Projet personnel

Documentation Technique de Réalisation

Livre de recettes personnel en forme de pokédex

Laravel 13 — Gourmandex / Projet perso

Candidat	Cyrille COUR
Établissement	Aristée
Stack technique	PHP 8.3 / Laravel 13 / MySQL / Tailwind CSS 4
Organisation	Union Patronale du Var (UPV)
Date	Été 2026

Sommaire

1. Présentation du projet

1.1 Concept

Gourmandex est un livre de recettes personnel conçu comme un pokédex : chaque plat obtient son numéro d'entrée, ses types (jusqu'à deux, jamais zéro ni trois), sa nature, sa difficulté en étoiles et sa vignette pixel art. L'application est rendue entièrement côté serveur avec Laravel, sans framework JavaScript et sans comptes utilisateurs — c'est un livre personnel, pas un site communautaire.

L'ensemble a été construit en 14 commits sur une seule journée, le 5 août 2026, depuis le squelette Laravel jusqu'à l'écran de statistiques.

1.2 Identité visuelle

Direction artistique pixel art sur fond beige crème (« parchemin »), portée par deux polices bitmap auto-hébergées : Press Start 2P pour les titres, numéros et libellés, et Pixelify Sans pour le texte courant. Les couleurs de marque sont déclarées une seule fois via la directive @theme de Tailwind CSS 4.

1.3 Partis pris de conception

Décision	Pourquoi
Pas de comptes utilisateurs	C'est un livre personnel, pas un site communautaire. Tables users / sessions / cache / jobs supprimées, drivers en mode fichier.
Les types en enum, pas en base	Les 7 types relèvent des règles du Gourmandex, pas de données saisies. L'ensemble est fermé.
Un ou deux types, jamais zéro ni trois	Garanti par le schéma lui-même : primary_type obligatoire, secondary_type facultatif, pas de troisième colonne.
La durée totale n'est pas stockée	C'est la somme des durées d'étapes, calculée à la volée. Une seule source de vérité.
Les convives ne sont pas une colonne	Même logique : les quantités sont écrites pour deux personnes et multipliées à l'affichage.
Vignettes en base, pas sur le disque	Sauvegarde et suppression atomiques avec la recette, servies par une route durcie.
Pas de framework JavaScript	Tout est rendu côté serveur. Le seul script est le JS natif du formulaire ; filtres et réglage des convives marchent sans une ligne de script.
Les statistiques comptées en mémoire	Le livre tient en une poignée de recettes ; une boucle est plus simple qu'une requête tordue sur type principal / secondaire.
Le seeder est volontairement vide	Aucune donnée d'exemple : le Gourmandex ne se remplit qu'avec de vraies recettes.
Vignettes fournies à la main	Rien n'est généré : chaque image est du pixel art déposé depuis le formulaire.

2. Stack technique et environnement

2.1 Langages

Langage	Rôle
PHP 8.3+	Toute la logique serveur — enums, modèles, contrôleurs
Blade	Les templates, en composants et partials
CSS (Tailwind 4, CSS-first)	Le design system, déclaré en @theme
JavaScript natif (ES modules)	Uniquement le formulaire de recette. Aucun framework.
SQL (migrations Eloquent)	Le schéma
Markdown	README.md, Recettes.md, la documentation

2.2 Back-end

Techno	Version	Rôle
Laravel	^13.8	Routing, contrôleurs, Blade, ORM Eloquent, migrations, validation
MySQL	—	Base de données
Eloquent ORM	inclus	Modèles, relations, casts, route model binding par slug
Laravel Tinker	^3.0	REPL en ligne de commande

2.3 Front-end

Techno	Version	Rôle
Tailwind CSS	^4.0	Design system utilitaire, configuré en CSS-first via @theme
Vite	^8.0	Bundler et serveur de développement avec HMR
laravel-vite-plugin	^3.1	Pont Vite ↔ Laravel + auto-hébergement des polices
Bunny Fonts	—	Source des polices, téléchargées au build : aucune requête tierce à l'exécution

Polices : Press Start 2P (titres, numéros, libellés) et Pixelify Sans (texte courant).

2.4 Qualité et outillage

Techno	Version	Rôle
PHPUnit	^12.5	Tests de fonctionnalité — filtres, formulaire, convives, stats
Laravel Pint	^1.27	Formatage PHP (preset Laravel)
Laravel Pail	^1.2	Lecture des logs en direct
Collision	^8.6	Rapport d'erreurs lisible en CLI
Faker	^1.23	Données de test
Mockery	^1.6	Doublures de test
concurrently	^9.0	Lance serveur + queue + logs + Vite d'un seul composer dev

2.5 Environnement

Outil	Rôle
Laragon	Environnement local Windows (Apache, MySQL, PHP) — sert http://gourmandex.test
Composer	Dépendances PHP et scripts (setup, dev, test)
npm / Node ≥ 20	Dépendances front et build
Git	Versionnage

3. Construction du projet — chronologie

3.1 Étape 0 — Le squelette

Point de départ : projet Laravel 13 nu, avec sa table users, ses tables de cache et de jobs, sa page welcome.blade.php et un README vide. Rien à illustrer ici : c'est la base fournie par composer create-project.

