Favre Sylvain - BTS SIO 1 - 25 Juin 2025

PHP/API/HTML/CSS/JS
Stage SIO1 - Rapport

Entreprise Team Moto Quad / Esprit Moto Quad / Moto Quad Passion
Maître de stage Leduc Kevin
Coordonnées kl@holding-alpes-moto.com (+33) 04 92 53 43 88
Localisation 2 Allée du Torrent 05000 Gap
Prof. Référent Marchand Sébastien

Remerciements

Merci à mon professeur principal Monsieur Marchand de m'avoir introduit le stage en début d'année, pour les compétences qu'il m'a fait travailler et que j'ai pu mettre en place la bas, et pour l'aide fournie pendant les difficultées subies par mes problemes de santé

Merci a mon tuteur de stage Leduc Kevin de m'avoir pris en stage , de m'avoir appris les bases du travail en groupe, de l'architecture web et de ce que c'est d'être un développeur au travers de plusieurs projets intéressants et qui apportent une réelle aide aux autres

Merci à Kévin, Bastien Jem et Julie (la team de développeurs) de m'avoir bien accueillis au sein de la structure

Merci à Cynthia de Mission Locale d'avoir soutenu ma candidature au bts Sio et de m'ouvrir les portes du monde de l'informatique

Sommaire

Introduction..... 3
I - Présentation de l'Entreprise..... 4
I.1 - Panorama.....4
I.2 - Organigramme..... 5
I.3 - Secteur d'activité.....6
I.4 - Pestel..... 8
I.5 - Historique.....9
I.6 - Environnement de travail..... 14
II - Missions et tâches effectuées..... 20
II.1 - Présentation du tuteur.....20
II.2 - Les différentes missions.....21
II.3 - Etude de cas..... 22
II.3.α - Prise en main des technologies..... 22
II.3.β - Outil de suivi de commandes DAFY ORDER..... 27
II.3.γ - Outil de visualisation des stocks STOCK VÉHICULE..... 34
III - Les bénéfices et inconvénients.....38
III.1 - Ce que le stage m'a apporté.....38
III.2 - Les problèmes rencontrés et solutions proposées.....40
Conclusion.....41
Annexes..... 42
Les Photos..... 43
Le Code.....44
Le Component Custom Double Slider.....46
Les Prototypes..... 47
Les Schémas.....49
Les Liens..... 54

Introduction

Lors d'une nuit de décembre 2021, j'ai décidé de terminer la licence de géologie que j'avais commencé afin de partir dans le domaine de l'informatique. Mon diplôme en poche, j'ai traversé les mers de l'auto-formation, défié les entreprises de mes cv et lettres de motivations avec Mission Locale a mes cotés, et trouvé la ligne de départ de cette nouvelle voie dans le bts SIO de Gap, caché derrière le lycée même ou j'ai passé mon bac.

Et aujourd'hui ?

Je suis déjà en train de travailler sur des technologies complexes et des outils qui aident les autres à avancer aux côtés de mon tuteur de stage et de son équipe. Et si il est vrai que je n'ai pas fait beaucoup de démarchage pour rentrer dans le meilleur stage d'informatique de la ville , j'y ai tout de même découvert ma place en tant que Développeur Web Junior.

De quel stage je parle ?

Au-dessus des locaux de TEAM MOTO QUAD, se trouve une branche informatique bien huilée. Introduite par notre professeur principal, monsieur Marchand, la notion qu'un magasin de vente, réparation et location de véhicules motorisés ait besoin de tout un service informatique peut paraître étonnante. Cependant, une telle activité nécessite de coordonner et faciliter le travail de ses employés, souvent avec des outils informatiques performants. Elle nécessite aussi un site internet sur lequel les usagers peuvent trouver pièces, prix et conseils. On se rend alors compte que la SAS (Société par Actions Simplifiées, se développe grâce à ses actions et de moindres restrictions légales) à des besoins internes en informatique qui ne peuvent être satisfaits que par l'existence d'un service informatique polyvalent et à l'écoute des autres salariés. Et j'ai pu en faire partie le temps de 6 semaines, du 26 mai au 6 juillet 2025.

C'est quoi Esprit Moto Quad ? Tu as fait quoi pendant ton stage ? Qu'en as-tu tiré ?

On va justement répondre à ces questions dans ce rapport. Lisez la suite et vous verrez !

I - Présentation de l'Entreprise
I.1 - Panorama

TEAM MOTO QUAD est une société par actions simplifiée (SAS) créée le 16 mars 2003. Son siège social est situé depuis le 30 novembre 2020 à GAP , dans les locaux de Dafy Moto (ZA de Tokoro). Ce rapprochement géographique marque le début d'une collaboration avec l'équipementier moto, dont on voit les effets à Sisteron.

Principalement orientée vers le commerce et la réparation de motocycles, elle aura tenté un peu de gestion d'installations sportives, notamment via un circuit aujourd'hui fermé Elle exploite actuellement cinq établissements en activité :

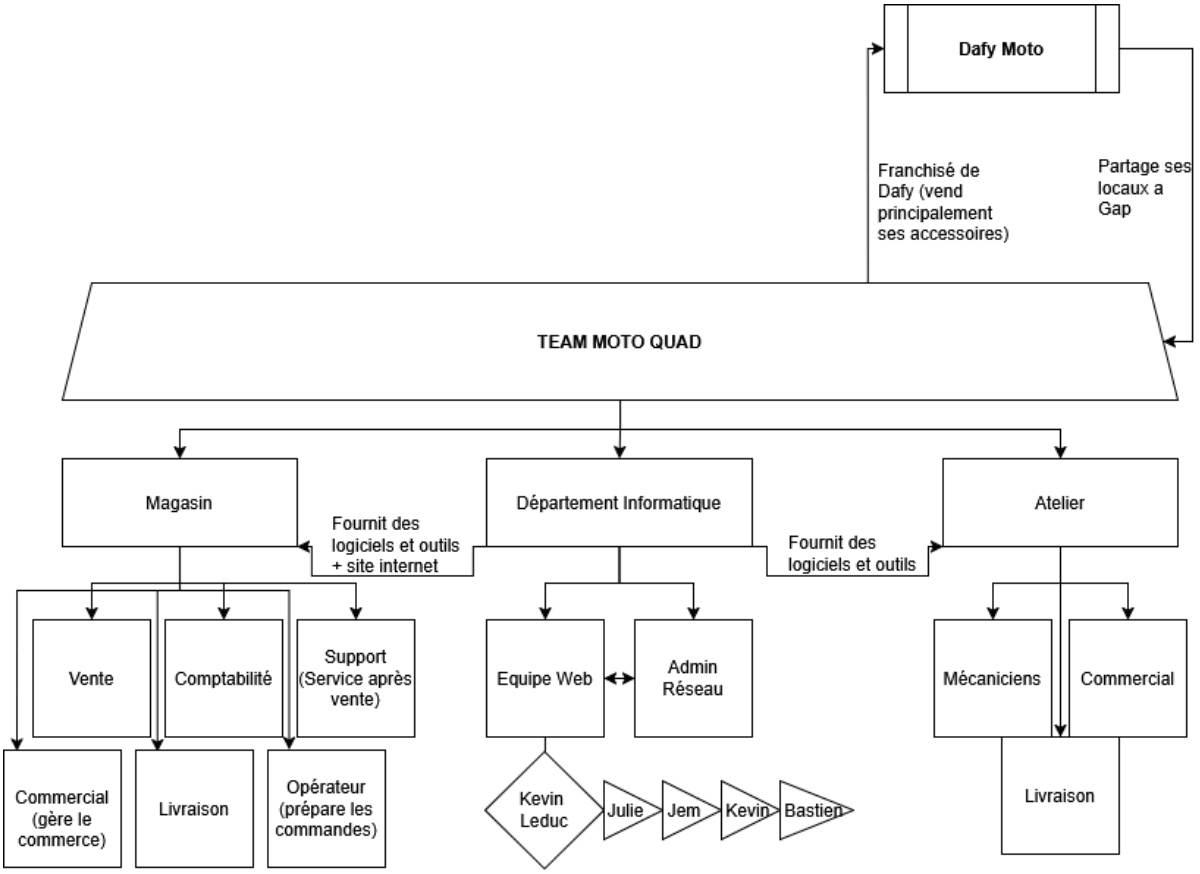
- GAP (siège)
- Sisteron (Esprit Moto Quad, franchisé Dafy Moto depuis juillet 2024)
- Briançon (Ville natale de la société, et de son ancien siège)
- Crèches-sur-Saône (2 sites ouverts en 2019)

Trois anciens établissements ont été fermés entre 2009 et 2024.

