Rapport Technique

ManitoDex

Khashan Damien

Rapport Technique sur les Migrations

1. Migration : 2024_06_15_154831_create_categories_table.php

Description de la table : La table `categories` est utilisée pour stocker les différentes catégories auxquelles un Pokémon peut appartenir. Chaque catégorie possède un nom unique et un lien vers une image.

Colonnes et types de données :

- `id` : entier, clé primaire.
- `name` : chaîne de caractères, unique.
- `imgLink` : chaîne de caractères.

Contraintes d'intégrité :

- `id` est la clé primaire de la table.
- `name` est une contrainte unique, assurant que chaque catégorie a un nom distinct.

Relations entre les tables :

- Aucune relation directe avec d'autres tables.

2. Migration : 2024_06_16_000000_create_types_table.php

Description de la table : La table `types` est utilisée pour stocker les différents types associés aux Pokémon et aux attaques, comme "Feu", "Eau", etc.

Colonnes et types de données :

- `id` : entier, clé primaire.
- `name` : chaîne de caractères, unique.
- `imgLink` : chaîne de caractères.

Contraintes d'intégrité :

- `id` est la clé primaire de la table.
- `name` est une contrainte unique.

Relations entre les tables :

- Cette table est liée aux tables pokemon et `attacks` via des clés étrangères, indiquant qu'un Pokémon ou une attaque appartient à un type spécifique.

3. Migration : 2024_06_16_000001_create_pokemon_table.php

Description de la table : La table pokemon est utilisée pour stocker les informations relatives aux Pokémon. Chaque Pokémon appartient à une catégorie et peut avoir un type.

Colonnes et types de données :

- `id` : entier, clé primaire.
- `name` : chaîne de caractères, unique.
- `category_id` : entier, clé étrangère vers la table `categories`.
- `type_id` : entier, clé étrangère vers la table `types`.

Contraintes d'intégrité :

- `id` est la clé primaire de la table.
- `name` est une contrainte unique.
- `category_id` est une clé étrangère référencée à `categories(id)`.
- `type_id` est une clé étrangère référencée à `types(id)`.

Relations entre les tables :

- La table pokemon est liée à `categories` par `category_id`.
- La table pokemon est liée à `types` par `type_id`.

4. Migration : 2024_06_16_122143_create_attacks_table.php

Description de la table : La table `attacks` est utilisée pour stocker les différentes attaques que peuvent utiliser les Pokémon. Chaque attaque a un nom unique, une puissance, et est associée à un type.

Colonnes et types de données :

- `id` : entier, clé primaire.
- `name` : chaîne de caractères, unique.
- `power` : entier.
- `type_id` : entier, clé étrangère vers la table `types`.

Contraintes d'intégrité :

- `id` est la clé primaire de la table.
- `name` est une contrainte unique.
- `type_id` est une clé étrangère référencée à `types(id)`.

Relations entre les tables :

- La table `attacks` est liée à `types` par `type_id`.

Rapport Technique Controller Admin

1. Introduction Générale

Ce rapport technique porte sur trois contrôleurs principaux dans une application Laravel : `AttackAdminController`, `PokedexAdminController`, et `TypeAdminController`. Chacun de ces contrôleurs est responsable de la gestion d'une ressource spécifique (attaques, Pokémon, et types) dans la section d'administration de l'application. Les fonctionnalités couvrent la création, la mise à jour, l'affichage, et la suppression de ces entités.

2. Détails des Contrôleurs

2.1. AttackAdminController

Fonctionnalité Principale :

Le contrôleur `AttackAdminController` gère toutes les opérations CRUD (Create, Read, Update, Delete) pour les attaques. Cela inclut la validation des données d'entrée via des requêtes spécifiques, la gestion des relations avec les types et les catégories, ainsi que le stockage des fichiers liés aux images.

Méthodes Clés :

- **`index()` :** Récupère et affiche toutes les attaques.
  - Modèle utilisé : `Attack::all()`
  - Vue retournée : `admin.attackadmin.index`
- **`create()` :** Prépare la vue pour créer une nouvelle attaque.

