

Runners



Bastien Nicoud
Nicolas Henry
2018

cpnv
Centre professionnel du Nord vaudois

Si-T1a

Sommaire

Introduction	3
Objectifs de l'application	4
Situation de départ	4
Versions	4
Outils utilisés	5
Bulma	5
Mix	5
Base de données	6
Modèle de données	6
Migrations	7
Seeding	7
Architecture de dossier	8
Dossiers	8
App	8
Ressources	9
L'application	9
API	10
Composants spécifiques à l'application	10
Blade components	10
Système de filtres	11
Gestion des statuts	11
Difficultés rencontrées	12
Reprise du projet	12
API	13
Création des runs	13
Conclusion	15
Annexes	16

Introduction

Runners est une application web de gestion, qui permet de coordonner les différents chauffeurs qui effectuent des transports d'artistes lors du paléo festival de Nyon. L'objectif est de proposer une plateforme qui permette de gérer la présence des chauffeurs, leur planning, les véhicules ainsi que les trajets durant le festival. Le back-end propose aussi une API qui est utilisée par l'application mobile, développée dans le cadre d'un autre projet.

Cette application est basée entièrement sur le framework Laravel et utilise Bulma pour toute la mise en forme. Quelques packages supplémentaires ont été intégrés, notamment pour la gestion des horaires, et des groupes.

Vous trouverez dans ce document toutes les informations techniques sur ce projet, ainsi que les difficultés rencontrées jusqu'au stade actuel du projet.

Objectifs de l'application

L'objectif de Runners est d'offrir un outil de gestion aux personnes qui se chargent de transporter les artistes pour le paléo festival de Nyon, qu'on appelle des runners.

L'application offre une plateforme de gestion des 'runs' (trajets effectués par les runners, ainsi qu'une api pour permettre à une application mobile de se greffer au système.

La gestion des runs requiert :

- Un système de gestion des utilisateurs (avec authentification).
- Un système pour gérer des groupes d'utilisateurs.
- Un système pour associer des horaires à ces groupes via un agenda.
- Un système pour gérer des véhicules.
- Un système de gestion des runs, qui lie la plupart de ces composants.

L'api permet de récupérer la plupart des ces informations, ainsi que de manager les runs (les démarrer/arrêter, ainsi que de prendre part à un run).

Situation de départ

Nous avons commencé le projet début 2018, en repartant de la version 1 de runners, qui a été développée par la volée précédente de techniciens. L'application est développée à l'aide du framework Laravel ainsi que de quelques extensions.

Après réflexion, il a été décidé de reprendre l'application de zéro, en conservant les éléments propres et réutilisables. Le reste des fonctionnalités sera redéveloppé en respectant au maximum les principes de laravel. L'objectif étant de construire une application plus facile à reprendre pour les prochains, et proche des concepts proposés par Laravel.

Au final seuls la base de données, ainsi que quelques blocs de code seront conservés pour commencer le développement de la v2 de runners.

Versions

Pour la v2 de runners nous décidons d'utiliser Laravel 5.6, dernière release en date lors du développement.

Outils utilisés

Cette section décrit et présente l'outillage utilisé pour le développement de l'application, nous parlerons uniquement des outils spécifiques, les outils utilisés de base par Laravel (comme composer) ne seront pas abordés. Se référer à la documentation officielle¹.

Nous utilisons deux outils supplémentaires avec Laravel, qui concernent le front-end.

Bulma

Bulma est le framework css que nous avons décidé d'utiliser pour la v2 de runners. Il propose un style de base assez simple, et une série de classes intuitives pour l'utiliser. De plus, les composants qu'il propose correspondent bien aux besoins de l'app.



Il dispose d'une documentation complète et simple, et il est possible de customiser facilement les couleurs via des variables scss.

Site officiel : <https://bulma.io/>

Mix

Mix est une surcouche de webpack² qui permet de gérer les assets facilement, il est fourni dans l'installation de base de Laravel, et propose une configuration par défaut prévue pour être utilisée avec les vues blade.

