ARTRAGIS

CC BY

Geste de savoir

A l'assault d'une nouvelle techno : laravel et react

28 février 2023

Table des matières

Introduction

Ça fait un petit moment que j'avais pas réussi à vraiment me mettre à un projet perso avec une nouvelle techno.

J'ai essayé de faire du flutter (et donc dart), mais à peine ai-je installé les dépendances que j'ai abandonné le projet. Idée trop ambitieuse pour être un projet jouet, motivation pas présente, fatigue, et la foultitude d'excuses habituelles. Quand ça veut pas, ça veut pas. Je ne culpabilise pas pour ça, mais ça me manquait.

Mais voilà, j'ai envie d'apprendre react. Et quand on arrive sur le site de react, ça semble pas trop complexe. Mais leur histoire de todolist, ça a le don pour… me démoraliser. Ça fait 7 ans et demi que mon métier c'est développer, j'ai plus la foi de revenir aux tuto mode SDZ. Ça par contre ça me démoralise car y'a encore trois/quatre ans j'étais un défenseur ardent de ces tutos là.

Bref, pas très loin d'une todo list mais un peu éloigné quand même, j'ai une idée et surtout ma flemme de chercher un des nombreux outils qui le fait déjà correctement me pousse à vouloir développer un truc. Du coup allons-y alonso !

1. L'idée

Mon idée est assez "simple" à comprendre et a l'intérêt de pouvoir vraiment y aller par petite itération en fonction de méta-objectif que je me fixerai.

Savoir quel objet est encore sous garantie sans m'y perdre dans le monceau de papier, de pdf numérisé en deux deux etc.

Mon idéal qui me ferait dire que "c'est terminé", ça serait une app web qui propose:

— tu prends ton ticket de caisse en photo ou tu upload un PDF
— tu dis quel est le produit, sa classe, là où tu l'as acheté
— tu dis si tu as pris une extension de garantie

— kaboum tu as ta liste de produits non expirés.

Ensuite ça se jouera avec un peu de recherche, l'upload vers un nextcloud ou un s3. Si j'ai la foi une fois cette étape terminée, je ferai peut être un peu d'extraction/ocrisation pour prépremplir les champs à partir de la photo. Mais c'est hors objectif.

Enfin, je voudrais avoir des objets contextuels venus de l'extérieur (notes de réparabilité par exemple). Bref on verra tout ce que je peux ajouter au fur et à mesure.

2. Contexte et techno

J'ai l'idée, elle est simple, j'ai aussi ma contrainte: j'utiliserai react. Et comme j'ai pas envie de m'embarquer dans des lieux que je ne connais pas bien, on sera sur une app web.

Côté technologie de back, je vais revenir à mes tous premiers amours du lycée: PHP. Et comme plusieurs personnes sur zds ont parlé de laravel, j'ai vu de la lumière, je suis entré. En plus, assez rapidement[1] on trouve des tools officiels pour faire du react.

Pourquoi PHP ? car je sais que je peux le faire tourner sur un hébergement mutualisé, notamment celui que j'ai avec mon nom de domaine francoisdambrine.me. Parce que même si ce langage se traine une salle image il a un avantage certain: il fonctionne.

Pourquoi Laravel ? Car flemme de tout faire en vanilla. Car symfony ne m'a pas laissé un souvenir impérissable à l'époque où je l'ai utilisé. Pas beaucoup d'autres raisons, mais là je pars vraiment en mode chill. Avoir un truc qui va générer pour moi l'authentification, le csrf, tous les petits bidules relou, c'est vraiment bien.

En somme il s'agit d'avoir un équilibre entre le nouveau et la zone de confort. C'est un projet perso, pas mon défi de l'année. J'ai pas forcément non plus envie d'ajouter une ligne à mon CV.

3. À l'abordage !

Je suis sous windows.

Commençons par tout installer:

— pour PHP, on prend les download windows de base , une fois dézippé, on met php dans le path, ça aidera pour la suite.
— Pour composer, je prends le setup.exe. Au cas où vous ne le sauriez pas, ce setup installe composer dans `il installe composer dans` C:\ProgramData\ComposerSetup\bin\composer‘
— Pour node, je prends l'installateur officiel en v18.
— Ah et aussi, il faut installer 7zip en non portable (ou mettre l'exécutable dans le Path). Du moins ça facilitera la vie pour composer, prendre les dépendances via zip étant 10 000 fois plus rapide que depuis les sources git.

1. "rapidement" c'est presque vite dit, on y reviendra après, tout n'a pas été aussi smooth que prévu mais pas de quoi casser trois pattes à un canard.

il faut modifier le fichier php.ini sinon vous allez avoir des soucis. Allez dans la partie des extensions et décommentez les extensions :

- fileinfo ;
- pdo_mysql ;
- intl ;
- mbstring.

Comme j'ai une license, je lance phpstorm et en avant!

J'ai fait l'erreur de chercher "laravel et react" sur google avant de commencer. Pourquoi est-ce une erreur ? Car google référence en top page les tutos pour laravel 9 et laravel 7. Laravel est en version 10 et aucun des tutos référencés ne fonctionne. Par exemple celui-ci qui arrivait 2ème lien utilise laravel-mix en le présentant comme "installé par défaut avec laravel". D'ailleurs laravel-mix le dit aussi.