    - Modèle utilisé : `Attack::all()`
    - Vue retournée : `admin.attackadmin.create`
- **`store(AttackCreateRequest $request)` :** Enregistre une nouvelle attaque dans la base de données après validation.
    - Données validées : `name`, `power`, `accuracy`, `maxpp`, `description`, `category_id`, `type_id`.
    - Gestion des images : Les fichiers d'images sont stockés dans le dossier public (`/storage/images/category` et `/storage/images/type`).
- **`edit(Attack $attack)` :** Prépare la vue pour modifier une attaque existante.
    - Modèle utilisé : `Type::all()` pour récupérer tous les types.
    - Vue retournée : `admin.attackadmin.edit`
- **`update(AttackUpdateRequest $request, Attack $attack)` :** Met à jour une attaque existante dans la base de données après validation.
    - Données mises à jour : Similaires à celles de la méthode `store`.
- **`destroy(Attack $attack)` :** Supprime une attaque de la base de données.

2.2. PokedexAdminController

Fonctionnalité Principale :

Le `PokedexAdminController` gère toutes les opérations liées aux Pokémon, incluant la création, la mise à jour, la recherche, et la suppression des enregistrements dans le Pokédex. Ce contrôleur inclut également la gestion des relations avec les types, permettant à un Pokémon d'avoir jusqu'à deux types.

Méthodes Clés :

- **`index(Request $request)` :** Affiche une liste de Pokémon, avec la possibilité de filtrer les résultats par nom et type.
    - Recherche par nom : `where('name', 'LIKE', '%'.$request->search.'%')`
    - Recherche par type : `whereHas('type1')` ou `whereHas('type2')`
    - Vue retournée : `admin.pokedexadmin.index`
- **`create(Pokemon $pokemon)` :** Prépare la vue pour ajouter un nouveau Pokémon.
    - Modèle utilisé : `Type::all()` pour récupérer tous les types.
    - Vue retournée : `admin.pokedexadmin.create`
- **`store(PokemonCreateRequest $request)` :** Enregistre un nouveau Pokémon après validation.

    - Données validées : `name`, `description`, `hp`, `att`, `attSpe`, `def`, `defSpe`, `vit`, `size`, `weight`, `type1_id`, `type2_id`.
    - Gestion des images : Les fichiers d'images sont stockés dans le dossier public (`/storage/images/pokemon`).
- **`edit(Pokemon $pokemon)` :** Prépare la vue pour modifier un Pokémon existant.
    - Vue retournée : `admin.pokedexadmin.edit`
- **`update(PokemonUpdateRequest $request, Pokemon $pokemon)` :** Met à jour un Pokémon existant dans la base de données après validation.
    - Données mises à jour : Similaires à celles de la méthode `store`.
- **`destroy(Pokemon $pokemon)` :** Supprime un Pokémon de la base de données.

2.3. TypeAdminController

Fonctionnalité Principale :

Le `TypeAdminController` gère les opérations CRUD pour les types. Les types sont utilisés pour catégoriser les Pokémon et les attaques, comme "Feu", "Eau", etc.

Méthodes Clés :

- **`index()` :** Récupère et affiche tous les types.
    - Modèle utilisé : `Type::all()`
    - Vue retournée : `admin.typeadmin.index`
- **`create()` :** Prépare la vue pour créer un nouveau type.
    - Vue retournée : `admin.typeadmin.create`
- **`store(TypeCreateRequest $request)` :** Enregistre un nouveau type après validation.
    - Données validées : `name`
    - Gestion des images : Les fichiers d'images sont stockés dans le dossier public (`/storage/images/type`).
- **`edit(Type $type)` :** Prépare la vue pour modifier un type existant.
    - Vue retournée : `admin.typeadmin.edit`
- **`update(TypeUpdateRequest $request, Type $type)` :** Met à jour un type existant après validation.
    - Données mises à jour : Similaires à celles de la méthode `store`.
- **`destroy(Type $type)` :** Supprime un type de la base de données.