En 2022, l'entreprise employait entre 10 et 19 salariés et son capital social fixe s'élève à 400 000 €. Bien qu'elle dépose ses comptes sous confidentialité, son chiffre d'affaires peut être estimé entre 2 et 5 millions d'euros, compte tenu de sa structure multi-sites et de son intégration partielle à un réseau national (Dafy).

Elle n'a fait l'objet d'aucune procédure collective et a procédé entre 2021 et 2022 à des restructurations importantes : changement de siège, augmentation de capital, et évolution de sa gouvernance. TEAM MOTO QUAD opère ainsi aujourd'hui comme un groupe régional structuré dans l'univers du deux-roues, combinant indépendance juridique et affiliation à une marque nationale.

I.2 - Organigramme



Comme on le verra plus tard, l'un des points forts de Team Moto Quad, c'est son organisation en triple point qui la rend virtuellement indépendante.

- Les magasins permettent de vendre des véhicules divers et variés ornés des marques avec lesquelles des contrats ont été signés, ainsi que des produits dérivés et accessoires estampillés DafyMoto.

- Les ateliers permettent d'assurer un contrôle qualité sur les produits vendus, ainsi qu'un assemblage sur place qui permet de diversifier les services proposés tout en fidélisant les clients (Le client ne vient alors pas seulement pour acheter sa bécane, il revient aussi pour la faire réparer, ou s'en faire livrer une autre).

- Le service informatique qui permet non seulement de créer et maintenir un site internet pour les clients, mais aussi de développer des outils accessibles et faciles d'utilisation pour tous les collaborateurs de toute branche. Son intérêt est donc double, car il augmente la compétitivité de l'activité en touchant plus de monde, mais il permet aussi d'avoir une logistique centrale bien huilée sur les commandes et les stocks. Autrement dit, ce service devient un élément clé de la chaîne de valeur de Porter pour TEAM MOTO QUAD. L'entreprise choisit d'y investir et de le maintenir en interne, contrairement à d'autres activités comme certaines livraisons, qui sont externalisées.

I.3 - Secteur d’activité

→ Code NAF : 93.11Z – Gestion d’installations sportives

Secteur temporaire dans lequel l’entreprise s’est engagée de 2013 à 2021, mais c’est bien de noter qu’elle a agit dans ce secteur.

→ Code NAF : 45.40Z – Commerce et réparation de motocycles

Ce secteur regroupe l’ensemble des activités commerciales, de services et de maintenance liés aux véhicules motorisés à deux ou trois roues :

- Vente de véhicules neufs ou d’occasion : motos, scooters, quads, SSV (Side-by-Side), motocross, enduro, etc.
- Atelier de réparation et entretien : révisions, changements de pièces, réparations après accident, montage de pneus, etc.
- Vente d’équipements & accessoires : casques, gants, vestes, bottes, pièces détachées, tuning, électronique embarquée.
- Service client : conseil, financement, assurance, livraison, SAV.

A l'intérieur, des acteurs interagissent pour diversifier l’offre et augmenter la concurrence. En général, ce sont :

- Concessions indépendantes : souvent multi-marques (KTM, Yamaha, Honda, etc.), locales ou régionales, comme TEAM MOTO QUAD.
- Franchisés spécialisés : rattachés à de grandes enseignes (Dafy, Moto Axxe, Maxxess) offrant force d'achat, notoriété et logistique.
- Grands distributeurs nationaux ou plateformes en ligne : tels que Motoblouz, La Bécanerie, ou Louis Moto.
- Constructeurs (intégrés ou non au réseau de distribution) : Honda, Yamaha, BMW Motorrad, etc.

Il est intéressant de replacer TEAM MOTO QUAD dans ce contexte, en comparant son CA à celui de quelques acteurs principaux :

Catégorie INSEE	Enseigne	Type	CA estimé (2023)	Zone d'activité
ETI	**Dafy Moto**	Franchise / Distribution	≈ 79 M€	France (plus de 180 magasins)
	Moto Axxe	Franchise	≈ 75 M€ (0.75M/magasin)	France (100+ magasins)
	Motoblouz (filiale de Bihr)	E-commerce / Retail	≈ 70 M€	Europe
	Maxxess	Franchise / Groupe indépendant	≈ 55 M€	France (60+ centres)
entre PME et ETI	**La Bécanerie**	E-commerce	≈ 21 M€	France & export
PME	**TEAM MOTO QUAD**	Dafy Moto	≈3.5 M€	Département 04 et 05
	MSP Racing (33)	Moto Axxe	~3,5 M€	Bordeaux / Sud-Ouest
	Aix Moto (13)	Indépendant	~2,2 M€	Aix-en-Provence
	BM Motorcycles (38)	Dafy Moto	~1,8 M€	Grenoble
	Pro Racing Service (69)	Maxxess	~2,5 M€	Lyon
	Sud Moto (34)	Indépendant	~3 M€	Montpellier / Béziers
	Moto + (83)	Indépendant / Dafy Moto	~3,8 M€	Var / PACA

On peut ainsi voir que TEAM MOTO QUAD se situe au-dessus des PME, et en dessous des ETI. C'est donc une SAS à plusieurs sites qui domine (ou en tout cas, se positionne bien au-dessus) des franchisés isolés. Son CA élevé viendrait potentiellement de son modèle hybride, entre magasin et atelier, et de ses nombreux contrats avec de nombreuses marques de véhicules. Reste à analyser son pestel pour comprendre quelles sont les tendances macro environnementales qui ont permis son succès à l'échelle régionale.

I.4 - Pestel

PESTEL – TEAM MOTO QUAD	
Facteurs	**Impacts clés pour TEAM MOTO QUAD**
Politique	- Réglementations routières et sécuritaires strictes sur les deux-roues. - Soutiens publics limités pour ce type de transport. - Tensions réglementaires sur les quads et motocross en zone naturelle.
Économique	- Cycle de vente saisonnier et sensible aux crises (Covid, inflation, pouvoir d'achat). - Hausse des coûts logistiques, dépendance à l'import (Asie, Europe). - Développement de l'occasion et de la location comme alternative.
Socioculturel	- Passion moto intacte, mais vieillissement du cœur de cible (40-60 ans). - Sensibilité croissante à la sécurité, au confort et au style. - Montée en puissance des femmes motardes et du tourisme moto.
Technologique	- Émergence de la moto électrique (Silence, Zero Motorcycles), encore marginale mais en croissance. - Digitalisation de l'achat et du SAV (réservations en ligne, comparateurs, diagnostics connectés).
Environnemental	- Pressions sur les nuisances sonores et émissions (zones ZFE). - Quads et cross parfois vus comme polluants ou intrusifs (accès à la nature). - Besoin de s'adapter à la transition écologique (gamme électrique, éco-entretien).
Légal	- Normes CE & homologation (route/circuit) - Obligations sur les équipements de sécurité (casques, gants certifiés). - Encadrement du commerce en ligne, RGPD, droit de la consommation.

I.5 - Historique

(source)

2003 – Création et début d'activité	
16 mars 2003	Création de l'entreprise TEAM MOTO QUAD, sous la forme juridique **SAS**.
24 mars 2003	Immatriculation au Registre National des Entreprises (INPI).
01 mai 2003	Début officiel d'activité.
Première implantation : à **Briançon** (SIRET 00010), sous l'enseigne BRIANÇON MOTOCULTURE, avec une activité déclarée de **commerce et réparation de motocycles (45.40Z)**.	

