
Les Commandes Artisan Utilisées

1. Génération des fichiers de langue

```
php artisan lang:publish
```

Explication

- Cette commande permet de publier les fichiers de traduction dans le dossier `lang/`.
- Elle crée un dossier contenant les fichiers de langues (`messages.php`) pour stocker les traductions.

 `lang/en/messages.php`

```
return [  
    'title' => 'This is an English title.',  
];
```

Explication

- Ce fichier contient un tableau associatif où chaque clé (`title`) correspond à une phrase traduisible.
- Laravel utilisera ces fichiers pour afficher le texte dans la langue choisie.

 **Même structure pour `lang/fr/messages.php` et `lang/ar/messages.php`.**

Les Fichiers de Traduction et Leur Rôle

2. Création du Contrôleur `LocaleController`

```
php artisan make:controller LocaleController
```

Explication

- Cette commande crée un contrôleur dans `app/Http/Controllers/LocaleController.php`.

 `LocaleController.php`

```
public function setLocale($lang)  
{  
    if (in_array($lang, ['en', 'fr', 'ar'])) {  
        App::setLocale($lang);  
    }  
}
```

```
        Session::put('locale', $lang);
    }
    return back();
}
```

🔍 Explication

- Cette fonction **change la langue de l'application** en fonction du paramètre `$lang`.
- Elle vérifie si `$lang` est bien une langue supportée (`en`, `fr`, `ar`).
- `App::setLocale($lang)` met à jour la langue de Laravel.
- `Session::put('locale', $lang)` enregistre la langue choisie pour la session utilisateur.
- `return back();` redirige l'utilisateur vers la page précédente.

🔒 Les Contrôleurs et Middlewares

🔗 3. Création du Middleware `LocalizationMiddleware`

```
php artisan make:middleware LocalizationMiddleware
```

🔍 Explication

- Cette commande crée un middleware (`app/Http/Middleware/LocalizationMiddleware.php`).
- Un **middleware** est une couche intermédiaire qui **modifie ou contrôle** les requêtes HTTP avant qu'elles n'atteignent les routes ou les contrôleurs.

💻 `LocalizationMiddleware.php`

```
public function handle(Request $request, Closure $next): Response
{
    $locale = Session::get('locale') ?? 'en'; // Langue par défaut : anglais
    Session::put('locale', $locale);
    App::setLocale($locale);
    return $next($request);
}
```

🔍 Explication

- Ce middleware **intercepte chaque requête** et applique la langue enregistrée en session.
- Par défaut, il utilise l'anglais si aucune langue n'est définie.
- `return $next($request);` permet de poursuivre l'exécution normale de l'application.

👉 **Ce middleware garantit que la langue reste la même après un changement de page !**

L'Intégration dans les Vues et Routes

4. Ajout du Middleware dans `bootstrap/app.php`

 `bootstrap/app.php`

```
$middleware->web(append: [  
    App\Http\Middleware\LocalizationMiddleware::class,  
]);
```

Explication

- Cette ligne **ajoute le middleware** `LocalizationMiddleware` dans le groupe `web`.
- Laravel va exécuter ce middleware sur chaque requête web.

 **Chaque visiteur verra l'interface dans la langue stockée en session !**

Création de la Vue `locale.blade.php`

5. Création de la Vue `locale.blade.php`

 `resources/views/tp8/locale.blade.php`

```
<html lang="{{ app()->getLocale() }}">
```

Explication

- `app()->getLocale()` récupère la langue actuelle de l'application et la définit dans la balise `<html>`.


```
<h1>@lang('messages.title')</h1>
```

Explication

- `@lang('messages.title')` affiche la traduction du fichier `messages.php` selon la langue active.

 **Si la langue est `fr`, il affichera :** *Ceci est un titre en français.*

Ajout des Boutons de Changement de Langue

 Ajout des boutons de changement de langue

```
<a href="/locale/en" class="btn btn-primary">GB English</a>
<a href="/locale/fr" class="btn btn-secondary">FR Français</a>
<a href="/locale/ar" class="btn btn-success">SA العربية</a>
```

Explication

- Chaque bouton **appelle la route** `/locale/{lang}` pour changer la langue.
- Si l'utilisateur clique sur **Français**, la requête `/locale/fr` est envoyée à `LocaleController`.

Définition des Routes dans `web.php`

6. Définition des Routes

 `routes/web.php`

```
Route::get('/locale', function () {
    return view('tp8.locale');
});

Route::get('/locale/{lang}', [LocaleController::class, 'setLocale']);
```

Explication

- 1 La route `/locale` affiche la page `locale.blade.php`.
- 2 La route `/locale/{lang}` est associée à `LocaleController::setLocale()` pour changer la langue et rediriger l'utilisateur.

Résumé du Fonctionnement Global

7. Résumé du Fonctionnement Global

- 1 L'utilisateur arrive sur `/locale` et voit la page avec un titre traduit.
- 2 Il clique sur un bouton de langue (ex : FR Français → `/locale/fr`).
- 3 `LocaleController::setLocale()` **change la langue** et l'enregistre en session.
- 4 Le **middleware** `LocalizationMiddleware` s'assure que la langue reste active.
- 5 Laravel charge automatiquement les **bons fichiers de traduction** (`lang/fr/messages.php`, `lang/en/messages.php`...).

Améliorations Possibles

8. Améliorations Possibles

- ☒ Ajouter une **liste déroulante** pour choisir la langue au lieu des boutons.
 - ☒ Stocker la langue **dans la base de données** pour chaque utilisateur connecté.
 - ☒ Charger les traductions **dans JavaScript** pour les rendre dynamiques sans recharger la page.
-

Conclusion

9. Conclusion

 **Ton TP8 Laravel implémente un système multilingue complet !**

Tu as appris à :

- ☒ **Créer et utiliser des fichiers de traduction**
- ☒ **Changer la langue dynamiquement avec un contrôleur**
- ☒ **Utiliser un middleware pour gérer la session de langue**
- ☒ **Intégrer le tout dans une vue avec Blade**

Si tu as des questions ou veux approfondir un point, dis-moi !  