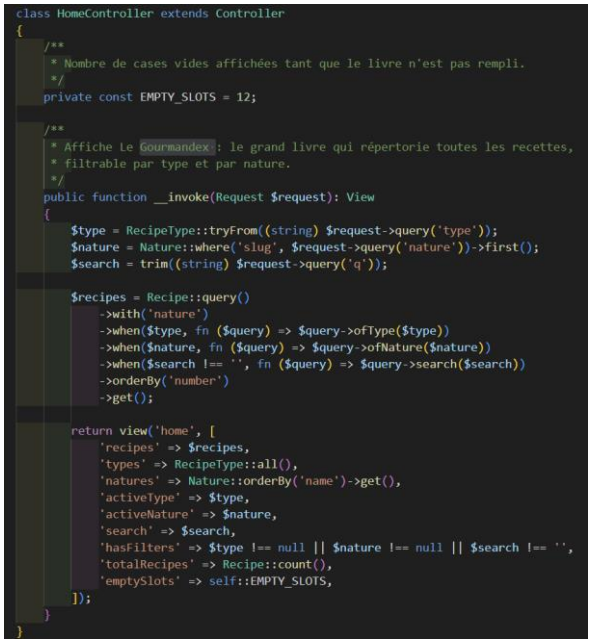
3.2 Étape 1 — La page d'accueil

Première vraie page : « Le Gourmandex », le grand livre où toutes les recettes s'affichent côte à côte dans des petits cadres. Naissance de la direction artistique pixel art / beige crème dans app.css, avec les polices bitmap branchées via Vite et Bunny Fonts.

```blade
@props(['recipe'])

{{-- Une entrée remplie du Gourmandex : vignette pixel art, numéro, nom et types. --}}
<a
    href="{{ route('recipes.show', $recipe) }}"
    {{ $attributes->class('pixel-frame group block bg-parchment-50 transition hover:-translate-y-0.5 hover:bg-parchment-200') }}
>
    <div class="relative border-b-4 border-ink-900 bg-parchment-200">
        @if ($recipe->hasImage())
            <img
                src="{{ route('recipes.image', $recipe) }}"
                alt="Vignette de la recette {{ $recipe->title }}"
                width="256"      You, il y a 20 heures • ajout des types natures et filtres pour les rec…
                height="256"
                class="aspect-square w-full object-cover"
            >
        @else
            {{-- Vignette pas encore fournie. --}}
            <div class="flex aspect-square items-center justify-center" role="img" aria-label="Vignette à venir">
                <span class="font-pixel text-4xl text-ink-300">?</span>
            </div>
        @endif

        <span class="absolute top-0 left-0 border-r-4 border-b-4 border-ink-900 bg-parchment-50 px-2 py-1 font-pixel text-[10px]
            N°{{ str_pad($recipe->number, 3, '0', STR_PAD_LEFT) }}
        </span>
    </div>
```

3.3 Étape 2 — Restructuration

Grand nettoyage juste après le premier jet, jugé trop chargé : 639 lignes retirées pour 380 ajoutées. HomeController allégé (106 → environ 30 lignes), home.blade.php dégraissé de 289 lignes, extraction du composant empty-slot et refonte des jetons de design dans app.css. Création du RecipeController et de la fiche recipes/show.blade.php.



```php
class HomeController extends Controller
{
    /**
     * Nombre de cases vides affichées tant que le livre n'est pas rempli.
     */
    private const EMPTY_SLOTS = 12;

    /**
     * Affiche le Gourmandex : le grand livre qui répertorie toutes les recettes,
     * filtrable par type et par nature.
     */
    public function __invoke(Request $request): View
    {
        $type = RecipeType::tryFrom((string) $request->query('type'));
        $nature = Nature::where('slug', $request->query('nature'))->first();
        $search = trim((string) $request->query('q'));

        $recipes = Recipe::query()
            ->with('nature')
            ->when($type, fn ($query) => $query->ofType($type))
            ->when($nature, fn ($query) => $query->ofNature($nature))
            ->when($search !== '', fn ($query) => $query->search($search))
            ->orderBy('number')
            ->get();

        return view('home', [
            'recipes' => $recipes,
            'types' => RecipeType::all(),
            'natures' => Nature::orderBy('name')->get(),
            'activeType' => $type,
            'activeNature' => $nature,
            'search' => $search,
            'hasFilters' => $type !== null || $nature !== null || $search !== '',
            'totalRecipes' => Recipe::count(),
            'emptySlots' => self::EMPTY_SLOTS,
        ]);
    }
}
```

```
@theme {    Unknown at rule @theme
    /* Pixel art : titres en 8 bits, texte courant en pixel lisible */
    --font-sans: 'Pixelify Sans', ui-sans-serif, system-ui, sans-serif;
    --font-pixel: 'Press Start 2P', ui-monospace, monospace;

    /* Parchemin — le beige crème du Gourmandex */
    --color-parchment-50: #fdf9f0;
    --color-parchment-100: #f8f0dd;
    --color-parchment-200: #f0e3c6;
    --color-parchment-300: #e4d2aa;
    --color-parchment-400: #d3bd8c;

    /* Encre — le brun des traits et du texte */
    --color-ink-900: #2d2317;
    --color-ink-700: #4b3b26;
    --color-ink-500: #705c40;
    --color-ink-300: #9c8a6c;

    /* Accents */
    --color-berry-400: #d06853;
    --color-berry-500: #bd4f39;
    --color-berry-600: #9d3c29;
    --color-leaf-500: #6f8f4a;
    --color-leaf-600: #567231;
    --color-sun-400: #e0a83c;

    /* Types du Gourmandex — teintes assourdies pour tenir sur le parchemin.
       Toutes portent du texte clair (parchment-50). */
    --color-type-feu: #cf6b3a;
    --color-type-eau: #4a7fa5;
    --color-type-fee: #b8608f;
    --color-type-glace: #5d93a8;
    --color-type-plante: #6f8f4a;
    --color-type-combat: #8f4b39;
    --color-type-normal: #8b7a5e;
}
```