Notes :

- *SAS (Société à Actions Simplifiés) est une société qui utilise les actions pour se développer (a la place des SARL qui utilisent les parts sociales pour se stabiliser). Les législations à leur sujet sont laxistes pour permettre une facilité de création et de maintien de la pérennité.*
- *Encore aujourd'hui, leur taille économique (<250 employés,<50Me de CA) les classe dans la catégorie PME (Petites et moyennes entreprises).*
- *Comme indiqué plus haut, leur première implémentation de siège social est à Briançon. De nos jours, leur siège social est à Gap.*

2009 – Changement stratégique et géographique	
01 octobre 2009	**Fermeture de l'établissement originel de Briançon** (00010). **Ouverture d'un nouvel établissement à Briançon** (SIRET 00028)

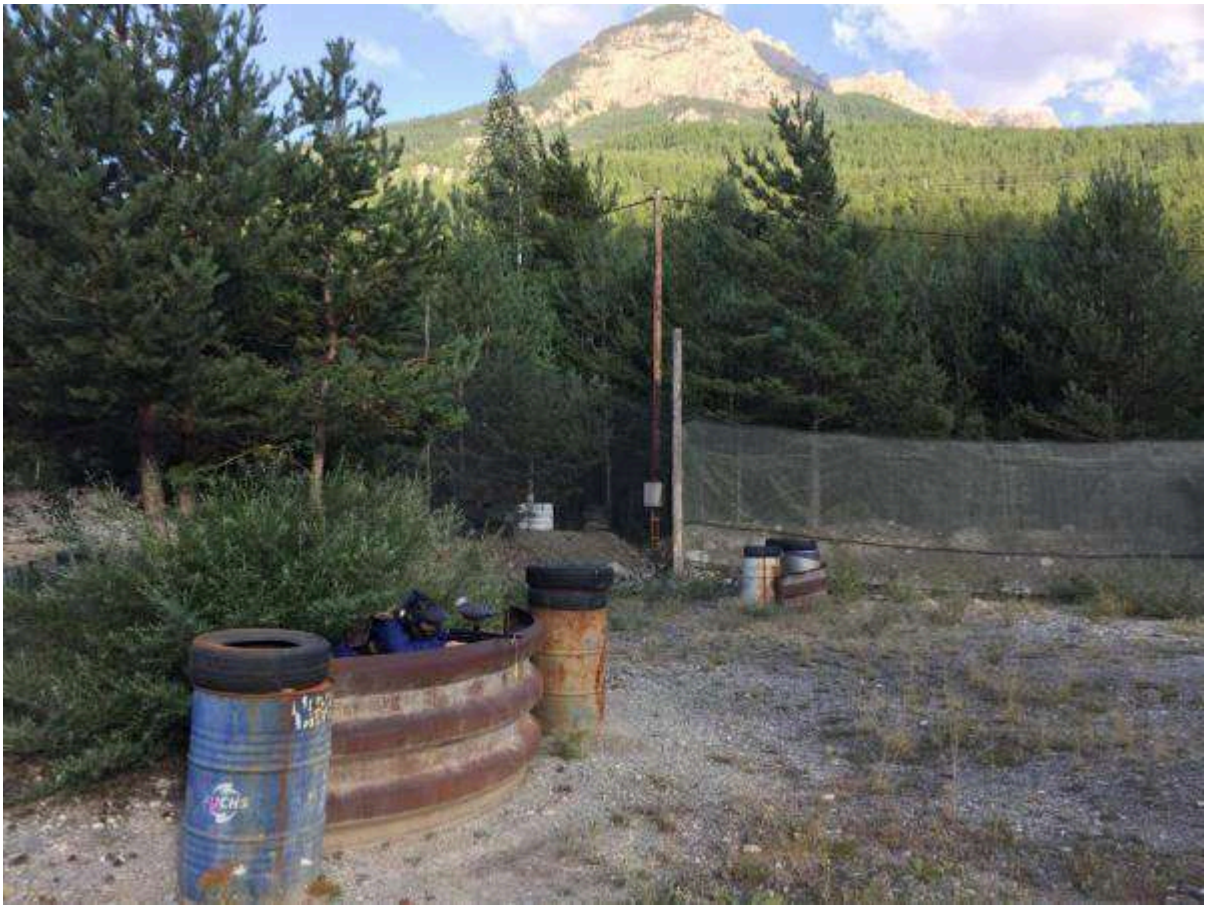
Notes :

- *La relocation de l'établissement originel s'est faite dans le centre d'activités au pied de Briançon. C'est donc un simple transfert d'activité / relocalisation plus stratégique.*

2013 – Diversification dans le sport motorisé	
24 avril 2013	Ouverture d'un établissement à **Saint-Martin-de-Queyrières**, sous le nom *LE CIRCUIT* (SIRET 00036), avec une **activité différente : la gestion d'installations sportives (93.11Z)**.

Notes :

- *LE CIRCUIT restera la seule tentative de TEAM MOTO QUAD de diversifier son activité vers le divertissement motorisé (circuits, courses, etc). IL est cependant bon de noter que, malgré la proximité entre Saint-MArtin-De-Queyrières et Briançon, cela reste un début d'expansion géographique : la sas commence à sortir de sa ville natale*



"Nous vous proposons dans un cadre un peu en retrait un espace composé de 3 circuits de quad (enfants, adolescents et adultes) et de 5000m² de terrain de Paint Ball divisé en deux secteur , l'un contexte forestier et l'autre plutot urbain et dégagé. Les quads utilisés vont de 50, 80 et 250 cm3 selon l'age et les capacités de chacun. Intransigeant sur les règles de sécurités les quads sont controlés tous les jours. En ce qui concerne les Paint Ball , un masque , plastron et combinaison vous sont fournis vous pouvez prévoir des gants et des bonne chaussures selon le temps un tenue plutôt légère…"

-extrait de ce site concernant LE CIRCUIT

2017 – Développement commercial

07 septembre 2017	Ouverture de l'établissement *BESTLINE MOTO* à **Peipin** (SIRET 00044), toujours dans le commerce/réparation de motocycles.

Notes :

- *Peipin est une commune au sud de Sisteron. Ce qui veut dire que TEAM MOTO QUAD s'étend maintenant de l'extremitée orientale à occidentale du département (voir plus : Peipin se situe dans le 04. On parle déjà d'une expansion régionale, mais trans-départementale). Sachant que cela fait, a ce stade, 14 ans que l'entreprise est née, on peut qualifier l'expansion géographique de "lente".*

2019 – Expansion vers la Bourgogne	
17 juin 2019	Double ouverture à **Crèches-sur-Saône** (SIRETs 00051 et 00069), dans un lotissement artisanal.

Notes :

- *Crêches-sur-Saône (noté Mâcon en interne car c'est juste à côté) se situe au-dessus de Lyon, en Bourgogne. C'est très loin du siège social, on parle d'une expansion hors PACA, dans une autre région. Si on devait faire une courte analyse, on pourrait dire que la société a pris du temps pour bien s'installer dans son département d'origine et s'adapter aux défis logistiques d'avoir plusieurs boutiques à gérer. Puis, une fois que les bases étaient suffisamment solides, a augmenté sa vitesse d'expansion.*
- *On parle d'une double ouverture, car les deux sites ont été ouverts en même temps pour profiter de leur synergie. On a ainsi un magasin et un atelier, un peu comme ce qui se trouve aujourd'hui à Gap, qui se complètent et évite aux motos du showroome de Crêche-sur-saône de devoir faire faire un aller retour aux ateliers de Briançon pour chaque réparation.*

Atelier *Magasin*

2020 – Nouveau siège social

30 novembre 2020	Transfert du **siège social** à GAP (SIRET 00077), au sein de la zone artisanale **Tokoro**, chez **Dafy Moto**.

Notes :

- *TEAM MOTO QUAD n'est pas devenu une filière de Dafy Moto, il n'est pas indiqué ici une quelconque affiliation au-delà de la simple localisation. En clair, TEAM MOTO QUAD n'est que simple colocataire de Dafy Moto . Ils occupent tous deux les locaux de Gap.*

2021 – Fermeture du circuit sportif	
01 janvier 2021	Fermeture de l'établissement *LE CIRCUIT* (SIRET 00036) à Saint-Martin-de-Queyrières.

Notes :

- *C'est la fin de la diversification de TEAM MOTO QUAD. A partir de maintenant, c'est vente, réparation, et équipement moto/quad.*

2021–2022 – Restructuration de la gouvernance et du capital
Plusieurs annonces BODACC témoignent de **changements administratifs majeurs** : ● **Changement de siège** (confirmé) ● **Modification de l'administration** ● **Modification de la forme juridique et du capital**

Notes :

- *Le BODACC (Bulletin officiel des annonces civiles et commerciales) est un genre de journal dans lequel sont publiés tous les changements juridiques, légaux et autres de toutes les entreprises, ceci afin d'avertir tout le monde des changements qui ont eu lieu. Ça peut être la création d'une entreprise, un dépôt de bilan, une procédure collective, une restructuration, et plus encore.*
- *Le temps entre le changement du siège social (30 novembre 2020) et son annonce dans le BODACC est due au fait que ce dernier est publié à certaines dates, et qu'il faut donc attendre la publication du prochain avant de le voir apparaître en tant qu'annonce.*

2024 – Ouverture à Sisteron

01 juillet 2024	• Ouverture d'un **nouvel établissement à Sisteron** (*Esprit Moto Quad*), sous SIRET 00085. • Fermeture simultanée du point de vente de **Peipin** (SIRET 00044).

