WANDERBOOK

Contexte : Wanderbook est un carnet de voyage numérique. Contrairement à un simple blog, il utilise une carte du monde interactive permettant de visualiser ses voyages et de consulter des fiches personnalisées par pays.

1. Initialisation de l'environnement

On commence par créer la structure de base du projet avec Laravel.
composer create-project laravel/laravel wanderbook

2. Installation des outils "Front-end"

On installe les outils qui gèrent l'aspect visuel et interactif.
npm install @inertiajs/vue3 vue @vitejs/plugin-vue leaflet

3. Connexion entre le moteur et l'interface

On configure Inertia, qui sert de pont entre le serveur (Laravel) et le visuel (Vue.js).
Modification du fichier bootstrap/app.php pour inclure le "Middleware" de gestion des requêtes grâce à composer require inertiajs/inertia-laravel
php artisan inertia:middleware

```php
return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        // C'est ici qu'on ajoute le middleware Inertia
        $middleware->web(append: [
            \App\Http\Middleware\HandleInertiaRequests::class,
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();
```

4. Configuration de la base de données SQLite

On choisit un format de base de données ultra-léger.
New-Item -ItemType File database\database.sqlite

COUR Cyrille – cyrillecour11@gmail.com – Aristée 2025/2027

5. Déploiement du code métier

On créer les fichiers (Modèles, Contrôleurs, Routes) dans l'architecture Laravel. Ces fichiers sont générés rapidement car l'idée de l'application était claire dans ma tête depuis un moment, et que Laravel est un framework que je maitrise.

```php
Route::get('/', [CountryController::class, 'index'])->name('map');
Route::get('/stats', [StatsController::class, 'index'])->name('stats');

Route::get('/countries/{country}', [CountryController::class, 'show'])->name('countries.show');
Route::patch('/countries/{country}/status', [CountryController::class, 'updateStatus'])->name('countries.status');
Route::patch('/countries/{country}/ratings', [CountryController::class, 'updateRatings'])->name('countries.ratings');

Route::post('/countries/{country}/trips', [TripController::class, 'store'])->name('trips.store');
Route::patch('/trips/{trip}', [TripController::class, 'update'])->name('trips.update');
Route::delete('/trips/{trip}', [TripController::class, 'destroy'])->name('trips.destroy');
```

```php
class Country extends Model
{
    protected $fillable = [
        'iso_code', 'name', 'flag_emoji', 'status', 'summary',
        'quality_of_life', 'living_environment',
    ];

    public const STATUS_COLORS = [
        null        => '#B4B2A9',
        'disliked'  => '#8B5E3C',
        'liked'     => '#1D9E75',
        'loved'     => '#378ADD',
        'favorite'  => '#E24B4A',
    ];

    public const STATUS_LABELS = [
        null        => 'Pas fait',
        'disliked'  => 'Pas aimé',
        'liked'     => 'Sympa',
        'loved'     => 'Bien',
        'favorite'  => 'Adoré',
    ];
```

Extrait du code des Pays qui sont essentiel dans la logique de notation de mon app.

6. Création de la structure (Migrations)

On demande à Laravel de créer les tables dans le fichier SQLite.
php artisan migrate

7. Importation des données géographiques

On récupère les contours des pays du monde
world.geojson (DL sur Github) placé dans le dossier public/.

8. Configuration du compilateur (Vite)

On paramètre l'outil qui prépare le code pour le navigateur.

```
import { createApp, h } from 'vue'
import { createInertiaApp } from '@inertiajs/vue3'
import { resolvePageComponent } from 'laravel-vite-plugin/inertia-helpers'
import 'leaflet/dist/leaflet.css'

createInertiaApp({
    title: title => `${title} – Wanderbook`,
    resolve: name => resolvePageComponent(
        `./Pages/${name}.vue`,
        import.meta.glob('./Pages/**/*.vue')
    ),
    setup({ el, App, props, plugin }) {
        createApp({ render: () => h(App, props) })
            .use(plugin)
            .mount(el)
    },
})
```

App.JS

```
export default defineConfig({
    plugins: [
        laravel({
            input: ["resources/js/app.js"],
            refresh: true,
        }),
        vue({
            template: {
                transformAssetUrls: {
                    base: null,
                    includeAbsolute: false,
                },
            },
        }),
    ],
    resolve: {
        alias: {
            "@": "/resources/js",
        },
    },
});
```

Vite.config.js

9. Peuplement initial (Seeding)

On remplit la base de données avec des informations de départ.
php artisan db:seed --class=CountrySeeder

10. Lancement de l'application

L'application est prête à être utilisée.
php artisan serve pour lancer le server puis npm run dev pour l'interface.