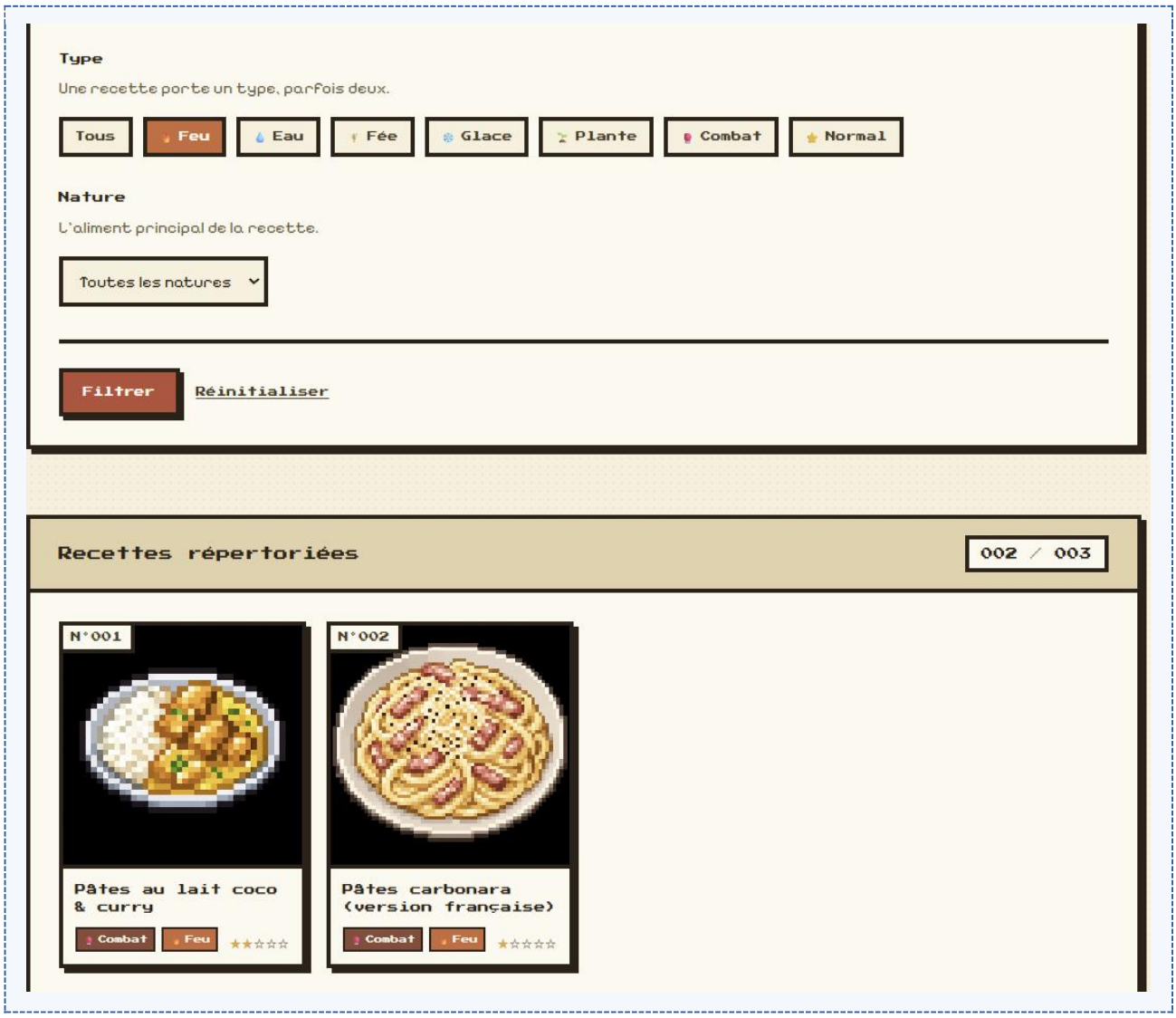
3.4 Étape 3 — Types, natures et filtres

Le cœur de la mécanique pokédex.