Notes :

- *L'ouverture/fermeture est semblable à celle qui a eu lieu avec l'ancien siège social de Briançon : on à juste déplacé le magasin de Peipin à Sisteron.*
- *Le magasin de Sisteron est une filière de Dafy Moto. Dans un cas normal type restauration, il faudra que le magasin change son identité, sa logistique, et toute sa culture pour celle de la filière mère, ceci a des fins de conformisme. Mais dans le cas des concessions, c'est différent : Le magasin garde le nom de TEAM MOTO QUAD, garde ses contrats exclusifs avec d'autres marques, et peut utiliser sa logistique. Cependant, pour ce qui est des accessoires, il ne peut vendre que ceux de la marque Dafy Moto.*

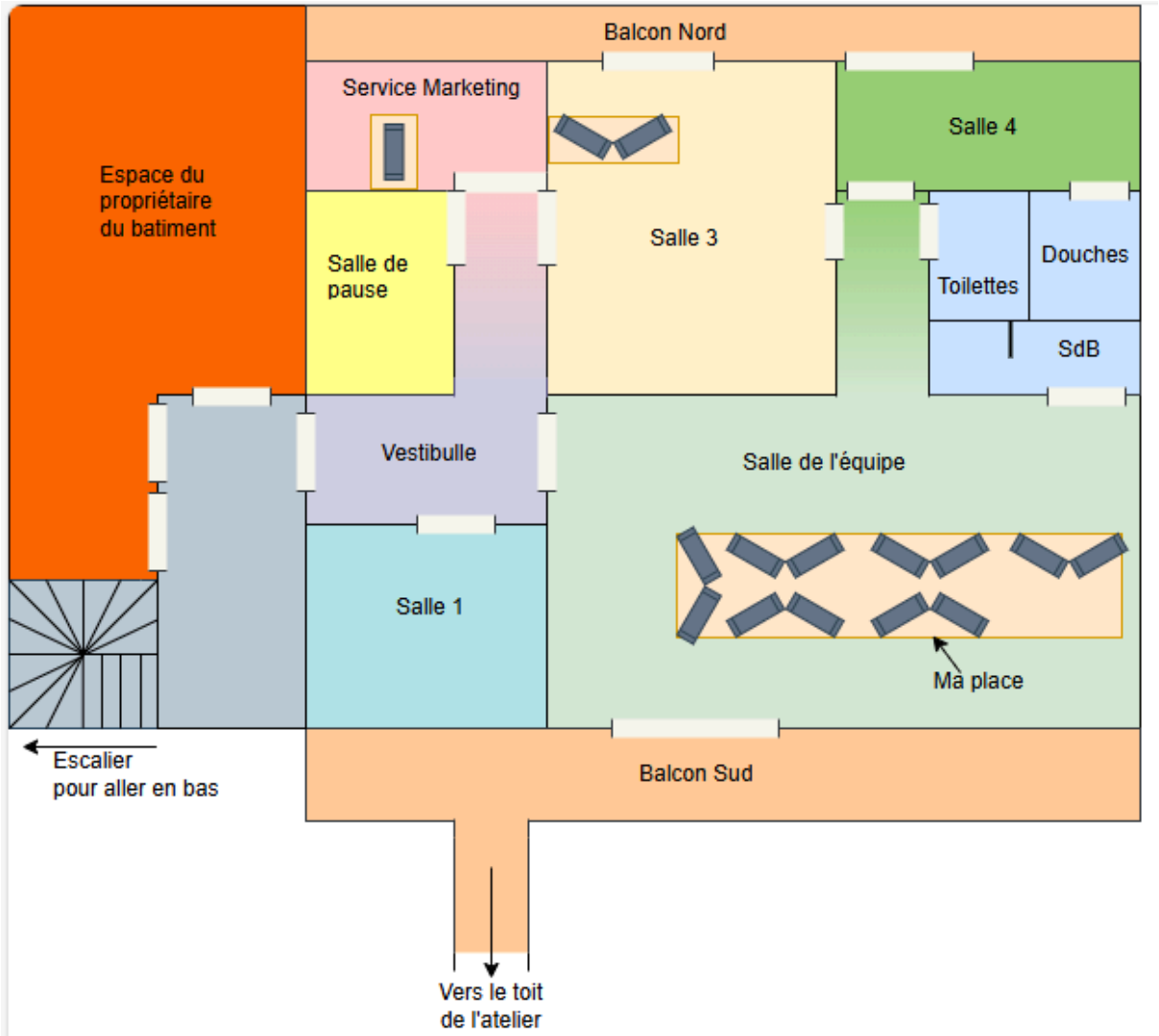
"Installée depuis 2017 dans la zone d'activités de Champarlau, à Peipin, la concession Esprit Moto Quad, franchisée Dafy Moto, poursuit son développement. L'enseigne vient d'inaugurer ses nouveaux locaux dans le parc d'activités Val Durance à Sisteron. « Nous étions installés dans des locaux d'une centaine de mètres carrés, mais ils ne répondaient pas aux exigences du groupe. Nous avons donc décidé de construire un nouveau magasin qui convenait davantage à nos besoins », détaille Jérôme Escallier, co-gérant de la structure lors de l'inauguration officielle, qui a eu lieu le 13 septembre 2024.

« Nous avons lancé la construction en janvier 2024. Le chantier est allé très vite. Il a été livré mi-juillet. Désormais, nous disposons de 1 000 m2 sur deux étages », explique-t-il. Une surface commerciale multipliée par dix qui permet aujourd'hui d'abriter la concession Can-Am Motorcycles et Triumph Motorcycles Limited, mais aussi une boutique d'accessoires Dafy Moto et un atelier d'entretien et de réparation multi-marques."

- Tiré de ce site, a propos de l'ouverture du nouveau magasin

I.6 - Environnement de travail

Les locaux se trouvent au-dessus du magasin, dans un appartement rénové auquel on accède par une porte à gauche de l'atelier. Il y a 2 étages à monter via un escalier en colimaçon, et on arrive sur le palier où se situe l'ensemble de la branche.



La salle de pause contient un frigo, une machine à café et un évier. Cela m'aura été utile pour laisser ma nourriture du midi au frais, ou juste pour boire quelque chose pendant les pauses.
Le Balcon Sud donne sur le toit de l'atelier, sur lequel se trouve divers véhicules, amenés là au moyen d'un ascenseur spécialisé à droite de l'atelier. Le toit sert ainsi de stockage pour les motos et quads. Ces derniers sont à l'abri des intempéries grâce à des toits construits au-dessus. C'est un espace dont on peut profiter pour se dégourdir les jambes ou manger, avec une belle vue sur Charance.
La salle 3 et celle du service marketing ont toujours quelques personnes qui y travaillent, tandis que la salle 1 est rarement occupée, et que la salle 4 est toujours vide.
Les balcons sont utilisés à part égale par toutes les personnes qui travaillent sur le palier.
En dehors de cet environnement, il est arrivé que je doive aller dans un magasin de l'autre côté de la rue, pour parler avec les vendeurs et ceux qui utilisent les outils que j'ai développés. Cela permet de récupérer du feedback, ou de régler des problèmes.

Cependant, pour les démonstrations des nouveaux outils, cela s'est toujours fait au niveau du bureau, dans un environnement local (Dans ce cas, c'est les vendeurs qui se déplacent). C'est d'ailleurs au niveau de ce magasin que se trouvent, derrières, d'autres espaces de travail. L'espace comptabilité par exemple , qui se trouve au deuxième étage, ou l'espace SAV , qui se trouve au fond du magasin (là ou des collègues peuvent nous faire remonter des demandes et réclamations des usagers des sites internets, externes à la société.)
Il y a aussi une baie de chargement pour tout ce qui est livraison, et qui ouvre sur un espace de préparation des commandes .
Pour tous ces espaces-là, on trouve des gens qui utilisent les outils que l'on développe de l'autre côté de la rue.
Par exemple, l'outil de récupération des stocks des différentes bases de données sur lequel j'ai travaillé sera utilisé par les vendeurs et la comptabilité, tandis que l'outil de gestion de commandes (remplaçant un vieillissant et chaotique excel) est utilisé à la fois par les vendeurs et les travailleurs de chez dafy.
Enfin, pour sécuriser l'accès au service informatique et éviter que du personnel non qualifié aie accès à notre infrastructure physique, la porte qui mène sur l'escalier est fermée par un code.

