LARAVEL – FRAMEWORK PHP
3-0

L'objectif est d'installer une application laravel avec Breeze.

Puis d'installer un contrôleur de ressources qui gère des tâches : ajouter, consulter, supprimer.

C'est une première approche, assez rapide, qui permet d'appréhender l'efficacité de Laravel, mais aussi sa complexité.

0 : Table des matières

Table des matières

1 : Cahier des charges : Application de gestion de tâches (To-Do List) avec Laravel et Breeze

1️⃣ Présentation du projet

L'objectif est de développer une **application web de gestion de tâches** où les utilisateurs peuvent **créer, modifier et supprimer leurs tâches personnelles**. L'application doit être sécurisée avec une authentification utilisateur via **Laravel Breeze**.

2️⃣ Fonctionnalités principales

✅ Gestion des utilisateurs

- Inscription et connexion avec **Laravel Breeze**
- Mot de passe oublié / réinitialisation
- Interface de profil pour modifier nom et email

✅ Gestion des tâches

- Ajouter une nouvelle tâche
- Modifier une tâche
- Marquer une tâche comme **complétée / non complétée**
- Supprimer une tâche
- Lister toutes ses tâches avec un système de **filtrage** (terminées, en cours)

✅ Organisation avancée

- Catégoriser les tâches (ex. : "Travail", "Personnel", "Courses")
- Ajouter une **date d'échéance** à une tâche
- Trier les tâches par **priorité** ou **date d'échéance**

✅ Expérience utilisateur (UX)

- Interface minimaliste et responsive (Bootstrap ou Tailwind CSS)
- Confirmation avant suppression d’une tâche
- Messages de succès / erreur

3️⃣ Rôles des utilisateurs

- **Utilisateur simple** : peut gérer ses propres tâches
- **Admin (optionnel)** : peut voir tous les utilisateurs et leurs tâches

4️⃣ Contraintes techniques

- Backend : **Laravel 10+**
- Authentification : **Laravel Breeze** (avec Jetstream optionnel pour plus de fonctionnalités)
- Base de données : **MySQL**
- Frontend : **Blade + Tailwind CSS**

5️⃣ Cas d'utilisation

✅ **Alice crée un compte** et se connecte
✅ Elle ajoute une tâche **"Réviser Laravel"** pour lundi
✅ Elle modifie la tâche pour y ajouter une **priorité haute**
✅ Le lendemain, elle coche la tâche comme **complétée**
✅ Elle filtre sa liste pour voir uniquement les tâches **en cours**
✅ Elle supprime une tâche qu'elle n’a plus besoin d’effectuer

6️⃣ Améliorations possibles

- Ajouter une **fonction de recherche**
- Permettre le **partage de tâches** entre utilisateurs
- Ajouter des **rappels par email**

Ce projet est parfait pour apprendre **Laravel, Breeze et les bonnes pratiques MVC** ! 😃
Tu veux des suggestions sur l'architecture de la base de données ?

2 : Étapes de réalisation du projet To-Do List avec Laravel et Breeze

Ce guide détaille **les étapes clés** pour développer une application Laravel **To-Do List** avec authentification via **Laravel Breeze**.

☐Installation et configuration du projet

✅ Installer Laravel

Dans ton terminal, exécute la commande suivante :

```
composer create-project laravel/laravel matodolist_e3_0
cd matodolist_e3_0
```

Puis, ouvre le projet dans ton éditeur (VS Code, PHPStorm...).

✅ Configurer l'environnement

Dans `.env`, configure la base de données :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE= matodolist_e3_0
DB_USERNAME=root
DB_PASSWORD=
```

💡 **Crée la base de données dans MySQL** (`todo_list`).