A ce stade peu de code écrit de ma main la majorité à été faite par Laravel. Mais ici on se heurte à un problème, le clic sur un pays d'une carte ne fais pas apparaitre sa fiche.

11. Correction de bug

Pour cela j'ai relu mon code et vue que Laravel lisé l'ID et non le Iso_code j'ai donc changé cela

```php
/**
* Utilise l'iso_code au lieu de l'ID pour les routes Laravel
*/
public function getRouteKeyName(): string
{
    return 'iso_code';
}
```

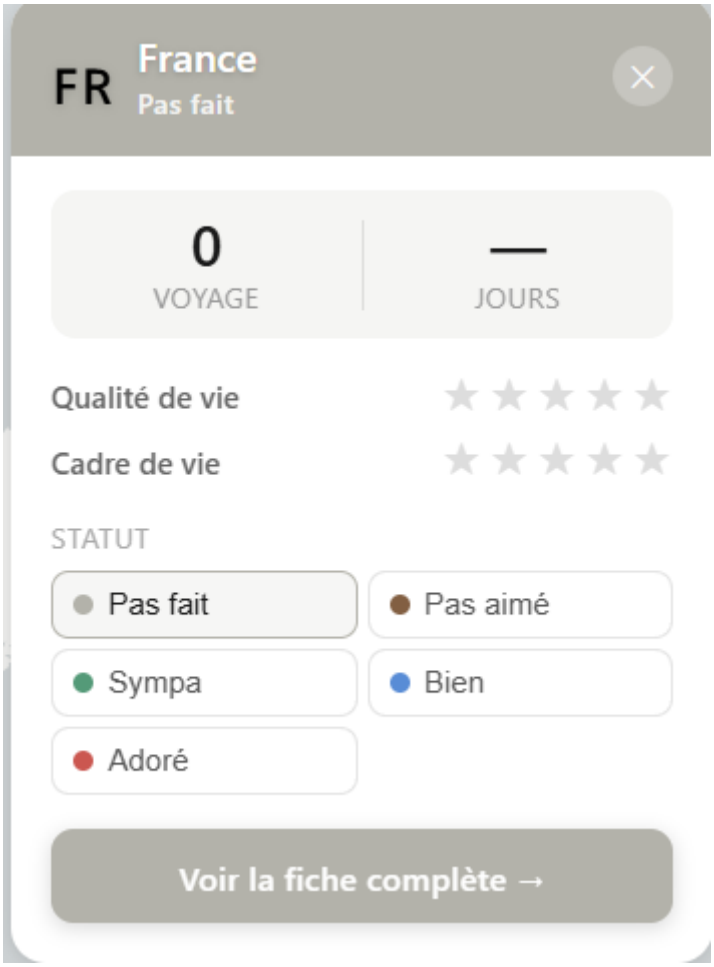
Cependant la France et la Norvège ne fonctionnait pas … Le fichier GeoJSON mondial utilisait par erreur le code -99 au lieu de FR pour ce pays.
Solution ? Ajout d'une condition logique pour intercepter ce code spécifique et le corriger "à la volée" lors du rendu de la carte.

```
function onEachFeature(feature, layer) {
    layer.on({
        mouseover(e) {
            e.target.setStyle({ fillOpacity: 1, weight: 1.5 });
        },
        mouseout(e) {
            geoLayer.resetStyle(e.target);
        },
        click() {
            // On récupère le code du GeoJSON
            let iso = feature.properties.ISO_A2;

            // CORRECTIF : Si le code est -99 ou absent, on regarde le nom
            if (iso === "-99" || !iso) {
                const name =
                    feature.properties.NAME || feature.properties.ADMIN;
                if (name === "France") iso = "FR";
                if (name === "Norway") iso = "NO";
            }

            // On cherche dans notre map (en majuscules)
            const country = countryMap[iso?.toUpperCase()];

            if (country) {
                selected.value = country;
            } else {
                console.error(
                    "Toujours pas trouvé pour :",
                    iso,
                    feature.properties.NAME,
                );
            }
        },
    });
}
```

WANDERBOOK

Voilà un extrait de la carte

Si on clique sur un pays :



COUR Cyrille – cyrillecour11@gmail.com – Aristée 2025/2027

WANDERBOOK

Cet fiche n’est pas encore remplie mais lorsque ça sera fait les infos détail de la fiche seront affiché ici, le pays se rempliera de couleur sur la carte également.