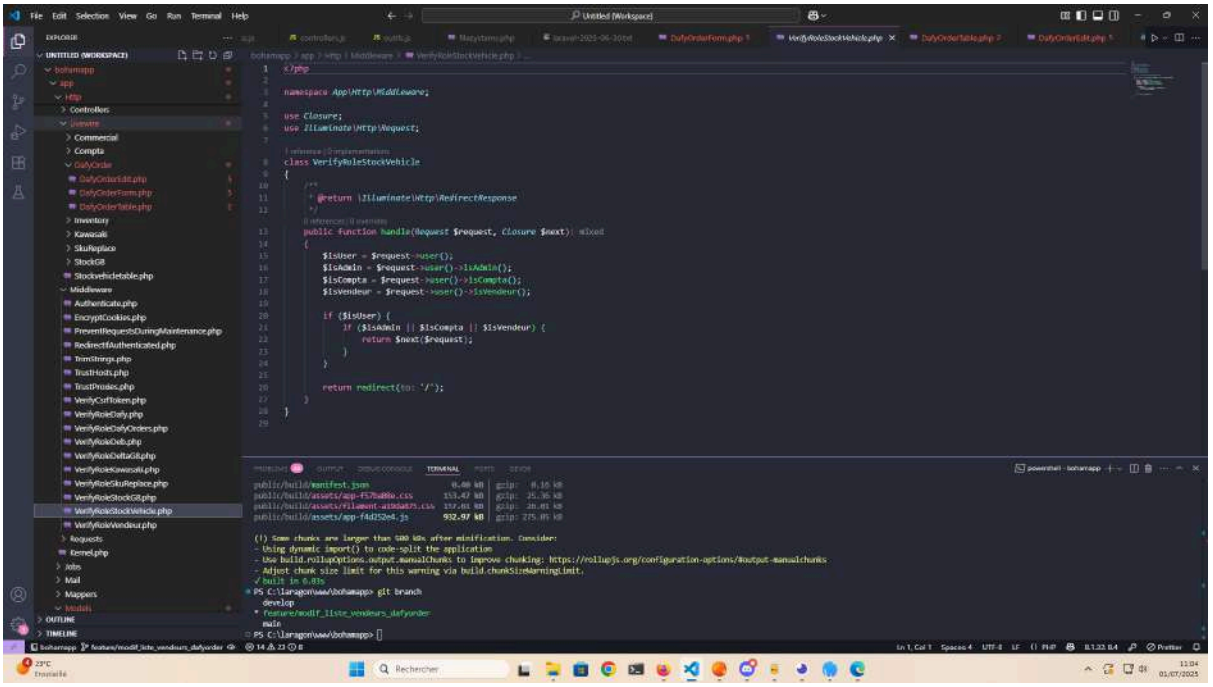
Pour ce qui est des conditions de travail, mon tuteur permet une certaine marge pour le retard, jusqu'à 10 minutes.
De plus, les pauses, leur longueur, leur fréquence et le moment où elles commencent sont laissées à la discrétion de l'employé. Tant qu'elles restent raisonnables, on peut prendre un café et se balader sur le toit pendant 20 minutes ou ne pas prendre de pause du tout, prendre l'air après une longue session de débogage ou rester sur son ordinateur pour regarder une quelconque vidéo entre deux long tests d'api.
Pour ce qui est de la pause du midi, j'ai aussi eu une grande liberté. De base, elle s'effectue de 12h30 à 14h. Mais on peut la finir X minutes plus tôt pour partir X minutes plus tôt. On est sur un style de management plutôt consultatif, où les employés font de leur plein gré et c'est le chef qui tranche lorsque ça ne lui convient pas.
Il n'y a d'ailleurs pas que les pauses qui sont libres : les capsules à café sont mises à disposition par mon tuteur, ainsi qu'un jour de canicule ou nous avons eu droit à des glaces.
Je suis très fan de ces conditions de travail basées sur le respect et la bonne foi, car cela donne naissances à des dynamiques de travail qui fonctionnent. Par ceci, j'entend que les employés n'ont pas peur de donner leur avis ou de rechercher de l'aide auprès de l'employeur, ce qui augmente la productivité et l'efficacité.
On voit aussi des actes désintéressés et volontaires qui apparaissent de temps à autre, comme un employé qui va prendre du temps pour aller aider un autre, un coup de ménage qui est donné volontairement tous les vendredis soir, ou encore le fait que les heures ne sont pas absolues. On va alors observer des employés qui restent un peu plus tard pour travailler ou finir une feature, ou même parler de travail.
En clair : on ne vient plus pour travailler, mais pour créer/améliorer quelque chose . Il n'y a même pas de notion de 'fidélité à l'entreprise', on reste avec plaisir et on travaille avec passion pour la beauté du métier.

Pour ma part, le fait que je sois autorisé à écouter de la musique pendant que je travaille, ou que je puisse regarder des vidéos entre deux implémentations qui me demandent beaucoup de concentration afin de me reposer un peu , sans avoir à regarder par dessus mon épaule

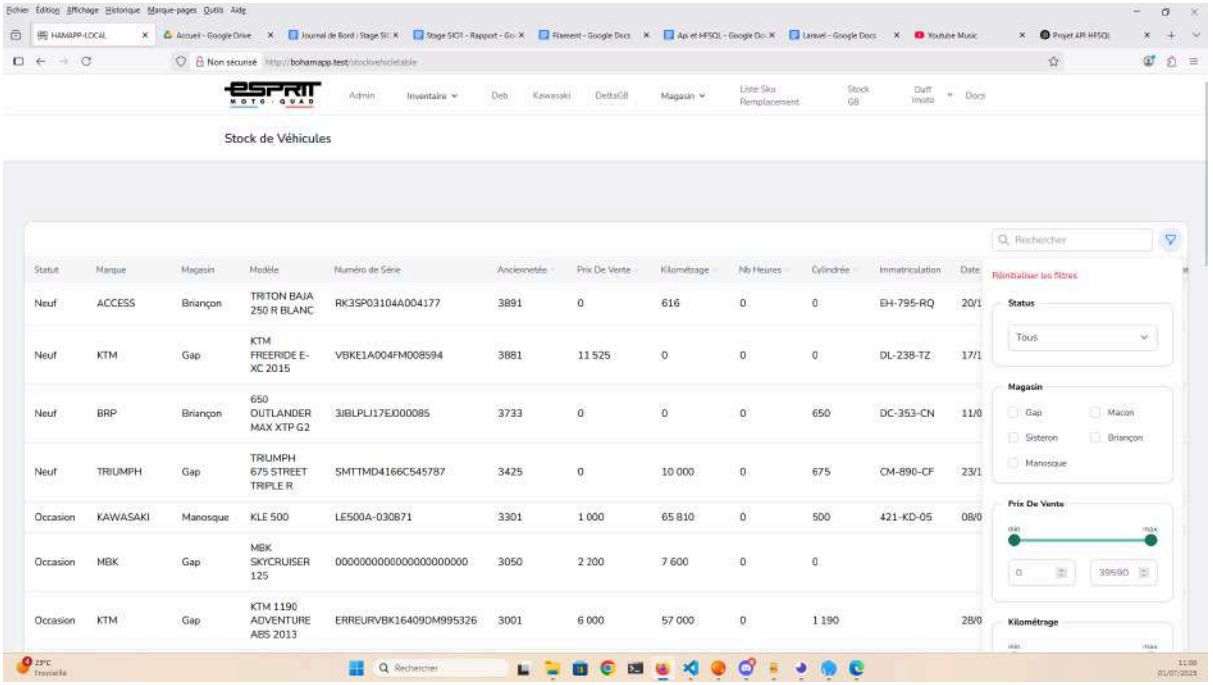
ou me dire que ‘me reposer pendant les heures de travail est une mauvaise chose’, ce sont des privilèges qui ont grandement participé à mon efficacité.
De plus, le fait que l’on puisse aussi autoriser l’utilisation d’ia dans le raisonnement ou la construction de features (tant que l’on peut expliquer ce que l’on a implémenté) a aussi participé a mon efficacité.
En clair : on est sur un environnement qui est idéal pour la création de systèmes informatiques et d’outils car la santée mentale (pauses et musiques quand on veut, café et glaces, convivialité) et physique (toit pour aller bouger) est prise en compte, et que l’on ne limite pas les outils et paradigmes de code à une liste arbitraire de technologie.
A noter aussi : une réunion tous les lundis matin pour faire le point sur ce que les collègues ont fait.