```
CREATE DATABASE matodolist_e3_0 ;
```

✅ Installer Laravel Breeze

Breeze gère **l'authentification** (login, inscription, réinitialisation de mot de passe).

Dans ton terminal (dans le dossier « matodolist_e3_0 » , exécute les commandes suivantes, une par une pour y voir clair :

```
composer require laravel/breeze --dev
```
`php artisan breeze:install` : faire tous les choix par défaut
```
php artisan migrate
npm install && npm run dev
```

Dans un autre terminal (dans le dossier « matodolist_e3_0 » , exécute cette commande :

```
php artisan serve
```

✔ **Laravel Breeze est prêt !** Tu peux tester l'inscription et la connexion.

2. Modélisation de la base de données

Nous allons gérer les tâches des utilisateurs. Pour ça, on gère un MVC des tâches.

✅ Créer le modèle et migration Task

Dans un autre terminal (dans le dossier « matodolist_e3_0 » , exécute cette commande :

```
php artisan make:model Task -m
```

Explications :

- **`php artisan make:model Task`** : Crée un modèle appelé `Task` dans le répertoire `app/Models`. Un modèle dans Laravel représente une table dans la base de données et permet de gérer les interactions avec cette table (comme l'ajout, la mise à jour, la suppression et la lecture des données).
- **`-m`** : L'option `-m` (ou `--migration`) crée également une migration pour la table associée au modèle `Task`. Une migration est un fichier PHP qui contient des instructions pour créer ou modifier des tables dans la base de données.

Après avoir exécuté cette commande, vous obtiendrez :

1. Un fichier `Task.php` dans le répertoire `app/Models`.
2. Un fichier de migration dans le répertoire `database/migrations` pour créer une table `tasks` (si vous ne spécifiez pas un nom de table différent dans la migration).

✅ Coder la migration Task

Ouvre **database/migrations/YYYY_MM_DD_create_tasks_table.php** et ajoute :

```
public function up()
{
    Schema::create('tasks', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id')->constrained()->onDelete('cascade');
        $table->string('title');
        $table->text('description')->nullable();
        $table->date('due_date')->nullable();
        $table->boolean('is_completed')->default(false);
        $table->timestamps();
    });
}
```

Explications :

Ce code est une méthode `up()` d'une migration dans Laravel, utilisée pour créer une table `tasks` dans la base de données. Voici un détail de ce que chaque ligne fait :

1. **`Schema::create('tasks', function (Blueprint $table) { ... });`**

Cette ligne utilise la classe `Schema` de Laravel pour créer une nouvelle table appelée `tasks`. Le bloc `function (Blueprint $table)` est une fonction anonyme qui définit les colonnes de la table.

2. **`$table->id();`**

Cela crée une colonne `id` de type entier (par défaut, c'est une clé primaire auto-incrémentée).

3. **`$table->foreignId('user_id')->constrained()->onDelete('cascade');`**

Cette ligne crée une colonne `user_id` de type `foreignId`, qui fait référence à la colonne `id` d'une autre table (généralement, la table `users`). La méthode `constrained()` indique que cette colonne est une clé étrangère. Ensuite, `onDelete('cascade')` signifie que si un utilisateur (de la table `users`) est supprimé, toutes les tâches associées à cet utilisateur seront également supprimées automatiquement.

4. **`$table->string('title');`**

Cela crée une colonne `title` de type chaîne de caractères, utilisée pour stocker le titre de la tâche.

5. **`$table->text('description')->nullable();`**

Cela crée une colonne `description` de type texte, utilisée pour stocker une description plus longue de la tâche. La méthode `nullable()` signifie que cette colonne peut contenir des valeurs `null`, donc une description peut être vide.

6. **`$table->date('due_date')->nullable();`**

Cela crée une colonne `due_date` de type `date`, utilisée pour stocker la date d'échéance de la tâche. Elle peut être `null` si la tâche n'a pas de date d'échéance définie, grâce à la méthode `nullable()`.

7. **`$table->boolean('is_completed')->default(false);`**

Cela crée une colonne `is_completed` de type booléen, utilisée pour indiquer si la tâche est terminée ou non. La valeur par défaut est `false`, ce qui signifie que la tâche n'est pas terminée lors de sa création.

8. **`$table->timestamps();`**

Cela crée deux colonnes `created_at` et `updated_at`, qui sont automatiquement gérées par Laravel pour suivre respectivement la date de création et la dernière modification de la tâche.

✅ **Exécuter la migration :**

Dans un terminal dans le dossier « matodolist_e3_0 » , exécute cette commande :

```
php artisan migrate
```

✅ **Vérifier en BD :**

Dans la BD « matodolist_e3_0 » , on trouve :

```
La table tasks
La table users
```

✅ **Mettre à jour la classe Task en BD :**

Dans app/Models/task.php, on trouve le code de la classe Task.

On va préciser que certains attributs sont saisissables et donc qu'il faut les protéger.