Mix se charge de transpiler tous les assets utilisés côté client de l'application (JavaScript et css). Il utilise une configuration par défaut qui suffit à nos besoins, on peut cependant facilement la modifier à travers le fichier `webpack.mix.js`.

Les fichiers originaux, sont situés dans le dossier `resources/assets` et sont transpilés par mix dans `public/**` dépendant du type d'asset.

Le fichier `webpack.mix.js` décrit exactement quels fichiers sont transpilés vers quelle destination.

Pour lancer la génération des assets il existe 3 commandes à taper à la racine du projet:

- **npm run dev** transpile pour le développement (avec sourcemaps)
- **npm run watch** retranspile automatiquement à chaque modification de fichier
- **npm run prod** transpile pour la production (minifié et nommé avec un hash)

¹ <https://laravel.com/docs/5.6>

² <https://webpack.js.org/>

La configuration de base de mix utilise plusieurs modules webpack qui permettent :

- JavaScript
 - Utiliser le standard de code ES6³
 - Utiliser les imports de modules (à la node.js)
 - Réécriture des urls (pour charger les modules node js)
- SASS
 - Réécriture des urls (~ permet de charger un module node.js)

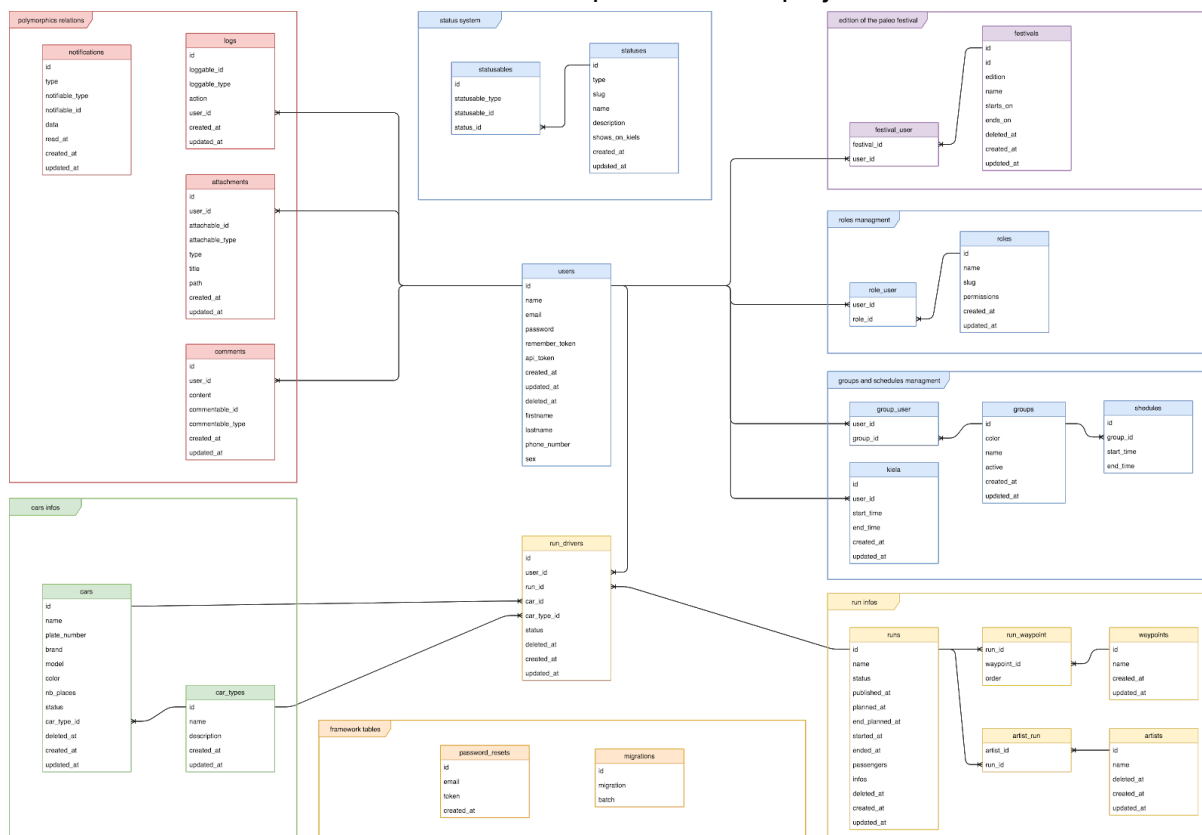
Base de données

L'application utilise mysql pour gérer la base de données au travers d'Eloquent⁴, l'ORM de laravel. La base de donnée est générée avec les migrations de laravel et remplie avec les seeders aussi inclus dans laravel. Cela permet de changer facilement la base de données utilisée et modifiant le driver utilisé par laravel.

Il est possible d'utiliser SQLite⁵, postgresSQL⁶ et MariaDB⁷ en utilisant les migrations fournies dans le projet.

Modèle de données

Vous trouvez une version zoomable sur le dépôt GitHub du projet.



³ <http://es6-features.org/#Constants>

⁴ <https://laravel.com/docs/5.6/eloquent>

⁵ <https://www.sqlite.org/index.html>

⁶ <https://www.postgresql.org/>

⁷ <https://mariadb.org/>

Toute la base de données est construite autour de la table `users`, un utilisateur est en fait : un runner, un coordinateur ou un administrateur.

Chaque user est lié à un rôle, qui définit ses permissions dans l'application ainsi qu'à un groupe, qui définit ses présences lors du festival via la table `schedules`, dans laquelle sont stockés les horaires des groupes.

