**Rapport de Conception - EuromedConnect**

**Introduction**

EuromedConnect est une application de réseau social développée avec Laravel 12. Elle utilise Breeze pour l'authentification, Blade pour les vues, et Tailwind CSS pour le style. L'application permet aux utilisateurs de créer des posts, de commenter, de liker, de suivre d'autres utilisateurs, et de gérer leur profil de manière fluide et sécurisée.

**Architecture du Projet**

L'architecture est construite sur le modèle MVC (Modèle-Vue-Contrôleur) de Laravel :

* **Modèles** : User, Post, Comment, Like, Follow
* **Vues** : Gérées par Blade, stylisées avec Tailwind CSS
* **Contrôleurs** : ProfileController, PostController, SearchController, UserController, FollowController
* **Routes** : Organisées dans le fichier web.php, avec une gestion fine des accès par middleware (auth, verified).

**Modèle Conceptuel de Données (MCD)**

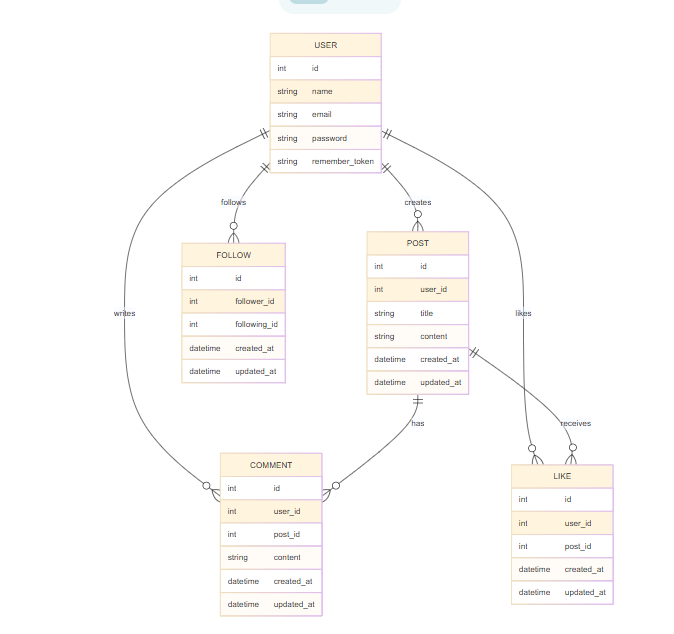
Le MCD représente les entités principales :

* User
* Post
* Comment
* Like
* Follow

Les relations entre les entités sont les suivantes :

* Un utilisateur peut créer plusieurs posts, commentaires, et likes.
* Un utilisateur peut suivre d'autres utilisateurs (relation de plusieurs à plusieurs).
* Un post peut avoir plusieurs commentaires et likes.

*(Le diagramme est disponible sous forme Mermaid pour une meilleure visualisation)*

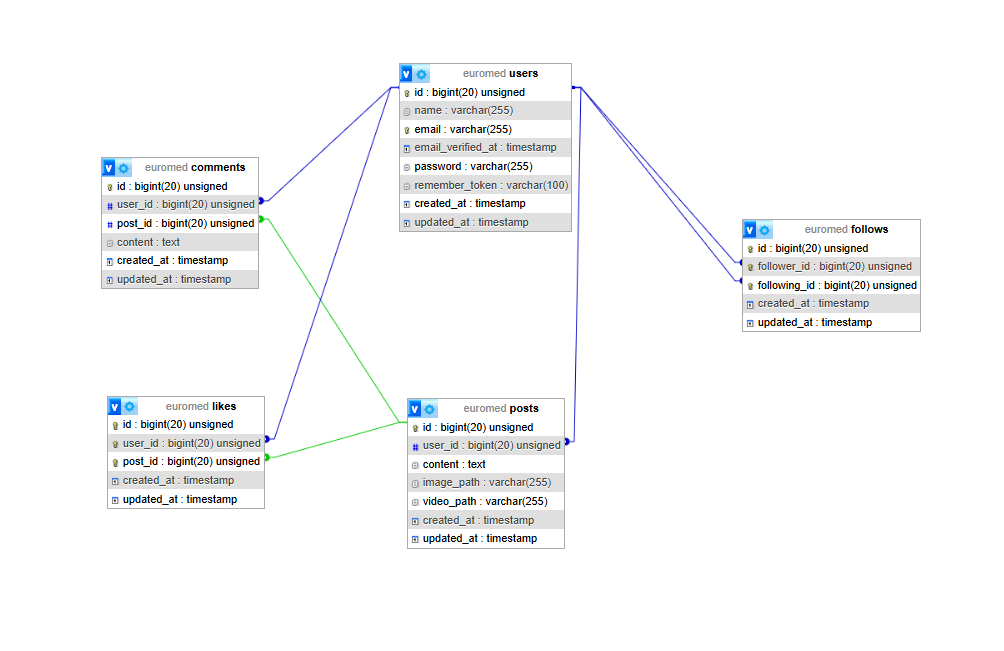


**Modèle Logique de Données (MLD)**

Le MLD est le reflet de la base de données :