```
class Task extends Model
{
    use HasFactory;

    // Ajoute 'user_id' dans le tableau $fillable pour autoriser
l'assignation de masse
    protected $fillable = [
        'user_id',       // Autorise 'user_id' pour l'assignation de masse
        'title',
        'description',
        'due_date',
        'is_completed',
    ];
}
```

3️⃣ Créer le contrôleur des tâches

✅ Génère le contrôleur :

Dans un terminal dans le dossier « matodolist_e3_0 » , exécute cette commande :

```
php artisan make:controller TaskController --resource
```

Ce contrôleur va contenir **toutes les actions CRUD** pour gérer les tâches.

Explications :

- **`php artisan make:controller`** : Cette partie de la commande est utilisée pour générer un nouveau contrôleur dans Laravel.
- **`TaskController`** : C'est le nom du contrôleur que vous souhaitez créer. Dans ce cas, le contrôleur sera nommé `TaskController`. Il sera créé dans le répertoire `app/Http/Controllers`.
- **`--resource`** : L'option `--resource` génère automatiquement un contrôleur **resourceful**, ce qui signifie qu'il va inclure les méthodes par défaut pour effectuer les actions liées à une ressource. Les méthodes générées seront :
    1. **`index()`** : Affiche une liste de toutes les ressources (tâches dans ce cas).
    2. **`create()`** : Affiche un formulaire pour créer une nouvelle ressource.
    3. **`store()`** : Gère la logique pour enregistrer une nouvelle ressource dans la base de données.
    4. **`show($id)`** : Affiche une ressource spécifique.
    5. **`edit($id)`** : Affiche un formulaire pour éditer une ressource existante.
    6. **`update($id)`** : Gère la logique pour mettre à jour une ressource spécifique dans la base de données.
    7. **`destroy($id)`** : Gère la logique pour supprimer une ressource.

Ces méthodes correspondent aux actions classiques nécessaires pour manipuler une ressource de type "Task", et elles suivent les conventions de Laravel pour un contrôleur de type **resource**.

✅ Vérifie le code :

Regarde le fichier : app/Http/Controllers/TaskController.php
C’est un « contrôleur de ressources ».

✅ Bilan :

On a un « contrôleur de ressources » : une classe qui gère toutes les actions sur la classe Tasks.
On a la classe Task qui est dans app/Models/Task.php

4️⃣ Développer les routes et la logique métier

Dans **routes/web.php**, ajoute :

```
use App\Http\Controllers\TaskController;

Route::middleware(['auth'])->group(function () {
    Route::resource('tasks', TaskController::class);
});
```

Explications :

1. **`Route::middleware(['auth'])->group(function () { ... });`**

- **`middleware(['auth'])`** : Cela applique un **middleware** à toutes les routes définies à l'intérieur de ce groupe de routes. Le middleware `auth` est utilisé pour garantir que seules les personnes authentifiées (connectées) peuvent accéder à ces routes. Si un utilisateur n'est pas connecté, il sera redirigé vers la page de connexion.

En d'autres termes, avant d'exécuter l'action de la route, Laravel vérifie que l'utilisateur est authentifié. Si ce n'est pas le cas, l'utilisateur est redirigé (par défaut) vers la page de connexion.

- **`group(function () { ... });`** : Cette méthode permet de regrouper plusieurs routes sous un même middleware ou configuration. Toutes les routes définies à l'intérieur de ce groupe hériteront du middleware `auth` et de toutes autres configurations ajoutées.

2. **`Route::resource('tasks', TaskController::class);`**

Cette ligne définit une **route de type resource** pour le contrôleur `TaskController`. Cela va automatiquement créer toutes les routes nécessaires pour effectuer des actions CRUD sur la ressource `Task`.

Les routes générées par `Route::resource('tasks', TaskController::class);` seront les suivantes, et toutes seront protégées par le middleware `auth` :

- **GET /tasks** → `TaskController@index` (affiche la liste des tâches)
- **GET /tasks/create** → `TaskController@create` (affiche le formulaire de création d'une tâche)
- **POST /tasks** → `TaskController@store` (enregistre une nouvelle tâche)
- **GET /tasks/{task}** → `TaskController@show` (affiche une tâche spécifique)
- **GET /tasks/{task}/edit** → `TaskController@edit` (affiche le formulaire de modification d'une tâche)
- **PUT/PATCH /tasks/{task}** → `TaskController@update` (met à jour une tâche spécifique)
- **DELETE /tasks/{task}** → `TaskController@destroy` (supprime une tâche spécifique)

5. Implémentation des fonctionnalités

Dans **app/Http/Controllers/TaskController.php**, remplace le contenu par :