Les tables en rouge utilisent des relations polymorphiques, ce qui permet de facilement lier leur contenu à n'importe quelle table dans la base de données. Par exemple, les commentaires sont attachés à des utilisateurs et des runs. Mais il est possible de les attacher aux véhicules aussi si nécessaire...

En dessous de la table `users`, on trouve la table `run_drivers`, cette table permet de faire le lien entre un run, et des chauffeurs/véhicules. C'est une table intermédiaire entre 4 tables. Un run peut avoir plusieurs `run_driver`, et chaque `run_driver` définit quel utilisateur le conduit, avec quel véhicule.

Migrations

Pour générer la base de données. Il suffit de configurer le driver selon votre environnement, et de lancer les migrations avec la commande `php artisan migrate:fresh`. Qui va construire toute la base de données de zéro.

Artisan exécute alors les fichiers de migrations⁸ situés dans `database/migrations` dans l'ordre chronologique.

Si vous souhaitez modifier la base de données, il vous suffit de créer une nouvelle migration, qui fera évoluer la base de données incrémentalement, depuis l'état des précédentes migrations.

Seeding

Le seeding⁹ vous permet de remplir la base de données avec des données fictives pour tester l'application.

Après avoir migré la base de données, vous pouvez lancer la commande `php artisan db:seed` pour la remplir automatiquement, les seeders sont dans le dossier `database/seeds`.

Si vous souhaitez migrer et seeder la base de données d'un coup, `php artisan migrate:fresh --seed` reconstruit la base de données de zéro et la seed.

⁸ <https://laravel.com/docs/5.6/migrations>

⁹ <https://laravel.com/docs/5.6/seeding>

Architecture de dossier

Dans ce chapitre nous verrons comment l'application est organisée, ou trouver les différents fichiers. Nous avons fait en sorte de respecter au mieux la structure de laravel, nous ne reviendrons donc pas sur les concepts décrits dans la doc, mais uniquement sur les petites spécialités liées à l'app, ou fonctionnalités peu décrites dans la doc officielle¹⁰.

Dossiers

La racine du projet est une installation classique de laravel à quelques dossiers près.

A noter, le dossier docs contient la documentation technique détaillée du projet.

Le fichier .eslintrc.js contient la configuration d'eslint¹¹ le linter que nous avons choisi d'utiliser pour valider notre code JavaScript.

Le fichier webpack.mix.js contient la configuration de mix, comme expliqué quelques chapitres plus tôt.