Les libs, notamment JS ne sont pas présentes dans laravel 10.

Heureusement, comme je l'ai dit plus haut, la doc officielle de laravel contient un "how to" qui explique comment utiliser vue ou react avec laravel . Alors forcément j'en profite.

Maintenant on peut lancer la commande initiale:

`composer create-project laravel/laravel guarantee-tracker`, le franglish est volontaire.

Une fois démarré, je saute tout de suite aux commandes pour mettre en place inertia et je configure mon fichier .env pour me connecter à un mysql.

```
1 php artisan breeze:install react
2
3 php artisan migrate
4 npm install
5 npm run dev
```

Listing 1 – on note que `artisan` est le point d'entrée de laravel pour faire du dev. C'est l'équivalent du `manage.py` de django.

A partir de là, je vais plus trop coder pendant une heure ou deux, non je vais m'adonner à mon activité favorite: lire le code.
Ça peut être con dit comme ça, mais ça me permet d'immiter ce qui est déjà fait tout en sachant *pourquoi* je le fais. Certes je n'ai pas encore la maîtrise pour le faire d'instinct mais au moins j'y vais pas au petit bonheur.

4. Un peu de code

Première étape: obtenir un formulaire qui ne s'affiche que si on est sélectionné, quand on le valide on a la liste de nos éléments sauvegardés qui apparaît.
Cet objectif basique permet de prendre en main:

- la notion de composant react tant que je crée que ceux qui existent déjà ;

— l'orm de laravel
— le routeur et l'authentification.

C'est la base mais quelle base!
Coup d'oeil à la doc, pour créer un modèle, il y a une commande. Je lance donc `php artisan make:model Guarantee --controller --resource --requests`. Naïvement, expérience de django et de son ORM (mais aussi de ASP.NET), je me dis que j'ai pas besoin de créer la migration de suite, elle sera générée quand j'aurais fait mon modèle.
Je commence donc à voir comment il faut que je configure mon modèle, quelles sont les annotations et. rien. Rien qui me dit comment faire pour lui donner ses trois/quatre premiers champs. Je sais que par défaut il aura un id, un creation_date et un update_date. C'est TOUT.
J'abandonne pour l'instant le modèle, je me dis que la fatigue est trop forte que la phrase qui explique tout et qui est devant mon nez est juste devenue invisible à cause de ça.
Je passe donc au contrôleur. Grâce à ma petite activité de lecture du code, je comprends vite comment faire ma vue react et ses composants.

```
php artisan breeze:install react

php artisan migrate
npm install
npm run dev
```

Listing 2 – la vue react avec la liste des garanties et un formulaire.

J'ai donc sélectionné mon layout qui doit avoir une authentification, plus qu'à dire à mon contrôleur qu'il doit forcer l'authentification.
Et c'est gagné avec simplement:

```
<?php

Route::middleware('auth')->group(function () {
    // existait déjà
    Route::get('/profile', [ProfileController::class,
        'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class,
        'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class,
        'destroy'])->name('profile.destroy');
    // mes vues !
    Route::get('/calendar', [GuaranteeController::class,
        'list'])->name('guarantee.cal');
    Route::post('/calendar/new', [GuaranteeController::class,
        'create'])->name('guarantee.new');
});
```

Listing 3 – le fichier routes/web.php

Je vous passe le développement du composant "Guarantee" qui ne fait qu'afficher les trois/quatre champs prévus des garanties (pour l'instant), avec un simple calcul du temps restant en JS.

Maintenant, pour que tout fonctionne, il n'y a plus qu'à créer le composant du formulaire et surtout mon modèle.
Je trouverais mon bonheur sur une page stackoverflow où ils expliquent qu'il suffit d'écrire la migration et dire les champs qu'on va mettre. Voilà.
Je dois quand même m'y prendre en deux fois et voici mes deux migrations.

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('guarantees', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('retailer');
            $table->integer('extra');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('guarantee');
    }
};
////
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table('guarantees', function (Blueprint $table) {
            $table->foreignIdFor(User::class);
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        //
```

```
44      }
45  };
```

A partir de là, tout sera assez simple. Le contrôleur ressemble vachement à ce dont j'ai l'habitude en PHP et mine de rien, premier objectif atteint:

```php
<?php
class GuaranteeController extends Controller
{

    public function list(Request $request): Response
    {
        $user = $request->user();
        $guarantees = $user->guarantees()->get();
        return Inertia::render('Guarantee/Calendar', ['guarantees'
            => $guarantees]);
    }

    public function create(StoreGuaranteeRequest $request):
        RedirectResponse
    {
        $guarantee = new Guarantee();
        $guarantee->user()->associate($request->user());
        $guarantee->name = $request->name;
        $guarantee->retailer = $request->retailer;
        $guarantee->extra = $request->extra;
        $guarantee->save();
        return Redirect::to(route('guarantee.cal'));
    }
}
```

Ma seule remarque c'est que décidément je n'aime pas le formattage moderne du PHP avec les accolades à la lignes. C'est d'un moche!

Conclusion

Voilà pour la première étape de mon périple. Prochain objectif : ajouter un peu d'interaction dans mes pages, peut-être un peu de style aussi. Surtout obtenir bien plus de détails sur les éléments.
Je voudrais aussi développer une vue "calendrier".
Ensuite on passera aux photos.