- App\Enums\RecipeType : les 7 types figés en enum (Feu, Eau, Fée, Glace, Plante, Combat, Normal), avec libellé, emoji et couleur — volontairement pas en base, ce sont les règles du livre.
- Modèle de données propre : migrations natures et recipes, modèles Recipe et Nature.
- Suppression de toute l'authentification : plus de User, plus de tables users, cache ni jobs.
- Filtres combinables par type et par nature, lisibles dans l'URL, en simple formulaire GET.
- Composant type-badge : les pastilles de type sous chaque nom.
- Premiers tests : GourmandexTest (135 lignes).

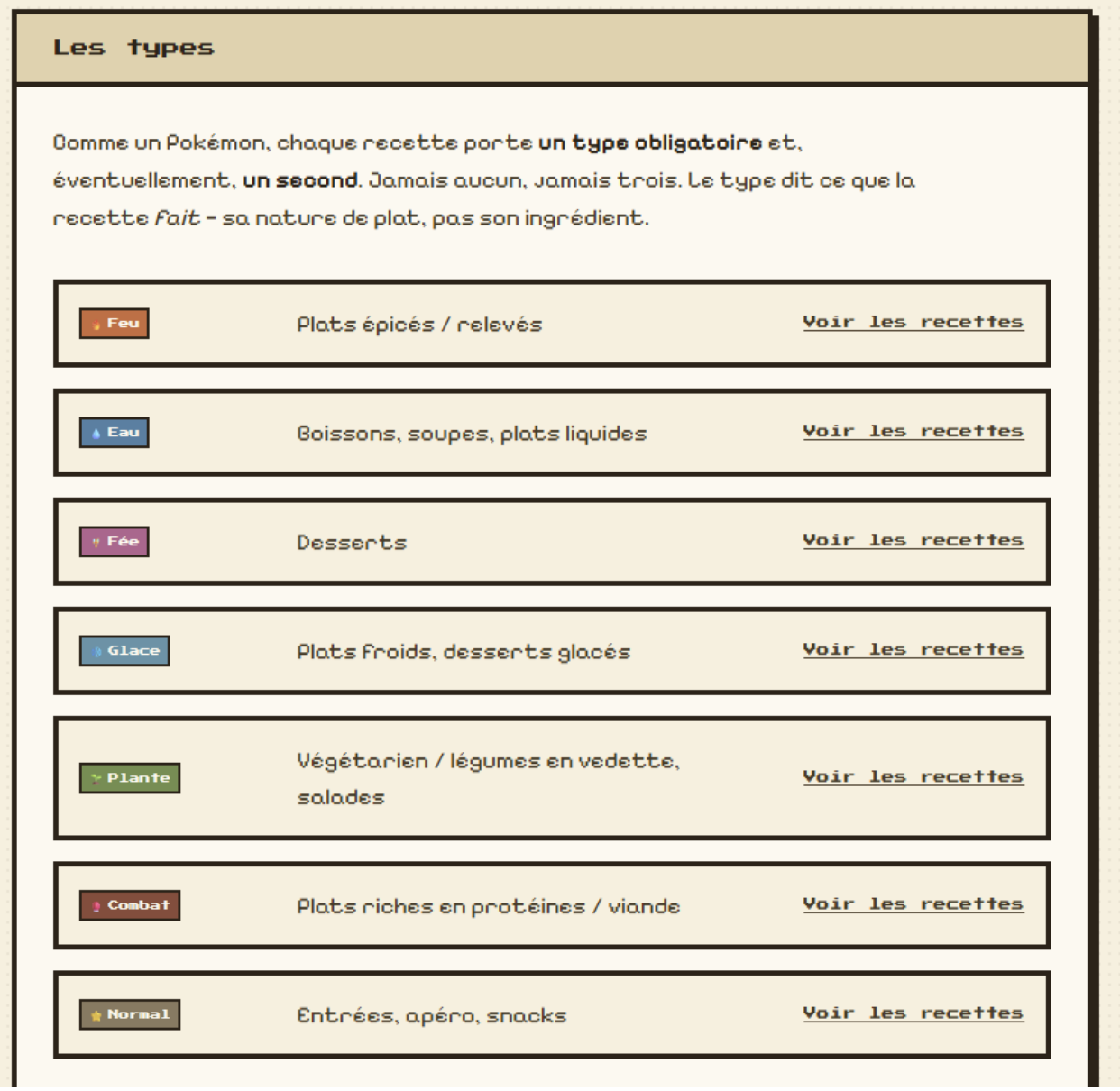
```php
enum RecipeType: string
{
    case Feu = 'feu';
    case Eau = 'eau';
    case Fee = 'fee';
    case Glace = 'glace';
    case Plante = 'plante';
    case Combat = 'combat';
    case Normal = 'normal';

    public function label(): string
    {
        return match ($this) {
            self::Feu => 'Feu',
            self::Eau => 'Eau',
            self::Fee => 'Fée',
            self::Glace => 'Glace',
            self::Plante => 'Plante',
            self::Combat => 'Combat',
            self::Normal => 'Normal',
        };
    }

    public function emoji(): string
    {
        return match ($this) {
            self::Feu => '🔥',
            self::Eau => '💧',
            self::Fee => '🧚',
            self::Glace => '❄️',
            self::Plante => '🌱',
            self::Combat => '🥊',
            self::Normal => '⭐',
        };
    }
```

```blade
<div class="space-y-8 p-5 sm:p-7">
    {{-- Type --}}
    <fieldset>
        <legend class="font-pixel text-[11px] text-ink-900">Type</legend>
        <p class="mt-2 text-ink-500">Une recette porte un type, parfois deux.</p>

        <div class="mt-4 flex flex-wrap gap-2.5">
            <label class="cursor-pointer">
                <input type="radio" name="type" value="" class="peer sr-only" @checked(! $activeType)>
                <span class="inline-block border-4 border-ink-900 bg-parchment-100 px-3 py-2 font-pixel text-[10px] text-ink-
                    Tous
                </span>
            </label>

            @foreach ($types as $type)
                <label class="cursor-pointer" title="{{ $type->description() }}">
                    <input type="radio" name="type" value="{{ $type->value }}" class="peer sr-only" @checked($activeType === $type)>
                    <span class="inline-flex items-center gap-1.5 border-4 border-ink-900 bg-parchment-100 px-3 py-2 font-pixel t
                        <span aria-hidden="true">{{ $type->emoji() }}</span>
                        {{ $type->label() }}
                    </span>
                </label>
            @endforeach
        </div>
    </fieldset>
```