```
namespace App\Http\Controllers;

use App\Models\Task;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class TaskController extends Controller
{
    public function index()
    {
        $tasks = Task::where('user_id', Auth::id())->orderBy('due_date')-
>get();
        return view('tasks.index', compact('tasks'));
    }

    public function create()
    {
        return view('tasks.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'nullable|string',
            'due_date' => 'nullable|date',
        ]);

        Task::create([
            'user_id' => Auth::id(),
            'title' => $request->title,
            'description' => $request->description,
            'due_date' => $request->due_date,
        ]);

        return redirect()->route('tasks.index')->with('success', 'Tâche
créée !');
    }

    public function edit(Task $task)
    {
        if ($task->user_id !== Auth::id()) abort(403);
        return view('tasks.edit', compact('task'));
    }

    public function update(Request $request, Task $task)
    {
        if ($task->user_id !== Auth::id()) abort(403);

        $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'nullable|string',
            'due_date' => 'nullable|date',
        ]);

        $task->update($request->all());
```

```
        return redirect()->route('tasks.index')->with('success', 'Tâche
mise à jour !');
    }

    public function destroy(Task $task)
    {
        if ($task->user_id !== Auth::id()) abort(403);

        $task->delete();
        return redirect()->route('tasks.index')->with('success', 'Tâche
supprimée !');
    }
}
```

Explications courtes :

- **`index()`** : Récupère les tâches de l'utilisateur authentifié et les affiche, triées par date d'échéance.
- **`create()`** : Affiche le formulaire pour créer une nouvelle tâche.
- **`store(Request $request)`** : Valide les données, crée une nouvelle tâche pour l'utilisateur authentifié, puis redirige vers la liste des tâches avec un message de succès.
- **`edit(Task $task)`** : Vérifie si la tâche appartient à l'utilisateur authentifié, puis affiche le formulaire de modification.
- **`update(Request $request, Task $task)`** : Vérifie la propriété de la tâche, valide les données, met à jour la tâche et redirige vers la liste des tâches avec un message de succès.
- **`destroy(Task $task)`** : Vérifie la propriété de la tâche, supprime la tâche, puis redirige vers la liste des tâches avec un message de succès.

6️⃣ Création des vues avec Blade et Tailwind CSS

Dans **resources/views/layouts/app.blade.php**, assure-toi d'inclure Tailwind CSS.

Cette commande le fait :

```
    <!-- Scripts -->
    @vite(['resources/css/app.css', 'resources/js/app.js'])
```

Sinon ça peut être :

```
    <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">
```

✅ Page liste des tâches

Créer le fichier :

resources/views/tasks/index.blade.php :

Dans **resources/views/tasks/index.blade.php** :

```
@extends('layouts.app')

@section('content')
<div class="max-w-4xl mx-auto p-4">
    <h1 class="text-2xl font-bold mb-4">Mes tâches</h1>

    <a href="{{ route('tasks.create') }}" class="bg-blue-500 text-white px-
4 py-2 rounded">+ Nouvelle tâche</a>