App

Le dossier app contient quant à lui un peu plus d'éléments spécifiques à l'application.

A la racine on trouve les modèles eloquent¹², comme dans une app laravel classique.

On notera les dossiers :

- **Events** : contient des classes qui représentent des événements émis par l'application (voir events laravel¹³).
- **Extensions** : Qui contient des traits PHP que nous avons utilisés pour séparer certaines fonctionnalités et éviter de surcharger les modèles.
- **Listeners** : Contient des classes qui vont réagir aux événements vu un plus haut.
- **Notifications** : Contient des classes qui se chargent d'envoyer des notifications¹⁴, par exemple un mail.
- **Policies** : Qui contient des classes qui permettent de vérifier les autorisations que possède l'utilisateur connecté.
- **Rules** : Contient des règles de validations personnalisées, utilisées par le validateur¹⁵ laravel.

¹⁰ <https://laravel.com/docs/5.6>

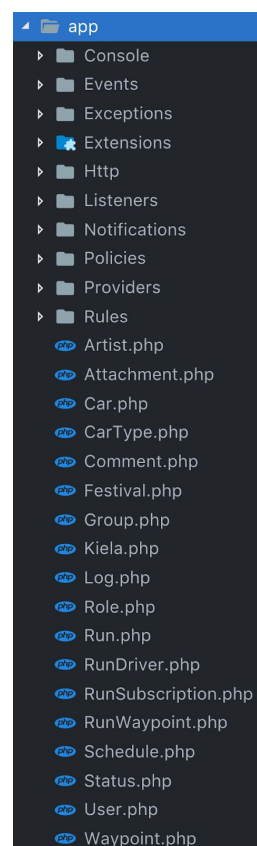
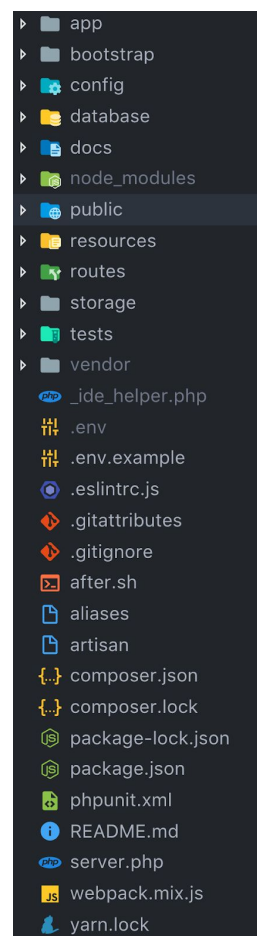
¹¹ <https://eslint.org/>

¹² <https://laravel.com/docs/5.6/eloquent>

¹³ <https://laravel.com/docs/5.6/events>

¹⁴ <https://laravel.com/docs/5.6/notifications>

¹⁵ <https://laravel.com/docs/5.6/validation>

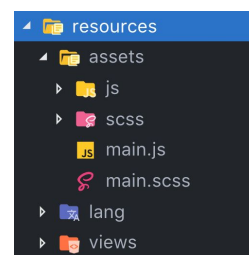


Ressources

Le dossier “resources” contient les vues laravel blade, les fichiers de traductions ainsi que les assets.

C’est dans le dossier “assets” que se situent les fichiers de bases que mix va transpiler dans le dossier public de l’application.

Le js est séparé du scss, et organisé par pages.



L’application

L’app suit la logique de construction de laravel, qui ressemble au pattern MVC. En plus des classiques modèle, vue et contrôleur, nous faisons appel à d’autres fonctionnalités de laravel. Voici le trajet que fait une requête classique dans l’application.

1. Le routeur va aiguiller la requête.
2. La requête traverse la pile de middlewares, dont celui qui va vérifier que l’utilisateur est authentifié.
3. La requête est envoyée au contrôleur correspondant.
4. Si la requête contient des paramètres, ils sont validés par la custom request¹⁶ laravel.
5. Si la requête laravel est validée, le contrôleur fait ses actions :
 - a. Il va vérifier que l’utilisateur authentifié a le droit d’effectuer l’action grâce aux politiques¹⁷.
 - b. Il peut faire appel à des modèles eloquent.
 - c. Il construit les données qui seront passées à la vue pour l’affichage.
6. Une fois le traitement du contrôleur effectué il renvoie la vue laravel correspondante.
7. Blade effectue le rendu de la vue.
8. La vue est renvoyée au client.