3.5 Étape 4 — Icône et page Infos

Favicon multi-tailles (16 à 256 px) rangé dans public/icons/, plus l'icône pixel art de l'interface. Nouvelle page /infos détaillant les 7 types, ce qu'est une nature et comment se lit une case du livre, via InfoController.



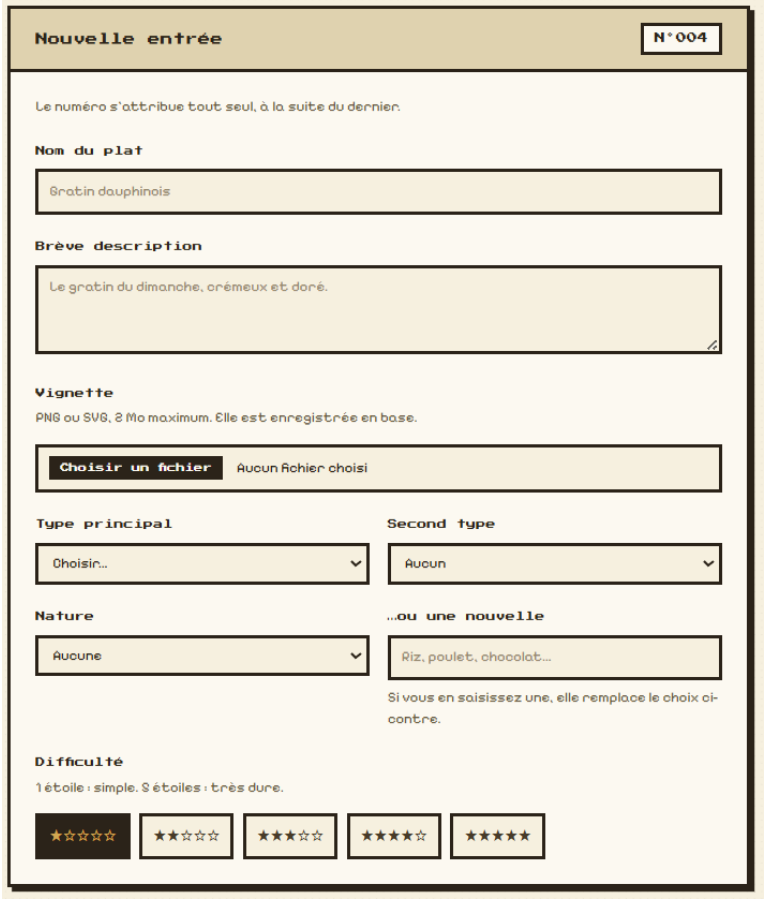
3.6 Étape 5 — La logique complète et la fiche recette

