Configuration d'un serveur web tel que Nginx.

8. Documentation :

- Documentation détaillée pour l'installation, la configuration et l'utilisation de l'application.
- Commentaires dans le code pour assurer une compréhension facile.

Remarques :

- La sécurité des informations des utilisateurs est une priorité.
- L'interface d'administration doit être conviviale et sécurisée.
- Les journaux d'activité doivent être complets et compréhensibles.

*Note : Le cahier des charges est une proposition générale, et il peut être ajusté en fonction des besoins spécifiques du projet. ```

Ce diagramme de classe présente les principales entités du système avec leurs relations. Chaque utilisateur peut avoir plusieurs adresses et numéros de téléphone associés. L'authentification vérifie si l'e-mail et le numéro de téléphone sont vérifiés avant d'accorder l'accès aux fonctionnalités métier.```

La mise en place d'une démarche DevOps pour votre projet est cruciale pour assurer un développement efficace, des déploiements fréquents et une gestion transparente des opérations. Voici une démarche DevOps recommandée pour votre projet :

1. Mise en Place des Outils de Gestion de Code Source :

- Utilisez un système de contrôle de version comme Git.
- Hébergez votre code source sur une plateforme telle que GitHub, GitLab, ou Bitbucket.

2. Automatisation des Tests :

- Intégrez des tests unitaires, des tests d'intégration, et des tests fonctionnels dans votre pipeline.
- Utilisez des outils comme Jest, Vue Test Utils, ou Cypress pour les tests.

3. Intégration Continue (CI) :

- Configurez des pipelines CI/CD pour automatiser la construction, les tests et le déploiement.
- Utilisez des outils tels que Jenkins, GitLab CI, GitHub Actions.

4. Automatisation des Déploiements (CD) :

- Automatisez le déploiement sur des environnements de développement, de test, et de production.
- Utilisez des outils comme Ansible, Docker, Kubernetes.

5. Gestion des Configurations :

- Utilisez des outils de gestion de configurations pour garantir la cohérence des environnements.
- Ansible, Puppet, ou Chef peuvent être utiles.

6. Infrastructure en tant que Code (IaC) :

- Définissez votre infrastructure en tant que code pour une gestion efficace.
- Utilisez Terraform, AWS CloudFormation, ou Azure Resource Manager.

7. Sécurité :

- Intégrez des outils de sécurité dans le pipeline pour identifier les vulnérabilités.
- Utilisez des outils comme SonarQube, OWASP Dependency-Check.

8. Monitoring et Logging :

- Intégrez des solutions de surveillance pour suivre les performances de l'application.

- Utilisez des outils tels que Prometheus, Grafana, ELK Stack.

9. Gestion des Incidents :

- Mettez en place un processus de gestion des incidents pour réagir rapidement aux problèmes de production.
- Utilisez des outils de gestion des incidents comme PagerDuty.

10. Documentation Automatisée :

- Maintenez une documentation à jour automatiquement générée depuis le code.
- Utilisez des outils comme Swagger ou ReDoc pour l'API, et des générateurs de documentation.

11. Formation de l'Équipe :

- Assurez-vous que l'équipe est formée aux pratiques DevOps.
- Organisez des sessions de formation sur les outils et les processus DevOps.

12. Amélioration Continue :

- Effectuez régulièrement des rétrospectives pour identifier les zones d'amélioration.
- Adoptez une approche itérative pour affiner continuellement les processus.

En suivant cette démarche, vous pouvez créer un environnement de développement collaboratif, automatisé et fiable, tout en garantissant la sécurité et la stabilité de votre application.