Cas spécifiques :

- Événement déclenché par un modèle. Certains modèles peuvent émettre des événements selon l’action effectuée (création, modification), qui seront ensuite traités par un listener laravel¹⁸. Ce listener va pouvoir faire des traitements supplémentaires, par ex envoyer une notification.

La logique métier est contenue dans les modèles, ce qui permet d’alléger les contrôleurs, et aussi d’utiliser la même logique dans plusieurs contrôleurs. Dans certains contrôleurs il faudra donc aller jeter un coup d’œil à la méthode correspondante dans le modèle pour voir le code réellement exécuté.

¹⁶ <https://laravel.com/docs/5.6/validation#form-request-validation>

¹⁷ <https://laravel.com/docs/5.6/authorization#writing-policies>

¹⁸ <https://laravel.com/docs/5.6/events#queued-event-listeners>

API

Les requêtes effectuées sur l'api sont traitées un peu différemment que celle de l'application. Les routes de l'api sont situées dans un fichier de routes séparé, `routes/api.php` qui passe à travers une pile de middlewares différente que les requêtes de l'app.

On trouve notamment un middleware spécial qui permet d'ajouter des Headers CORS, pour autoriser d'autres noms de domaines d'effectuer des requêtes sur l'api (ex l'app ionic).

L'authentification ne se fait non pas par session (comme pour l'app, ou l'on s'identifie par mot de passe, puis une session sauvegarde votre connexion), mais par un token, unique à chaque utilisateur, se token est passé dans les headers de la requête, et permet de vérifier que l'utilisateur est bien identifié. Ce type d'authentification nommé bearer est standardisé dans open api v3¹⁹.

Une fois authentifié, la requête passe par les mêmes étapes que pour l'app, sauf à la fin : cette fois nous ne rendons pas une vue, mais faisons appel à une "API resources"²⁰. Les ressources API, sont des classes PHP qui définissent comment les données doivent être converties pour générer la réponse en JSON qui sera renvoyée. Ces ressources sont directement liées aux modèles eloquent, et vont permettre d'avoir le contrôle sur comment les données sont parsées/formatées (par ex pour les dates).

Toutes les "API resources" sont situées dans le dossier : `app/Http/Resources`.

Composants spécifiques à l'application

Pour nous simplifier la vie, nous avons développé certaines parties de l'application de façon à ce qu'elle soit réutilisable, nous passons en revue ici quelques informations importantes à propos de ces composants :

Blade components

Pour éviter de répéter trop de code HTML, nous avons séparé certains éléments dans des composants blade²¹. Un composant blade est un fichier blade dans lequel nous plaçons notre code HTML, puis quand on importe ce composant, nous pouvons lui passer des données et il générera le HTML en fonction.

Quelques exemples :

- Les petits badges affichés dans l'app pour renseigner sur le statut d'une ressource sont générés à partir d'un composant, on lui passe le statut et il génère le HTML.
- Les champs de formulaires aussi, on passe au composant toutes les infos importantes du champ, et il génère le HTML correspondant.

Les composants blade sont situés dans : `resources/views/components`

¹⁹

<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md#securitySchemeObject>

²⁰ <https://laravel.com/docs/5.6/eloquent-resources>

²¹ <https://laravel.com/docs/5.6/blade#components-and-slots>

Systeme de filtres

Sur le crud des utilisateurs et des runs nous avons besoin d'un système qui permet de filtrer en fonction du statut, et/ou de faire une recherche dans la table. Nous avons donc créé un système qui permet de facilement ajouter ce comportement à n'importe quelle crud.