* **users** : id, name, email, password, remember\_token...
* **posts** : id, user\_id, title, content...
* **comments** : id, user\_id, post\_id, content...
* **likes** : id, user\_id, post\_id
* **follows** : id, follower\_id, following\_id

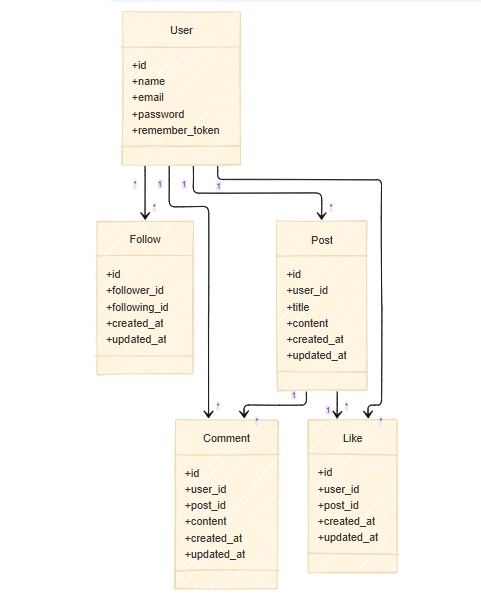
Chaque table est reliée via des clés étrangères pour assurer l'intégrité référentielle.



**Diagramme de Classes UML**

Un diagramme UML structure les différentes entités et leurs relations. Les relations sont :

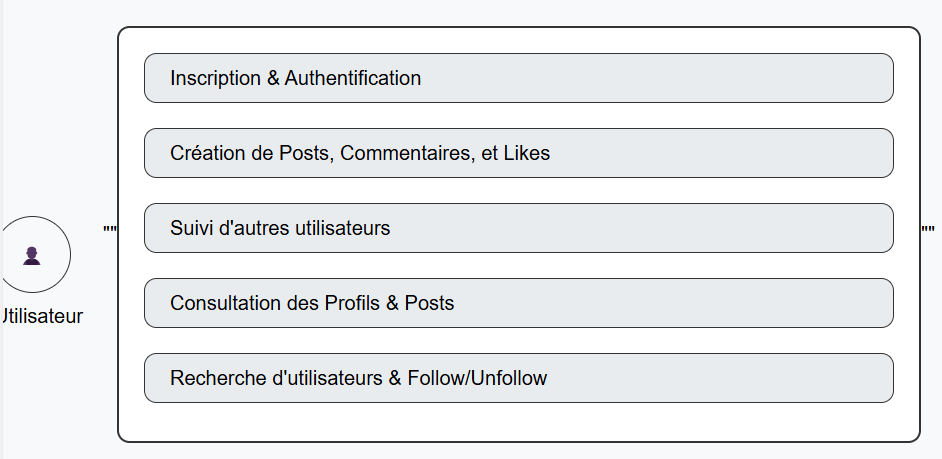
* 1:N entre User et Post
* 1:N entre User et Comment
* 1:N entre User et Like
* N:N entre User et Follow (pivot)



**Cas d'Utilisation**

Les cas d'utilisation principaux sont :

* Inscription et Authentification (gérées par Breeze)
* Création de posts, commentaires, et likes
* Suivi d'autres utilisateurs
* Consultation des profils et des posts des utilisateurs suivis



**Détails des Routes et Contrôleurs**

Voici un aperçu des routes définies dans l'application :

* /home → Affichage des posts
* /dashboard → Vue du tableau de bord
* /profile → Modification du profil
* /profile/{user} → Affichage du profil d'un utilisateur spécifique
* /search → Recherche d'utilisateurs ou de posts
* /posts → Création, modification et suppression de posts
* /posts/{post}/comments → Ajout et suppression de commentaires
* /posts/{post}/like → Ajout et suppression de likes
* /users/{user}/follow → Suivre ou ne plus suivre un utilisateur

Les contrôleurs associés sont bien structurés pour interagir avec les modèles et gérer la logique de l'application.

**Authentification avec Breeze**

L'application utilise **Breeze**, le starter kit officiel de Laravel pour l'authentification :

* **Inscription** : les utilisateurs peuvent s'inscrire avec un email et un mot de passe.
* **Connexion/Déconnexion** : accès sécurisé via middleware auth.
* **Réinitialisation de mot de passe** : géré directement par Breeze.
* **Vérification d'email** : Breeze permet également la vérification d'adresse email.