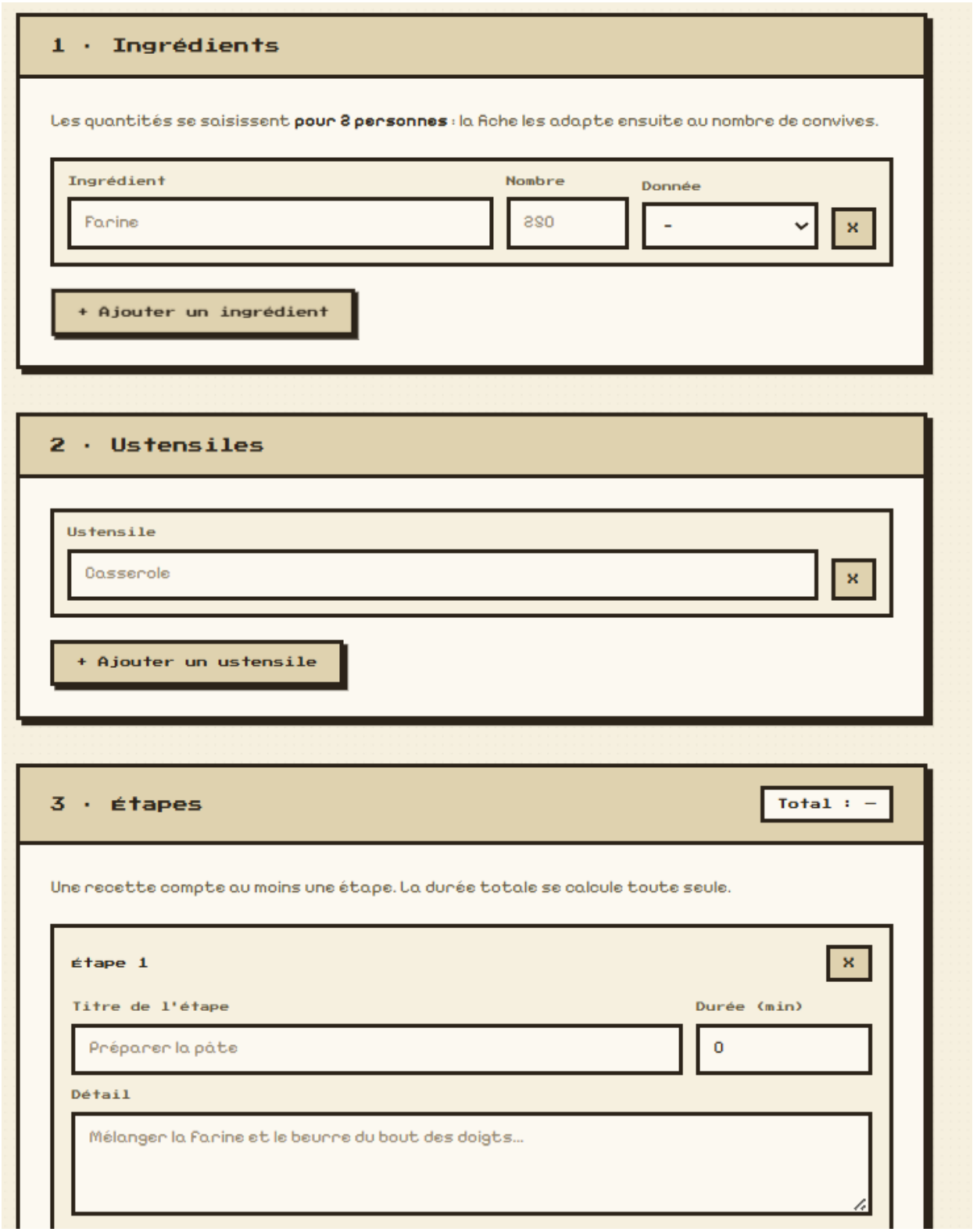
Le plus gros commit du projet : 1 698 lignes ajoutées sur 26 fichiers. C'est ici que l'application devient réellement utilisable.