3 choses :

1. Un trait PHP est rajouté au modèle que vous voulez rendre filtrable. Ce trait va rajouter des scopes, qui vont permettre de modifier la requête SQL en fonction de ce que l'utilisateur souhaite obtenir. Le trait se situe dans `app/Extensions/Filters`.
2. Dans la vue qui affichera le contenu, il faut rajouter le composant blade qui générera la box qui contient tous les filtres. Vous pouvez passer une paramètre de ce composant le nom des statuts par lesquels vous voulez que l'utilisateur puisse filtrer la requête.
3. Dans le contrôleur, lorsque vous appelez le modèle, il faut utiliser le scope ajouté grâce au trait.

Quand l'utilisateur utilise un filtre, une requête sera effectuée, avec les paramètres de la recherche dans l'url. Le scope va déduire de ces paramètres les filtres à rajouter sur la requête SQL, et retourner les résultats.

Gestion des statuts

Dans l'optique de centraliser les statuts que les différentes ressources peuvent avoir (par ex. les utilisateurs peuvent être : libre, occupés, absents...), nous avons créé une table séparée qui stocke ses statuts, on peut ensuite la lier avec n'importe quelle ressource grâce à une relation polymorphique.

Afin de ne pas répéter la relation polymorphique, et les quelques méthodes qui tournent autour de la gestion des statuts, nous avons séparé le tout dans un trait séparé. Il suffit de l'ajouter au modèle auquel nous souhaitons pouvoir ajouter des statuts.

Le trait se situe dans `app/Extensions/Statusable.php`.

Une fois ajouté, on a accès à quelques méthodes :

- `statutes()` : Permet d'accéder directement à la relation polymorphique.
- `status()` : Retourne le statut actuel.
- `setStatus($newStatus)` : Permet de fixer un nouveau statut, par son slug.

Difficultés rencontrées

Ici nous rentrerons plus en détail sur les quelques difficultés rencontrées lors du projet, les solutions trouvées et intégrées au projet.

Reprise du projet

Nous avons repris Runners début 2018, à ce stade, il existait déjà une v1, développée par la volée précédente d'étudiants.

Nous avons pris du temps pour nous plonger dans le code et s'imprégner de ce qui avait été fait. Après analyse nous avons pu relever certains problèmes, qui nous ralentissait dans notre reprise du code, et aurait rendu très compliqué de continuer le projet proprement.

Les points que nous avons notés :

- Le projet, bien que basé sur Laravel, utilise énormément d'extensions externes au framework qui ajoute des fonctionnalités pourtant déjà intégrées à laravel. Cela force à maintenir/vérifier régulièrement plusieurs dépendances externes pas forcément maintenues, alors que nous pourrions utiliser ce que le framework propose directement.
- Une architecture de dossier/projet qui diffère beaucoup de la structure proposée par laravel. Les dossiers ne respectent pas les mêmes standards que propose laravel, certaines parties du code ont été séparées pour des raisons justifiées, mais dans des endroits différents de ce que laravel propose. Cela rend l'architecture très difficile à comprendre/utiliser, car différente de ce que Laravel documente. Le projet ne contenant pas de documentation, il était difficile de reprendre cette architecture.
- Une doc très légère, la documentation du projet ne renseigne pas ou très peu sur les fonctionnalités implémentées, les spécialités liées à l'application et ça conceptions, ce qui rend la compréhension du code difficile.
- Un code peu commenté.

Suite à ces différents constats, nous avons discuté avec les différentes parties du projet, et décidé de reprendre le projet depuis une installation propre de laravel, et de récupérer uniquement les parties saines de la v1 de l'app (ex. base de données). Le reste de l'app sera reconstruite de zéro, en essayant de respecter au maximum ce que propose Laravel.

L'objectif étant de rester le plus proche de ce que le framework propose dans son installation par défaut, et d'utiliser des packages externes que dans des cas justifiés.

API

Lorsque nous avons attaqué le développement de back-end, une équipe de deuxième année, se chargeait de développement de l'application mobile Runners, qui utilisait notre API, il fallait donc leur mettre à disposition rapidement une API qui respecte les mêmes formats que l'ancienne api, tout en travaillant avec la nouvelle architecture de l'application.