Pour ce qui est de l’environnement de travail au niveau outils et technologies utilisées, on m’a mis sur un ordinateur qui tourne sous windows, et équipé de deux écrans. Sur l’écran de gauche, j’ai mis mon vscode afin de coder. Sur l’écran de droite, j’ai mis mon navigateur internet qui contient une vue sur les outils développés en local (HAMAPP-LOCAL), mon drive ainsi que tout les google docs créés dans le cadre du stag et qui pourraient m’aider pour certaines implémentations (pour l’instant, le journal de bord pour noter mes progrès au fur et à mesure, mon rapport de stage que j’avance quand j’ai rien a faire, un google doc de notes que j’ai pris sur filaments, les API et HFSQL, et Laravel), un ongle youtube pour écouter de la musique pendant que je travaille ou regarder des vidéos quand je sens que ma concentration échoue, et un onglet chatgpt pour poser toute mes questions.

Ecran Gauche : Vscode

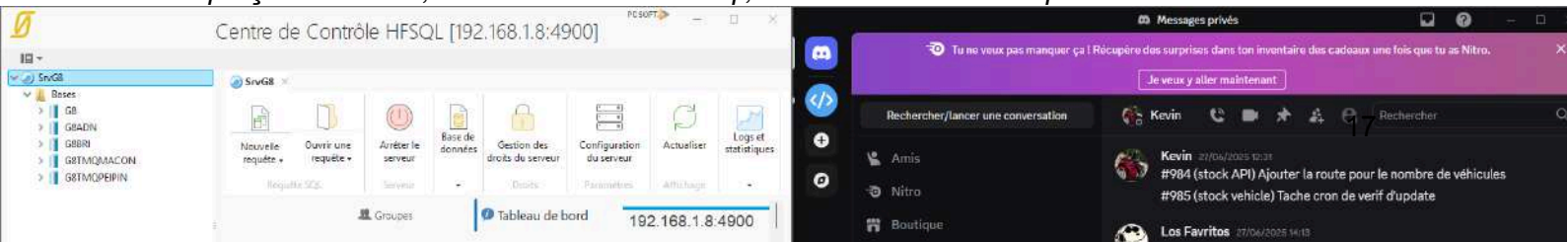


Ecran Droite : navigateur avec vue sur l’application développée, notes google doc sur les technos utilisés, youtube pour musique/pause et chatgpt pour conseils



Ceci étant dit, je n'utilise pas que vscode et un onglet pour voir ce que je développe. En effet, on utilise de nombreuses technologies supplémentaires pour travailler plus efficacement. Par exemple, j'utilise Discord pour communiquer avec mon tuteur et avec l'équipe, car cela permet de laisser une trace écrite des conversation et points abordés, tout en permettant à mon tuteur de pouvoir répondre à tout le monde tout en gardant du temps pour ses travaux. J'utilise aussi insomnia pour tester mes appels api et voir comment les utiliser ainsi que le type de réponse qu'ils peuvent me renvoyer. Au niveau des bases de données, je possède antares SQL qui me permet de me connecter à une base de données en local, de la parcourir, et de changer ce que je veux changer. C'est particulièrement pratique pour changer le rôle du user avec lequel je suis connecté afin de tester des fonctionnalités, sans avoir à créer de nouveaux user ou de changer manuellement. J'utilise aussi le centre de contrôle HFSQL pour pouvoir communiquer nativement avec la base de données G8 qui contient les véhicules. Sachant que cette base est en hfsql et que les tables sont donc des fichiers, je ne peux pas parcourir directement la base. Le centre de données permet ainsi d'aller me connecter à distance a la bonne base de donnée, et surtout de voir comment chaque champ est appelé afin de récupérer les données dont j'ai besoin (ce qui est une nécessitée, car les personnes qui ont crées la base ont donné des noms alambiqués à leurs champs, tels que SER_CHA01 pour le numéro de série).

Aperçu de discord, Centre de contrôle hfsql, insomnia et antarès sql



Ce n’est pas tout : j’utilise laragon afin de développer (c’est la dessus que fonctionne l’application). Laragon offre un stack similaire au lamp, mais sur Windows. On a ainsi tout ce qui faut pour faire tourner une application en local ou en ligne (tel que laravel,apache), ainsi que des outils qui rendent l’affichage et sa bonification facile (tel que tailwind, bootstrap, heroicon, ou encore livewire et filament pour les panels admin). A noter que pour ce genre de technologie, c’est assez difficile de faire un petit projet, mais facile de maintenir un grand projet. Par exemple, pour simplement aller sur une page, il faut que le lien retourne une route, qu’il faut configurer pour qu’elle envoie sur un controlleur, qu’il faut configurer pour qu’il envoie sur blade, sur laquelle on va afficher notre page avec les facilitées de blade, et potentiellement un livewire (livewire qu’il faut créer, configurer, relier à la blade, qui envoie sur un contrôleur du livewire, qui envoie sur une blade spéciale du livewire). Chaque petit changement est une matriochka qui nécessite une myriade d’étapes avant d’implémenter. Mais à la fin, on a un système qui marche, qui est maintenu, et où chaque bout de code est dans un endroit spécifique qui lui correspond. En clair : tout est organisé. Et avec git et le workflow git, on peut travailler en conjonction avec les collègues de l’équipe (son fonctionnement est expliqué en annexe).

Quelques outils de laragon

Composant	Outil par défaut	Détails
Serveur Web	Apache ou Nginx	Tu peux choisir au démarrage.
Serveur PHP	PHP (multi-versions)	PHP 5.x à 8.x faciles à basculer.
Base de données	MySQL / MariaDB	Admin via HeidiSQL ou phpMyAdmin.
Serveur mail local	MailHog ou smtp4dev	Pour tester l'envoi de mails.
Terminal / shell	Cmder intégré	Puissant, personnalisable.
Node.js / npm	(optionnel)	Pour frontend, build tools, etc.
Composer	Oui	Déjà installé.

Tous ces outils nous permettent de développer en local toutes les features que l'on souhaite dans un environnement proche de celui de déploiement, de les tester en un clin d'œil, puis de les mettre sur un dépôt distant pour un déploiement futur sur le cloud. L'architecture sur ce cloud est un peu complexe (une partie de son explication se trouve en annexe), mais pour résumer rapidement, c'est une toile complexe reliant plusieurs machines virtuelles entre elles, machines qui ont chacune un but bien à elles. Il y a un accès contrôlé au site web par les utilisateurs, et un autre accès par nos dépôts distants . Enfin, tout ceci se situe dans un cloud Microsoft Azure Cloud, qui est un environnement sous linux.

Cet environnement de travail me permet ainsi de coder dans une ambiance agréable, de tester rapidement mon code comme si il était sur le cloud, puis quand je suis satisfait, de mettre mon travail sur le dépôt distant avant qu'il soit envoyé sur le cloud pour être vu/utilisé par tout le monde.

II - Missions et tâches effectuées
II.1 - Présentation du tuteur

À propos

De formation universitaire, je suis passionné par le développement logiciel. J'ai actuellement plus de 7 ans d'expérience en systèmes critiques embarqués.
Rigoureux, voire perfectionniste, mon autonomie et ma faculté d'adaptation sont mes forces.