Ensuite nous avons configuré le modèle Country pour qu'il soit le pivot entre la carte et les données.
Ajout d'une colonne status à la table countries.
Ajout de status dans le tableau $fillable pour autoriser l'écriture.
Configuration de getRouteKeyName() pour retourner 'iso_code', permettant à Laravel de trouver un pays via son code (ex: /countries/FR) plutôt que son ID numérique.

```php
public function getRouteKeyName(): string
{
    return 'iso_code';
}
```

La fonction de mise à jour a été simplifiée pour être ultra-fiable :

```php
<?php

use App\Http\Controllers\CountryController;
use App\Http\Controllers\TripController;
use App\Http\Controllers\StatsController;
use Illuminate\Support\Facades\Route;

Route::get('/', [CountryController::class, 'index'])->name('map');
Route::get('/stats', [StatsController::class, 'index'])->name('stats');

Route::get('/countries/{country}', [CountryController::class, 'show'])->name('countries.show');
Route::patch('/countries/{country:iso_code}/status', [CountryController::class, 'updateStatus'])->name('countries.status');
Route::patch('/countries/{country:iso_code}/ratings', [CountryController::class, 'updateRatings'])->name('countries.ratings');
Route::patch('/countries/{country:iso_code}/details', [CountryController::class, 'updateDetails'])->name('countries.details.update');
Route::post('/countries/{country:iso_code}/flag', [CountryController::class, 'uploadFlag'])->name('countries.flag.upload');

Route::post('/countries/{country:iso_code}/trips', [TripController::class, 'store'])->name('trips.store');
Route::patch('/trips/{trip}', [TripController::class, 'update'])->name('trips.update');
Route::delete('/trips/{trip}', [TripController::class, 'destroy'])->name('trips.destroy');
```

Le plus gros défi était de faire correspondre le fichier de la carte (GeoJSON) avec les données Laravel. Partout ou l’on cherche country.id on le remplace par. Iso_code

Pour que la carte change de couleur sans recharger la page :

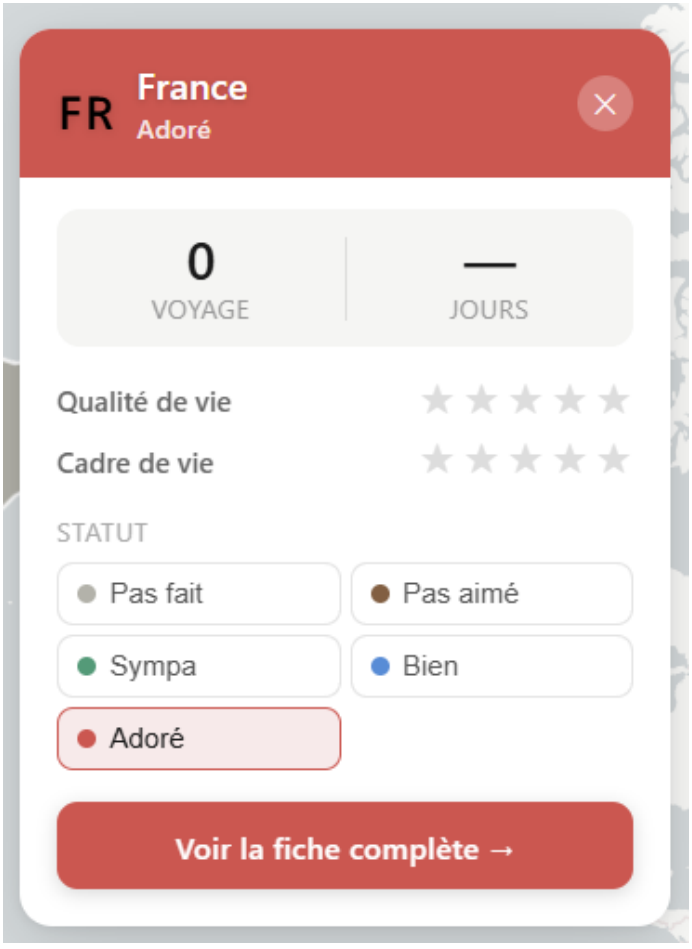
```javascript
// ── Leaflet helpers ──────────────────────────────────
function getColor(isoCode) {
    if (!isoCode) return "#B4B2A9";
    const code = String(isoCode).toUpperCase();
    const c = countryMap.value[code]; // .value car c'est un computed
    if (!c) return "#B4B2A9";
    return statuses.find((s) => s.value === c.status)?.color ?? "#B4B2A9";
}
```