Nous avons donc développé une API en nous basant sur les spécifications de l'ancienne API. En utilisant cette fois les "API ressources" proposées par laravel, qui permettent de transformer les données récupérées depuis la base de données vers des JSON pour les rendre au client.

Une fois la nouvelle API testée par nos soins, pour vérifier sa conformité avec l'ancienne, l'application mobile runners a pu de manière transparente migrer vers la nouvelle API. Nous avons finalement effectué les tests finaux avec l'application mobile pour vérifier que tout fonctionne correctement.

A noter :

- L'ancienne API utilisait un token "X-acces-token" pour l'authentification. Que nous avons changée pour un token "bearer" comme proposé par défaut par laravel, ainsi que défini dans la norme open api 3.
- Certaines routes ont été renommées, pour respecter au mieux le principe de crud. Cependant, nous avons gardé les anciens endpoint pour garantir la rétrocompatibilité. Dans le développement d'une future version de l'app mobile, il faudrait idéalement migrer vers le nouveau nommage des endpoints.

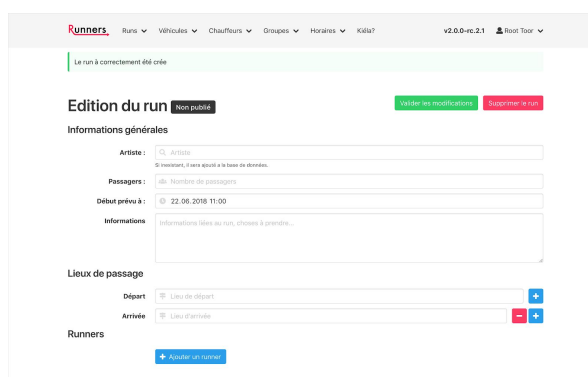
Création des runs

La page permettant de créer un run soulève un certain nombre de problématiques. Nous allons ici voir les quelques solutions que nous avons choisi d'utiliser dans ce formulaire.

Pour créer un run, nous devons faire un certain nombre d'insertions, dans plusieurs tables différentes, et certaines parties du formulaire sont dynamiques, donc pas forcément pareilles entre deux runs, il faut donc pouvoir valider le formulaire différemment, dès la création, alors qu'il n'y a encore aucune données dans la base de donnée.

Il nous faut aussi pouvoir modifier le formulaire alors qu'il n'est pas encore soumis.

Pour finir, nous devons fournir une auto complétions à l'utilisateur pour certains champs.



- **Validation du formulaire :**

Pour valider le formulaire de la même manière, qu'il ait 1 ou 4 waypoints, un chauffeur ou aucun chauffeur, nous avons utilisé les tableaux.

Lorsque l'on nomme un champ HTML, nous pouvons utiliser une notation avec des crochets, pour que lors de l'envoi les données soient regroupées sous forme d'un tableau. Dans notre formulaire, les champs de waypoints sont donc nommés `waypoint[1]` puis `waypoint[2]`...

Lors de la soumission du formulaire, tous les waypoints, identifiés par l'id entre crochets seront regroupés dans un tableau `waypoints`. Le validateur laravel permet ensuite de valider tous les éléments d'un tableau avec les mêmes règles en utilisant la notation : `waypoints.* => 'rules'` ici tous les waypoints seront validés par la liste de règles après la flèche.

- **Ajout dynamique d'éléments au formulaire :**

Pour permettre à l'utilisateur de rajouter des éléments dynamiquement au formulaire, sans perdre les données, par ex, un waypoint de plus. Il nous fallait générer un champ de plus :

Plusieurs solutions, le faire avec JavaScript, ou PHP. Le faire en JS posait un problème, c'est qu'en cas de rechargement de la page, le nouveau champ n'avait pas été persisté. Ne voulant pas rajouter d'AJAX nous avons opté pour de faire depuis laravel : La solution pour laquelle nous avons opté est assez simple, chaque bouton qui permet de faire une modification du formulaire effectue en fait une soumission du formulaire, en rajoutant une clé qui définit l'action du bouton.