- App\Enums\Unit : les unités du menu déroulant (pièce(s), g, cl, c. à soupe, gousse(s)…).
- Trois tables filles ingredients, utensils, steps, chacune en cascadeOnDelete avec une position d'affichage.
- CRUD complet (RecipeController + RecipeRequest pour la validation).
- Formulaire à lignes répétables (recipes/form.blade.php + composants form/ingredient-row, form/utensil-row, form/step-row).
- JavaScript natif dans app.js : ajout/suppression de lignes, renumérotation des étapes, durée totale calculée en direct pendant la saisie.
- App\Support\Duration : le formatage « 45 min », « 2 h 30 ».
- Vignette stockée en base (image_data + image_mime), servie par sa propre route /recette/{slug}/image avec nosniff et une CSP verrouillée.
- Numéro d'entrée attribué automatiquement, à la suite du dernier.

```php
<?php       You, il y a 20 heures • ajout de la logique app , add fiche …

namespace App\Support;

You, il y a 20 heures | 1 author (You)
class Duration
{
    /**
     * Met une durée en minutes sous forme lisible : « 45 min », « 2 h 30 »,
     */
    public static function format(int $minutes): string
    {
        if ($minutes <= 0) {
            return '—';
        }

        if ($minutes < 60) {
            return $minutes.' min';
        }

        $hours = intdiv($minutes, 60);
        $rest = $minutes % 60;

        return $rest === 0
            ? $hours.' h'
            : $hours.' h '.str_pad((string) $rest, 2, '0', STR_PAD_LEFT);
    }
}
```

```php
<?php

use App\Http\Controllers\HomeController;
use App\Http\Controllers\InfoController;
use App\Http\Controllers\RecipeController;
use App\Http\Controllers\StatsController;
use Illuminate\Support\Facades\Route;       You, il y a 21 heures • First commit …

Route::get('/', HomeController::class)->name('home');

Route::get('/infos', InfoController::class)->name('infos');

Route::get('/stats', StatsController::class)->name('stats');

/*
 * « nouvelle » se déclare avant « {recipe} », sinon elle serait prise pour un
 * slug de recette.
 */
Route::get('/recette/nouvelle', [RecipeController::class, 'create'])->name('recipes.create');
Route::post('/recette', [RecipeController::class, 'store'])->name('recipes.store');

Route::get('/recette/{recipe}', [RecipeController::class, 'show'])->name('recipes.show');
Route::get('/recette/{recipe}/image', [RecipeController::class, 'image'])->name('recipes.image');
Route::get('/recette/{recipe}/modifier', [RecipeController::class, 'edit'])->name('recipes.edit');
Route::put('/recette/{recipe}', [RecipeController::class, 'update'])->name('recipes.update');
Route::delete('/recette/{recipe}', [RecipeController::class, 'destroy'])->name('recipes.destroy');
```




1 · Ingrédients
Les quantités se saisissent pour 8 personnes : la fiche les adapte ensuite au nombre de convives.
Ingrédient
Nombre
Donnée
Farine
250
-
×
+ Ajouter un ingrédient
2 · Ustensiles
Ustensile
Casserole
×
+ Ajouter un ustensile
3 · Étapes
Total : —
Une recette compte au moins une étape. La durée totale se calcule toute seule.
étape 1
×
Titre de l'étape
Durée (min)
Préparer la pâte
0
Détail
Mélanger la farine et le beurre du bout des doigts…

3.7 Étape 6 — Documentation et ménage

Six petits commits de documentation et de nettoyage : README entièrement réécrit puis raffiné à plusieurs reprises, suppression de l'ancien logo devenu inutile, création de Recettes.md, la liste des recettes encore à saisir (fajitas, pizza, spaghetti bolognaise, riz poulet crème fraîche…).

3.8 Étape 7 — UX et écran de stats

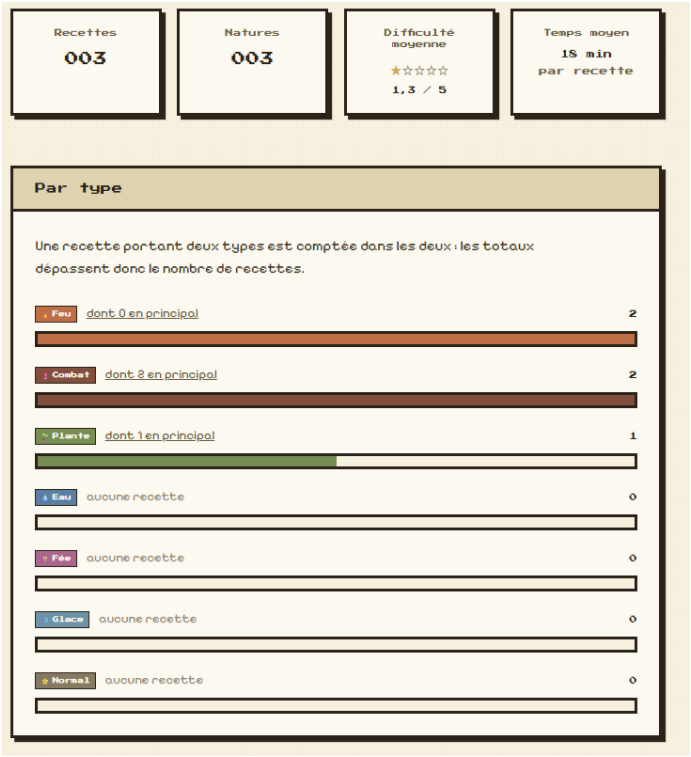
Second gros commit du projet : 1 004 lignes ajoutées sur 19 fichiers.