**Interface Utilisateur (Blade + Tailwind CSS)**

Les vues sont gérées par **Blade**, le moteur de template de Laravel, et stylisées avec **Tailwind CSS** pour une apparence moderne et responsive. Les pages sont structurées comme suit :

* **Accueil (home.blade.php)** : affichage des posts des utilisateurs suivis.
* **Profil (profile.blade.php)** : gestion des informations personnelles et des publications.
* **Recherche (search.blade.php)** : affichage des résultats de recherche d'utilisateurs.
* **Post (post.blade.php)** : création, affichage et gestion des commentaires et likes.

**Schéma de Navigation Général**

Le schéma de navigation suit une logique fluide :

1. **Page d'accueil** → Consultation des posts des utilisateurs suivis.
2. **Recherche** → Permet de découvrir de nouveaux utilisateurs.
3. **Profil** → Permet à l'utilisateur de gérer ses informations personnelles et de visualiser ses publications.
4. **Création de Post** → Interface claire pour ajouter de nouveaux contenus.
5. **Interactions sociales** → Liker, commenter, et suivre d'autres utilisateurs.

**Conclusion**

EuromedConnect est un réseau social intuitif et bien structuré, construit avec les bonnes pratiques de Laravel. L'utilisation de Breeze simplifie l'authentification, et Blade associé à Tailwind CSS offre une interface utilisateur moderne et responsive.

Le modèle de données est optimisé pour permettre des interactions sociales fluides, et les contrôleurs gèrent efficacement les opérations CRUD.

L'application est prête à évoluer pour intégrer de nouvelles fonctionnalités tout en conservant sa robustesse et sa simplicité d'utilisation.

""# Connexion MySQL et Eloquent ORM dans Laravel

## 1️⃣ Connexion à la base de données MySQL

Laravel utilise un fichier de configuration appelé .env pour gérer les paramètres de connexion à la base de données. Pour se connecter à MySQL, les paramètres suivants doivent être définis dans le fichier .env :

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=nom\_de\_la\_base

DB\_USERNAME=nom\_utilisateur

DB\_PASSWORD=mot\_de\_passe

Une fois les informations correctement renseignées, Laravel est capable d'interagir avec la base de données MySQL.

## 2️⃣ Modèles et Eloquent ORM

Laravel utilise **Eloquent ORM** pour interagir avec la base de données de manière fluide et intuitive. Chaque table dans la base de données correspond à un modèle dans Laravel.

### Modèle User

Le modèle User est lié à la table users dans la base de données. Il possède plusieurs relations :

* **hasMany** avec les modèles Post, Comment, et Like (un utilisateur peut avoir plusieurs posts, commentaires, et likes).
* **belongsToMany** avec lui-même pour les relations de suivi (follows).

public function posts(): HasMany

{

return $this->hasMany(Post::class);

}

public function comments(): HasMany

{

return $this->hasMany(Comment::class);

}

public function likes(): HasMany

{

return $this->hasMany(Like::class);

}

public function following(): BelongsToMany

{

return $this->belongsToMany(User::class, 'follows', 'follower\_id', 'following\_id');

}

public function followers(): BelongsToMany

{

return $this->belongsToMany(User::class, 'follows', 'following\_id', 'follower\_id');

}

### Modèle Post

Le modèle Post représente les publications de l'utilisateur et est lié à la table posts. Un post appartient à un utilisateur (belongsTo) et peut avoir plusieurs commentaires et likes.

public function user()

{

return $this->belongsTo(User::class);

}

public function comments()

{

return $this->hasMany(Comment::class);

}

public function likes()

{

return $this->hasMany(Like::class);

}

### Modèle Comment

Le modèle Comment est lié à la table comments. Il est associé à un utilisateur et un post via des relations belongsTo.

public function user()

{

return $this->belongsTo(User::class);

}

public function post()

{

return $this->belongsTo(Post::class);

}

### Modèle Like

Le modèle Like représente les "j'aime" qu'un utilisateur peut attribuer à un post. Il est également associé à un utilisateur et un post.

public function user()

{

return $this->belongsTo(User::class);

}

public function post()

{

return $this->belongsTo(Post::class);

}

### Modèle Follow

Le modèle Follow gère les relations de suivi entre utilisateurs (follower et following). Il utilise un pivot dans la base de données nommé follows.

public function follower()

{

return $this->belongsTo(User::class, 'follower\_id');

}

public function following()

{

return $this->belongsTo(User::class, 'following\_id');

}

## 3️⃣ Schéma de la base de données

Le schéma de la base de données est structuré de manière à refléter ces relations :

* users: contient les informations des utilisateurs.
* posts: contient les publications créées par les utilisateurs.
* comments: contient les commentaires laissés par les utilisateurs sur les posts.
* likes: contient les informations sur les likes attribués.
* follows: représente les relations de suivi entre utilisateurs.

## 4️⃣ Conclusion

Grâce à Eloquent ORM, les relations sont facilement accessibles sans avoir à écrire de SQL complexe. Les méthodes comme hasMany, belongsTo, et belongsToMany permettent de naviguer entre les entités de manière fluide, rendant le code plus lisible et maintenable.  
""