A la recherche d'opportunités dans les hautes-alpes ou en télétravail, j'étudie toute proposition.

https://fr.linkedin.com/in/kevin-leduc-750563b2

II.2 - Les différentes missions

Le projet du stage tient en une seule page. Ce n'est pas une figure de style, on m'a donné un document en .txt qui regroupe mes différentes missions, et il contient 3 étapes majeures :

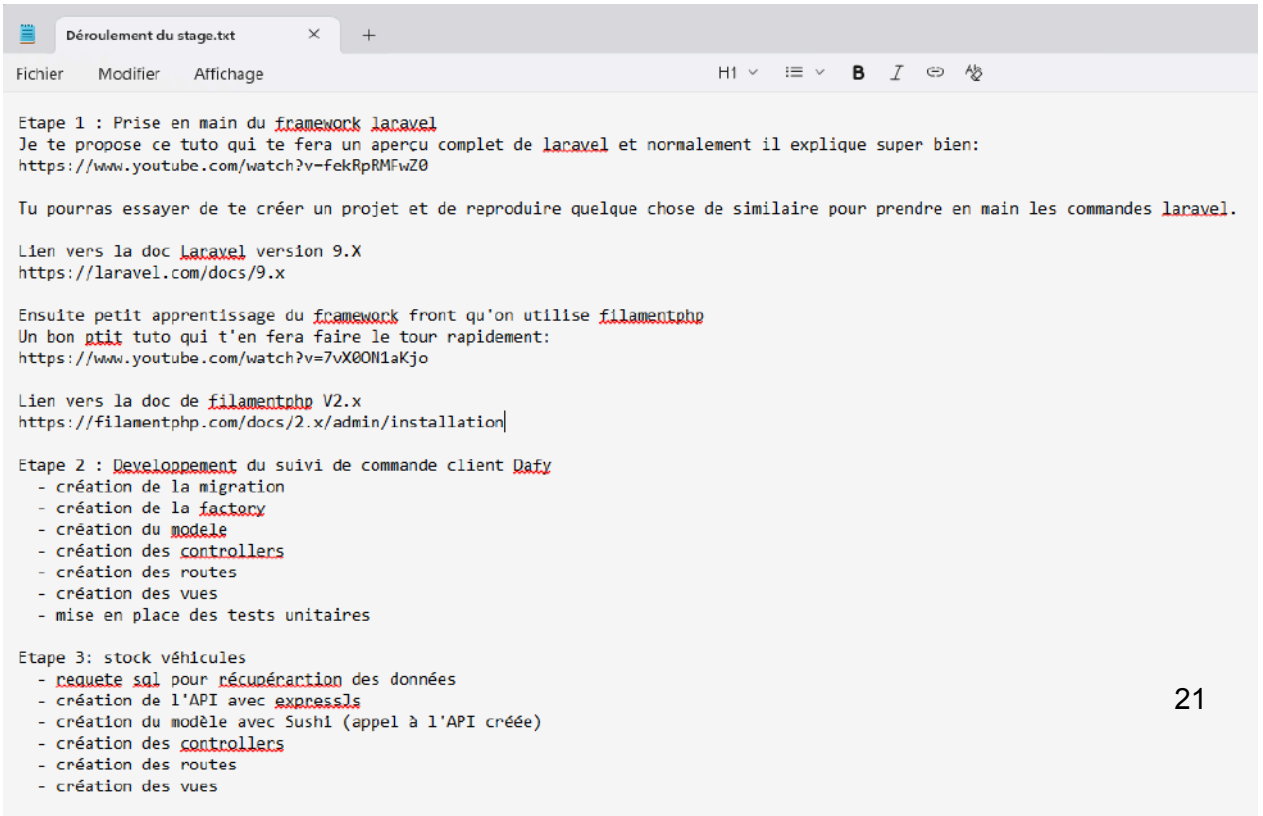
- Prise en main des technologies utilisées. Plus précisément, laravel (le framework php qui donne outil et convention pour aider au maintien d'une structure de projet efficace et soignée) et filament (un autre framework qui fournit des composants réactifs tels que des tables formulaires et autres). J'ai eu aussi besoin de me familiariser avec d'autres concepts, comme les api et les livewires, mais ce sont des missions secondaires qui viennent naturellement s'intégrer au stage au fur à mesure des idées d'implémentation

- Développement d'un outil pour gérer les commandes. DafyOrder; c'est son nom, permet aux rôles autorisés de visualiser d'un coup d'œil les commandes enregistrées. Il permet aussi de gérer l'état des commandes (commandé , reçu, prévenu, récupéré), de supprimer des commandes, et d'éditer/ajouter des commandes via une interface pratique de wizard. Avec cet outil, le but principal était de remplacer l'usage de vieux tableaux excels mals entretenus, de faciliter la rentrée de commandes, de normaliser la manière dont les commandes sont enregistrées, et surtout de permettre au système tout entier d'évoluer vers une utilisation plus globale (par exemple, le rendre compatible avec le système de commandes de Dafy Moto utilisé partout en france)

- Développement d'un outil de visualisation des stocks de véhicules. En effet, les vendeurs devaient se connecter et faire leur recherche de stock sur plusieurs bases différentes, une après l'autre. Le but de cet outil est donc de regrouper cette recherche/visualisation de stock dans une seule table , avec des filtres adaptés.

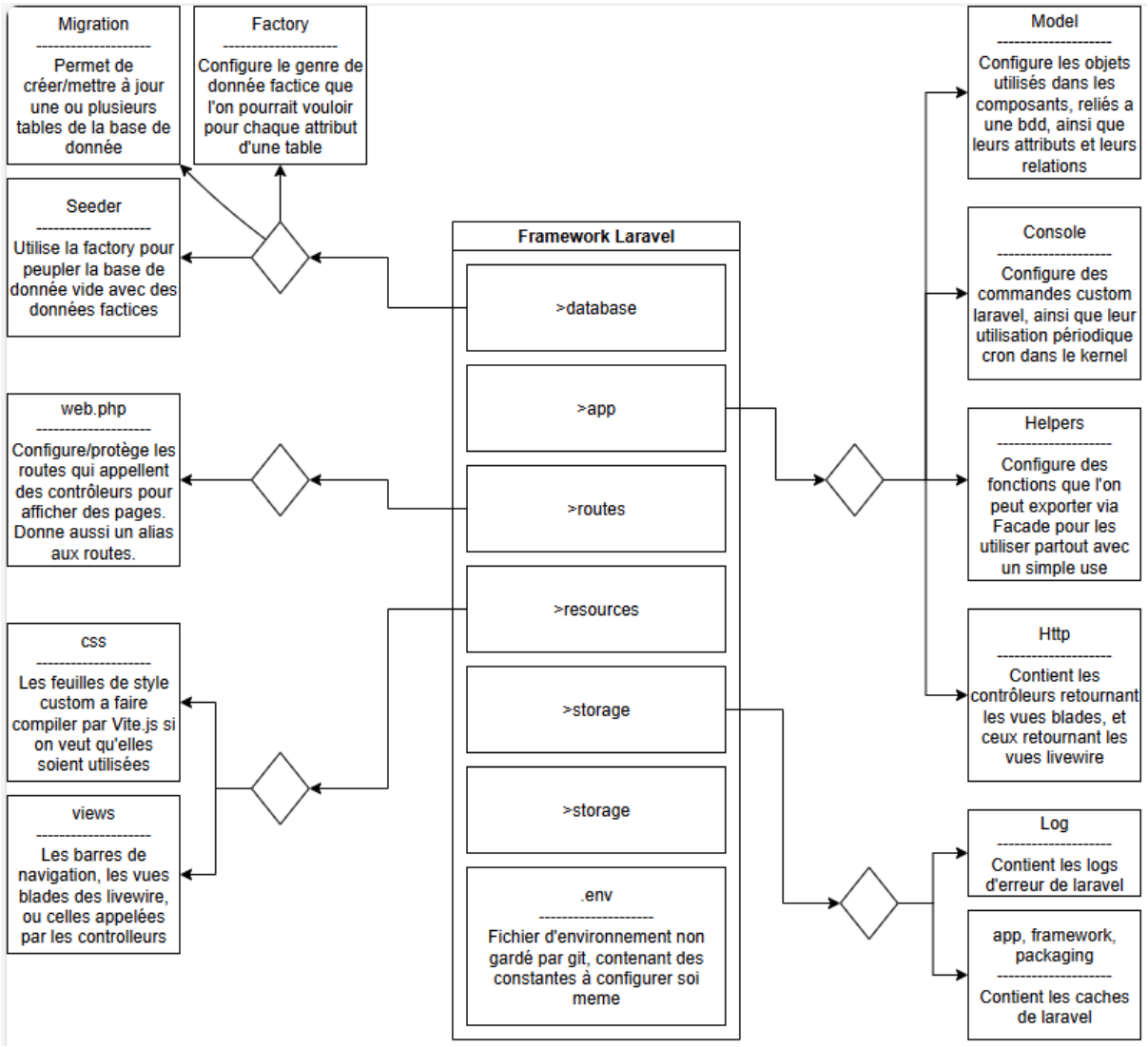
Au vu de la faible quantité de missions abordées, on va toutes les étudier en détail.