COUR Cyrille – cyrillecour11@gmail.com – Aristée 2025/2027

WANDERBOOK

Voici l'application Wanderbook :

L'écran principal -> une carte du monde les pays en gris ne sont pas fait, ceux coloré oui.

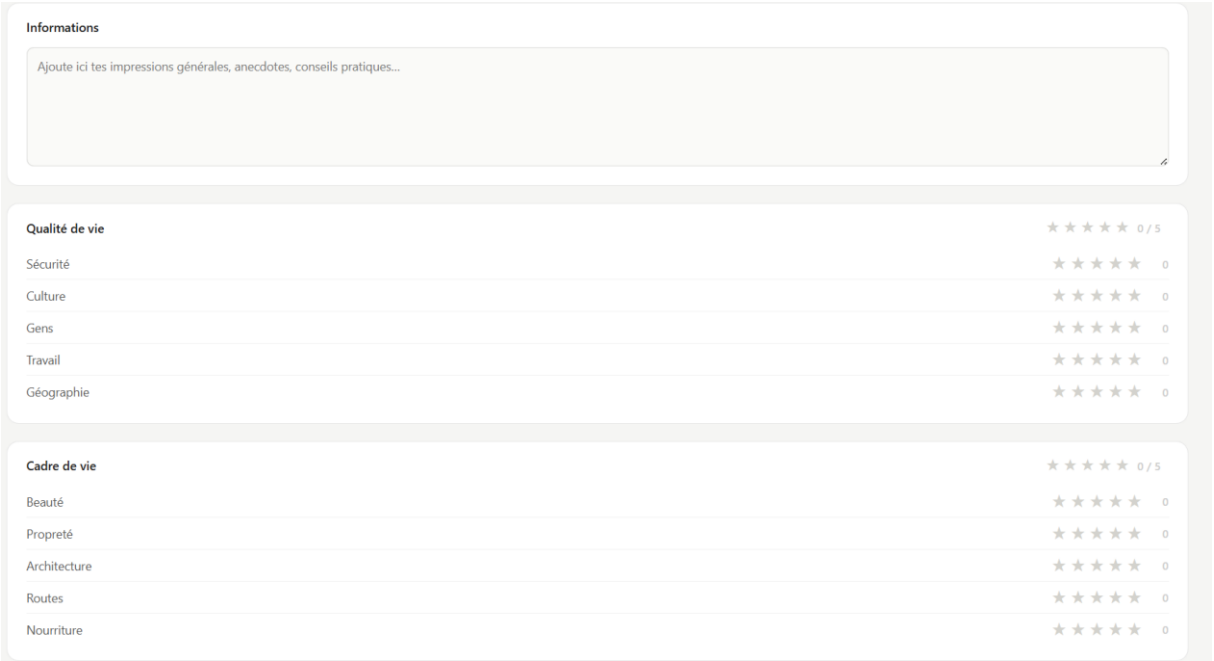


Lors ce qu'on clique sur un pays sa fiche résumé apparait.

COUR Cyrille – cyrillecour11@gmail.com – Aristée 2025/2027

Enfin l’écran de détail :

Le bandeau supérieur.



L’écran de notation.

Et voilà l’application est parfaitement fonctionnelle ! Cette application utilise Laravel, Vue, Leaflet, Vite js et d’autre technologie.