Projet personnel — En développement actif

Wanderbook

Carnet de voyage numérique interactif

Laravel — Inertia — Vue 3 — Leaflet

Auteur	Cyrille COUR
Type de projet	Personnel — application web, maintenue activement
Stack technique	Laravel / Inertia.js / Vue 3 / Leaflet / Vite / SQLite
Objectif	Cartographier ses voyages et noter chaque pays visité selon des critères personnalisés
Filiation	Suite logique de MyVAR (notation de lieux visités)

1. Contexte et objectif

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.

Filiation avec MyVAR : Wanderbook est la suite logique de MyVAR. Le principe de noter des lieux visités selon différents critères existait déjà avec MyVAR (PowerApps, limité aux villes du Var) ; Wanderbook reprend cette idée à l'échelle mondiale, avec une vraie stack de développement (Laravel/Vue) et une carte interactive plutôt qu'une simple liste.

Ce projet, contrairement à MyVAR ou Pi-hole, est toujours activement maintenu : les voyages et les notes par pays continuent d'y être renseignés au fil du temps.

2. Stack technique

Couche	Techno	Rôle
Backend	Laravel	Framework PHP — modèles, contrôleurs, routes, migrations
Pont serveur ↔ front	Inertia.js	Fait communiquer Laravel et Vue sans API REST séparée
Frontend	Vue 3	Composants d'interface (carte, fiches pays, formulaires)
Cartographie	Leaflet	Rendu de la carte du monde interactive
Build tool	Vite	Compilation et rechargement à chaud du frontend
Base de données	SQLite	Format de base de données ultra-léger, un seul fichier

3. Mise en place du projet

3.1 Initialisation de l'environnement

Création de la structure de base du projet avec Laravel :

```
composer create-project laravel/laravel wanderbook
```

3.2 Installation des outils front-end

Installation des outils qui gèrent l'aspect visuel et interactif :

```
npm install @inertiajs/vue3 vue @vitejs/plugin-vue leaflet
```

3.3 Connexion entre le moteur et l'interface

Configuration d'Inertia, qui sert de pont entre le serveur (Laravel) et le visuel (Vue.js) :

```
composer require inertiajs/inertia-laravel
php artisan inertia:middleware
```

Modification du fichier bootstrap/app.php pour inclure le middleware de gestion des requêtes :

```
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();
```

3.4 Configuration de la base de données SQLite

Choix d'un format de base de données ultra-léger :

```
New-Item -ItemType File database\database.sqlite
```

4. Déploiement du code métier

Création des fichiers (modèles, contrôleurs, routes) dans l'architecture Laravel. Ces fichiers ont été générés rapidement car l'idée de l'application était claire depuis un moment, et Laravel est un framework maîtrisé.

Routes principales (routes/web.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');
```

Modèle Country — pivot de la logique de notation

Extrait du code des pays, essentiel dans la logique de notation de l'application :

```
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',
    ];
```

4.1 Migrations et données géographiques

Création des tables dans le fichier SQLite :

```
php artisan migrate
```

Récupération des contours des pays du monde : le fichier world.geojson (téléchargé sur GitHub) est placé dans le dossier public/.

4.2 Configuration du compilateur (Vite)

Paramétrage de l'outil qui prépare le code pour le navigateur.

resources/js/app.js

```
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)
    },
})
```

vite.config.js

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

4.3 Peuplement initial et lancement

Remplissage de la base de données avec des informations de départ :

```
php artisan db:seed --class=CountrySeeder
```

Lancement de l'application :

```
php artisan serve
npm run dev
```

> *À ce stade, peu de code avait été écrit à la main — la majorité provenait de la génération Laravel standard.*

5. Correction de bug — correspondance des codes pays

Premier problème rencontré : le clic sur un pays de la carte ne faisait pas apparaître sa fiche.

En relisant le code, il est apparu que Laravel cherchait les pays par leur ID numérique et non par leur iso_code. Correction apportée au modèle Country :

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

Complication supplémentaire : la France et la Norvège ne fonctionnaient toujours pas. Le fichier GeoJSON mondial utilisait par erreur le code -99 au lieu de FR/NO pour ces deux 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,
        );
      }
    },
  });
}
```

Le plus gros défi était de faire correspondre le fichier de la carte (GeoJSON) avec les données Laravel : partout où l'on cherchait country.id, il a fallu le remplacer par country.iso_code — y compris dans les routes finales :

```
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');
```

Pour que la carte change de couleur sans recharger la page, une fonction dédiée détermine la couleur à appliquer à chaque pays selon son statut :