    @foreach($tasks as $task)
    <div class="border p-4 mt-4">
        <h2 class="text-lg font-semibold">{{ $task->title }}</h2>
        <p>{{ $task->description }}</p>
        <p class="text-sm text-gray-600">Échéance : {{ $task->due_date
}}</p>
        <div class="mt-2">
            <a href="{{ route('tasks.edit', $task) }}" class="text-blue-
500">Modifier</a>
            <form action="{{ route('tasks.destroy', $task) }}"
method="POST" class="inline-block">
                @csrf
                @method('DELETE')
                <button type="submit" class="text-red-
500">Supprimer</button>
            </form>
        </div>
    </div>
    @endforeach
</div>
@endsection
```

✅ Page de création d'une tâche

Créer le fichier :

resources/views/tasks/create.blade.php :

Dans **resources/views/tasks/create.blade.php** :

```
@extends('layouts.app')

@section('content')
<div class="max-w-4xl mx-auto p-4">
    <h1 class="text-2xl font-bold mb-4">Créer une tâche</h1>

    <form action="{{ route('tasks.store') }}" method="POST">
        @csrf
```

```
        <input type="text" name="title" class="w-full border p-2"
placeholder="Titre de la tâche" required>
        <textarea name="description" class="w-full border p-2 mt-2"
placeholder="Description"></textarea>
        <input type="date" name="due_date" class="w-full border p-2 mt-2">
        <button type="submit" class="bg-blue-500 text-white px-4 py-2
rounded mt-4">Créer</button>
    </form>
</div>
@endsection
```

7️⃣ Ajouter un menu de navigation pour les tâches

Comment ajouter un menu simple dans la barre de navigation en haut de ta page, ce qui permet à l'utilisateur d'accéder aux différentes pages de ton application (comme la liste des tâches, la création, etc.).

Modifier `resources/views/layouts/app.blade.php`

Dans ce fichier, tu peux ajouter une **barre de navigation** qui permet à l'utilisateur d'accéder à la liste des tâches et à la création de nouvelles tâches. On va également y inclure les liens nécessaires pour l'authentification (connexion/déconnexion).

Voici un exemple simple de code pour la barre de navigation avec Tailwind CSS :

Exemple de code pour la barre de navigation :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do List</title>
    @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
<body class="bg-gray-100">

    <!-- Barre de navigation -->
    <nav class="bg-blue-600 p-4">
        <div class="max-w-7xl mx-auto flex items-center justify-between">
            <a href="{{ route('tasks.index') }}" class="text-white text-xl
font-semibold">Ma To-Do List</a>

            <div class="space-x-4">
                <!-- Lien vers la liste des tâches -->
                <a href="{{ route('tasks.index') }}" class="text-white">Mes
Tâches</a>

                <!-- Lien vers la création de tâches -->
                <a href="{{ route('tasks.create') }}" class="text-
white">Nouvelle tâche</a>

                <!-- Si l'utilisateur est authentifié -->
                @auth
                    <form action="{{ route('logout') }}" method="POST"
class="inline">
                        @csrf
                        <button type="submit" class="text-white">Se
déconnecter</button>
                    </form>
                @else
                    <a href="{{ route('login') }}" class="text-white">Se
connecter</a>
                    <a href="{{ route('register') }}" class="text-
white">S'inscrire</a>
                @endauth
            </div>
        </div>
```

```
    </nav>

    <!-- Contenu principal -->
    <div class="py-6">
        @yield('content')
    </div>

</body>
</html>
```

Explication :

- **Barre de navigation** : Un menu avec des liens pour naviguer vers la page des tâches (`tasks.index`), la page de création d'une tâche (`tasks.create`), et des liens de connexion/déconnexion pour l'authentification.
- **`@auth`** : Utilisé pour afficher les options de déconnexion si l'utilisateur est authentifié. Sinon, les options de connexion et d'inscription seront affichées.

8️⃣ Tester l'application

- Lance le serveur Laravel :

```
php artisan serve
```

- Accède à `http://127.0.0.1:8000`
- Teste l'inscription et la gestion des tâches 🚀

9️⃣ Conclusion

✅ **Laravel Breeze** gère l’authentification
✅ **CRUD complet** pour gérer les tâches
✅ **Interface simple avec Tailwind CSS**

💡 **Prochaines améliorations :**

- Ajouter une **barre de recherche**
- Trier les tâches par priorité
- Ajouter des **notifications d'échéance**