- App\Support\Difficulty : l'échelle d'étoiles 1 à 5 (Simple, Facile, Moyenne, Difficile, Très dure) — libellés, descriptions et étoiles au même endroit.
- App\Support\Servings : la règle des 2 personnes. Toute recette est écrite pour deux ; le réglage Convives (1 à 20) adapte les quantités à l'affichage seulement, rien n'est réécrit en base. Les durées d'étapes ne bougent pas.
- Page /stats : StatsController + stats.blade.php + composant stat-bar. Quatre compteurs, plus la répartition par type, par nature et par difficulté. Chaque jauge est cliquable et renvoie au filtre correspondant de l'accueil.
- Recherche par nom dans l'en-tête (?q=), combinable avec les autres filtres.

```
    public static function fromRequest(mixed $value): int
    {
        $count = (int) $value;

        if ($count < 1) {
            return self::BASE;
        }

        return min($count, self::MAX);
    }

    /**
     * Le facteur à appliquer aux quantités : 1 pour deux personnes, 2 pour
     * quatre, 0,5 pour une.
     */
    public static function factor(int $servings): float
    {
        return $servings / self::BASE;
    }

    /**
     * « 1 personne », « 4 personnes ».
     */
    public static function label(int $servings): string
    {
        return $servings.' personne'.($servings > 1 ? 's' : '');
    }
}
```

```php
public static function label(int $level): string
{
    return match (self::clamp($level)) {
        1 => 'Simple',
        2 => 'Facile',
        3 => 'Moyenne',
        4 => 'Difficile',
        5 => 'Très dure',
    };
}
```

```php
public function __invoke(): View
{
    $recipes = Recipe::query()
        // La vignette n'a rien à faire ici : on ne lit que les colonnes utiles.
        ->select(['id', 'number', 'slug', 'title', 'primary_type', 'secondary_type', 'nature_id', 'difficulty'])
        ->with('nature')
        ->withSum('steps as total_minutes', 'duration_minutes')
        ->get();

    return view('stats', [
        'totalRecipes' => $recipes->count(),
        'types' => $this->byType($recipes),
        'natures' => $this->byNature($recipes),
        'withoutNature' => $recipes->whereNull('nature_id')->count(),
        'difficulties' => $this->byDifficulty($recipes),
        'averageDifficulty' => $recipes->isEmpty() ? null : $recipes->avg('difficulty'),
        // avg() écarte les null : une recette sans étape ne tire pas la
        // moyenne vers le bas, elle n'a simplement pas de durée.
        'averageMinutes' => $recipes->avg('total_minutes'),
        'longest' => $recipes->sortByDesc('total_minutes')->first(),
    ]);
}
```



Fichier : 3.9 Étape 8 — Correctif stats

Le compteur « temps total » devient « temps moyen par recette », avec une subtilité : les fiches sans étape sont écartées du calcul, pas comptées comme zéro — sinon la moyenne serait faussée. Couvert par 31 lignes de tests supplémentaires.

4. L'application aujourd'hui

4.1 Routes

Route	Rôle
GET /	Le Gourmandex — le grand livre, ses cases, ses filtres (?type=, ?nature=, ?q=)
GET /infos	Les règles : types, natures, difficulté, quantités, lecture d'une case
GET /stats	Le livre en chiffres
GET /recette/nouvelle	Formulaire de création
POST /recette	Enregistrement
GET /recette/{slug}	La fiche complète (?personnes=1..20)
GET /recette/{slug}/image	La vignette, servie depuis la base
GET /recette/{slug}/modifier	Formulaire prérempli
PUT /recette/{slug}	Mise à jour
DELETE /recette/{slug}	Suppression (ingrédients, ustensiles et étapes compris)

4.2 Composition du code

Catégorie	Détail
Enums	RecipeType, Unit
Modèles	Recipe, Nature, Ingredient, Utensil, Step
Contrôleurs	Home, Info, Stats, Recipe
Form request	RecipeRequest
Classes de support	Duration, Difficulty, Servings
Migrations	natures · recipes · ingredients + utensils + steps
Vues Blade	18 vues — 5 pages, 8 composants, 3 partials, 1 gabarit

4.3 Couverture de tests

Fichier de test	Cas
GourmandexTest	17
RecipeFormTest	21
ServingsTest	9
StatsTest	10

Soit 57 cas au total, exécutés par php artisan test.

5. Le design system

Jetons déclarés dans resources/css/app.css via @theme :

Jeton	Usage
parchment-50 → parchment-400	Fonds beige crème
ink-900 → ink-300	Encre brune : texte, bordures, ombres — jamais de noir pur
berry-400 → berry-600	Accent principal (boutons d'action)
leaf-500 / leaf-600	Accent secondaire
sun-400	Surbrillance, sélection de texte
type-feu, type-eau, …	Une teinte par type, assourdie pour tenir sur le parchemin

Trois utilitaires portent le style pixel : pixel-frame (bordure 4 px + ombre pleine 5 px, sans arrondi), pixel-frame-sm (ombre 3 px) et pixel-button (s'enfonce au survol et au clic, en transitions steps()).

Les classes de couleur des types sont écrites dans `RecipeType`, pas dans les vues — d'où le `@source '../../app/**/*.php'` d'app.css, pour que Tailwind les voie.