Lorsque laravel traite la requête, il regarde si une action est présente dans les données du formulaire, si oui, il l'effectue (par ex. rajouter un waypoint). Puis il redirige l'utilisateur sur le formulaire, qui s'affiche alors mis à jour.

Cette méthode offre aussi l'avantage que les données du formulaire sont bien validées à chaque action.

Le désavantage est d'effectuer un aller-retour au serveur à chaque action, nous l'avons jugé acceptable.

- **Persistance dans la base de données :**

Pour sauvegarder les différentes données correctement dans chaque modèle, nous avons choisi de déléguer le travail. Le contrôleur va appeler une méthode sur le modèle `run` qui va elle se charger de dispatcher sur les modèles liés en lui transmettant les données nécessaires.

On appelle donc la méthode de sauvegarde sur `App\Run` en lui passant toutes les données du formulaire, la méthode analyse et dispatch, par ex en appelant `App\Waypoints` en lui passant les waypoints à sauvegarder.

Conclusion

Il faut retenir :

- La structure du projet respecte le plus possible ce que laravel propose, on peut donc se référer à la doc officielle. A noter quelques spécialités :
 - app/Extensions : Contient des traits qui permettent d'ajouter des fonctionnalités à des modèles eloquent.
 - app/Http/Resources : Contient les classes qui se chargent de transformer les données eloquent sous forme de JSON pour l'api.
 - Les assets avant compilation sont dans ressources/assets, une fois transpilés ils sont transférés par mix dans le dossier public.
- La génération de la base de données se fait à l'aide des migrations et seeders, on peut facilement modifier ces derniers pour changer les données de test.
- Chaque utilisateur possède un rôle, qui définit les actions qu'il peut effectuer, nous vérifions ce qu'il peut faire en utilisant les polices laravel, qui sont chacune liées à un modèle eloquent : RunPolicy va vérifier les permissions de l'utilisateur pour le modèle Run.
- L'API utilise des middlewares différents de l'application, notamment un qui ajoute des headers CORS.
- Plusieurs bouts de HTML qui sont régulièrement utilisés ont été séparés dans des composants blade. Par ex. les champs de formulaire, ou encore les tags de statut.
- Tous les cruds de l'application sont relativement simples, sauf celui des runs, qui intègre quelques spécialités liées à l'enregistrement des données, on utilise pas les habituelles méthodes eloquent (fill, save) mais une méthode personnalisée qui se charge elle d'exécuter les actions eloquent sur les bons modèles.

Le projet a été versionné avec git en utilisant la méthodologie git flow, il est donc possible de retourner voir dans l'historique les commentaires des commits, qui peuvent éventuellement aider sur des bouts de l'application qui ne serait pas claire.

Pour plus de détails techniques sur le code, n'hésitez pas à consulter la documentation technique en markdown et en anglais qui se situe dans le dépôt.

Annexes

- Dépôt GitHub : <https://github.com/CPNV-ES/Runners-Laravel>
- Liste des releases effectuées :
<https://github.com/CPNV-ES/Runners-Laravel/releases>
- Liste des bugs encors présents, ou des fonctionnalités à implémenter :
<https://github.com/CPNV-ES/Runners-Laravel/issues>
- Laravel : <https://laravel.com/docs/5.6>
- Bulma (framework CSS) : <https://bulma.io/>
- Mix (gestion des assets) : <https://laravel.com/docs/5.6/mix>
- Diagrams technique de l'application :
<https://github.com/CPNV-ES/Runners-Laravel/tree/master/docs/diagrams>
- Documentation technique en anglais :
<https://github.com/CPNV-ES/Runners-Laravel/tree/master/docs>
- Procédure d'installation :
https://github.com/CPNV-ES/Runners-Laravel/blob/master/docs/install/1_requirements.md
- One pager et présentation :
<https://github.com/CPNV-ES/Runners-Laravel/tree/master/docs/2018>