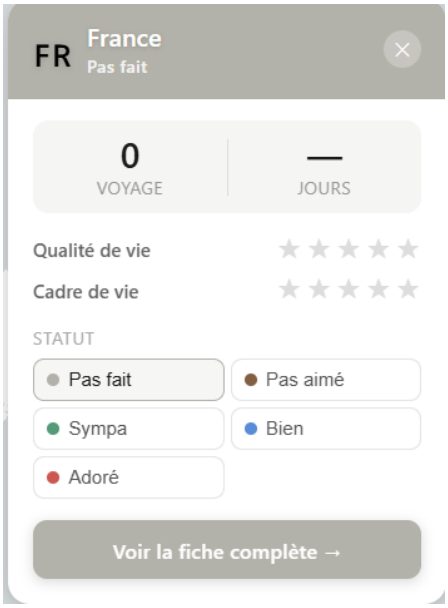
```
// — Leaflet helpers —
function getColor(isoCode) {
  if (!isoCode) return "#B4B2A9";
  const code = String(isoCode).toUpperCase();
  const c = countryMap.value[code]; // .value car c'est un computed
  if (!c) return "#B4B2A9";
  return statuses.find((s) => s.value === c.status)?.color ?? "#B4B2A9";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

6. L'application en images

Écran principal : une carte du monde où les pays en gris ne sont pas encore visités, ceux en couleur le sont déjà.

Carte du monde avant renseignement des voyages

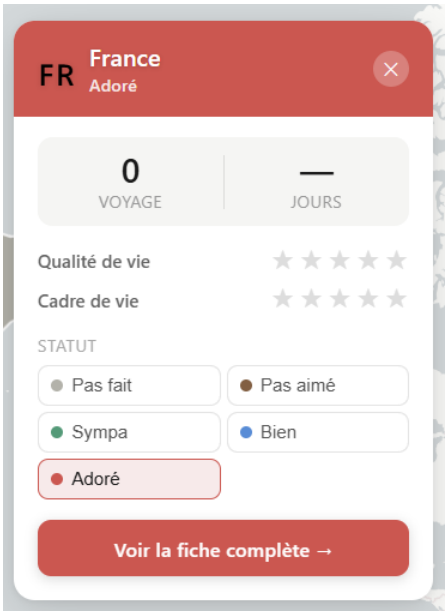
Lorsqu'on clique sur un pays, sa fiche résumé apparaît, avec le statut, la note et un accès à la fiche complète :



Fiche résumé d'un pays — statut « Pas fait »

Une fois le pays noté, la carte se colore selon le statut choisi — ici la France en rouge (« Adoré ») :

Carte du monde — la France apparaît en rouge (statut Adoré)



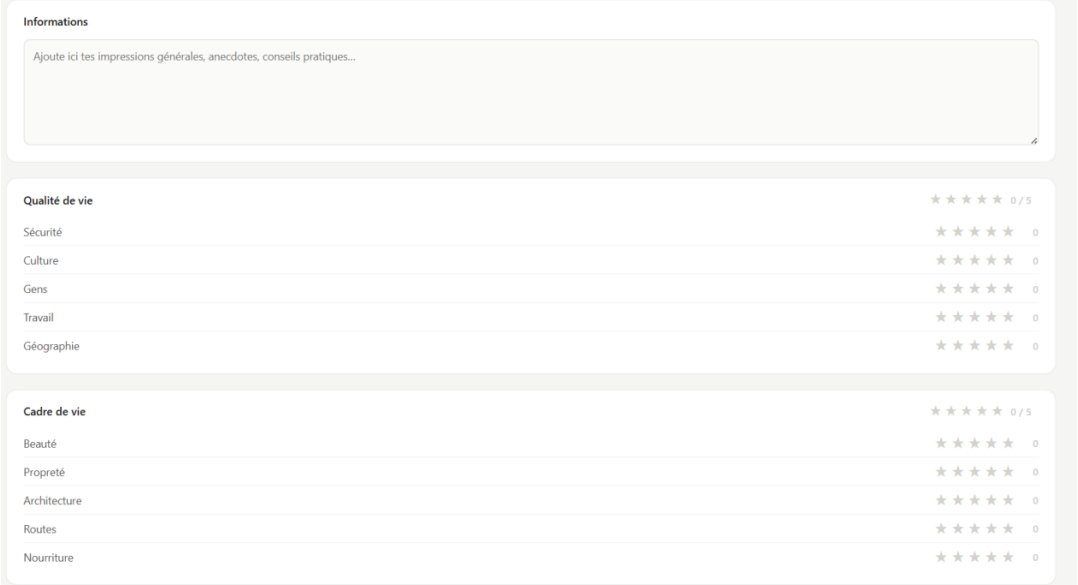
Fiche résumé — la France notée « Adoré »

6.1 Écran de détail

Le bandeau supérieur affiche le nom du pays, son drapeau, son statut, sa note globale et quelques compteurs (voyages, jours, villes) :

Bandeau supérieur de l'écran de détail — France

Plus bas, l'écran de notation permet d'évaluer le pays selon des critères détaillés, regroupés en deux catégories (Qualité de vie et Cadre de vie), ainsi qu'un champ libre pour les impressions générales :



Écran de notation détaillée par critères

7. Bilan

Wanderbook est aujourd'hui parfaitement fonctionnel et utilise Laravel, Vue, Leaflet, Vite et d'autres technologies pour offrir un carnet de voyage interactif complet. Contrairement à MyVAR ou Pi-hole, c'est un projet vivant : les voyages et les notes par pays continuent d'y être renseignés au fil du temps.

Prochaine étape naturelle : remplir progressivement les fiches détaillées pour chaque pays déjà visité, et enrichir les statistiques globales (page /stats) à mesure que les données s'accumulent. Création de la fonction voyages qui permet de renseigner le nombre de jours passé dans un pays.