Les missions telles qu'elles apparaissent dans mon .txt



II.3 - Etude de cas
II.3.α - Prise en main des technologies

Mon tuteur m'a donné une vidéo youtube qui explique comment monter un petit projet sur laravel, pas à pas. On y voit alors les bases de ce framework, notamment sa manière de gérer les différentes fonctionnalités d'un projet web. Comme expliqué plus haut, pour aller sur une autre page, il faut utiliser une combinaison de 3 fichiers .
Le fichier route, qui va permettre de donner un 'alias' à un contrôleur. De cette manière, si on appelle cet alias dans une autre page, en retournant par exemple route('alias'), on va en réalité appeler un contrôleur spécifique. L'attrait de cette méthode, c'est que l'on peut faire passer en argument de cette route un middleware, qui va permettre de vérifier certaines choses, comme les qualifications du user connecté par exemple. On peut faire passer plusieurs middleware à une même route, afin de vérifier plusieurs choses, et on peut même sécuriser la route de manière à ce qu'on ne puisse pas y accéder si on ne remplies par les conditions du middleware (le user est alors renvoyé à une autre page, souvent la page index ou racine du projet.) De cette façon, même si un petit malin essaye de retrouver une page en tapant directement son url dans la barre de recherche, il tombera sur la page de redirection si ne serait-ce qu'une seule middleware échoue.
Le contrôleur appelé par cette route, c'est une classe que l'on peut rendre invokable, et qui permet de renvoyer la véritable page à afficher. Pourquoi ne pas directement faire en sorte que la route appelle la page à visualiser ? C'est afin de respecter le modèle MVC dont l'influence touche lourdement toutes les couches de ce framework. On sépare ainsi la logique de Vue (qui va servir d'interface à l'utilisateur, prenant ses informations et affichant d'autres informations), de Modele (une sorte de base de donnée, souvent une classe comportant méthodes et attributs de l'objet dont il est question dans la page que l'on veut afficher) et de contrôleur (ce dont on parle, qui va faire le pont entre la vue et le modèle, en récupérant des informations de ces deux endroits , et qui va surtout abriter la logique. C'est là où on va pouvoir utiliser ces informations et en faire des choses utiles). Ici donc, le contrôleur, appelé par la route, va appeler lui-même la vue. De cette manière, on peut faire passer, au niveau du contrôleur, des informations récupérées/envoyées par la route, pour la vue. Ca va être, par exemple, récupérer les informations d'une seule commande que l'on va donner à la vue et qui va pouvoir les afficher.
La vue, enfin, est elle même spéciale. Déjà, laravel et plus particulièrement laragon nous donne de nombreux outils pour afficher de nombreuses choses. Par exemple tailwind qui possède entre autres des sliders, ou heroicon qui possède des belles icônes en svg dont on peut faire changer la couleur avec des palettes prédéfinies dans alpine. Mais le plus important dans la vue, c'est qu'elle est en .blade. Blade, c'est une manière de mélanger php et html ensemble. Dans un projet sans blade, si l'on veut par exemple afficher une variable, il faut ouvrir des balises php et faire un écho sur la variables. Avec blade, on peut simplement mettre le nom de la variable entre doubles crochets, et le tour est joué. De la même manière, mettre une condition if demande d'ouvrir une balise php pour mettre le if,puis de la fermer, et ensuite de la rouvrir pour mettre le crochet de closure du if,avant de refermer la balise php. Avec blade, on peut simplement faire @if et @endif . Une autre chose que blade change, c'est que le nom est différent. En plus d'avoir une extension en .blade.php, on appelle la vue par le nom du dossier dans laquelle elle est, puis le nom de la vue, séparées d'un point. Cette séparation est importante, parce que certains composants tels que le contrôleur dont on a parlé plus tôt, reconnaissent cette convention de nommage .


Migration
Permet de créer/mettre à jour une ou plusieurs tables de la base de donnée
Factory
Configure le genre de donnée factice que l'on pourrait vouloir pour chaque attribut d'une table
Seeder
Utilise la factory pour peupler la base de donnée vide avec des données factices
Framework Laravel
>database
>app
>routes
>resources
>storage
>storage
.env
Fichier d'environnement non gardé par git, contenant des constantes à configurer soi meme
web.php
Configure/protège les routes qui appellent des contrôleurs pour afficher des pages. Donne aussi un alias aux routes.
css
Les feuilles de style custom a faire compiler par Vite.js si on veut qu'elles soient utilisées
views
Les barres de navigation, les vues blades des livewire, ou celles appelées par les controlleurs
Model
Configure les objets utilisés dans les composants, reliés a une bdd, ainsi que leurs attributs et leurs relations
Console
Configure des commandes custom laravel, ainsi que leur utilisation périodique cron dans le kernel
Helpers
Configure des fonctions que l'on peut exporter via Facade pour les utiliser partout avec un simple use
Http
Contient les contrôleurs retournant les vues blades, et ceux retournant les vues livewire
Log
Contient les logs d'erreur de laravel
app, framework, packaging
Contient les caches de laravel

Tout cela ne sert qu'à afficher une page, mais qu'en est il de ce qu'on affiche dans la page ? Laragon nous donne accès à des outils de base de données afin de pouvoir utiliser des objets un peu partout.

Il suffit de faire une migration dans laquelle on va spécifier les champs et types de champs que l'on veut voir dans la bdd, puis de faire une factory pour dire quels genre de données bidon on voudrait voir dans les champ de ce nouvel objet, et enfin d'appeler cette factory dans le seeder. Ce que fait cette combinaison de migration/factory/seeder, c'est créer une table pour une objet, puis remplir cette table avec des objets bidons pour que l'on puisse tester notre code et nos affichage.

Encore une fois, on est dans des technologies qui facilitent le maintien d'un gros projet, mais est plus compliqué pour les projets plus petits. Cette méthode de population de données évite ainsi d'avoir à passer du temps à peupler nous même la base pour tester nos codes avec des données . Maintenant que l'on a une base de donnée et une vue qui va afficher des éléments, il nous suffit simplement de créer un objet dans lequel vont aller les informations stockées dans notre base de donnée.

Cet objet, appelé model, laragon va nous permettre de lui donner des attributs remplissables par un user ou non, des méthodes getter/setter automatique sur ces attributs et des relations avec d'autres objets du modèle. De plus, laragon nous donne un outil spécial pour manipuler cet objet : le modèle Eloquent. Ce modèle est une surcouche d'obfuscation des base de données, qui évite au développeur de faire toute une méthode de connection de base de donnée, requêtage, préparation du requêtage,exécution de la requête, récupération des données,mise en place des données dans le model. A la place, eloquent permet de faire du requêtage facile à base de méthode sur un objet? Par exemple, pour trouver le user avec l'ancienneté la plus élevée, Eloquent nous permet (une fois que le model User est importé dans la blade) de faire User::max('ancienneté') . Et si on veut rajouter une condition ou le rôle doit être dafy, on peut rajouter ->where('role','dafy'). Ce modèle la permet de pouvoir récupérer des informations de la bdd de manière plus simple, rapide, et efficace, tou en simplifiant la lecture du code.

Pour comprendre toutes ces informations, il a fallu que j'étudie le tuto pendant plusieurs jours, tout en suivant les instructions, posant des questions a chatgpt, et marqué les connaissances acquises et validées en théorie et pratique dans un googledoc (disponible en annexe).

Après m'être approprié laravel, j'ai pu un peu tester mes connaissances en faisant des migrations et vue sur une copie du dépôt distant, afin de préparer la mise en place de l'outil de suivi de commande. Une fois fait, cependant, il a fallu organiser les données de la bdd à l'intérieur du model afin de pouvoir les afficher. J'aurais pu construire un simple tableau avec des balise table, mais le stack laragon installé possède un autre outil pour afficher des informations dans une sorte de panel admin : Livewire et Filament. Et comme pour la précédente technologie, j'avais un tuto et une documentation à parcourir. Une chose qu'il est bon de noter dans cette étude de cas, c'est que j'étais laché dans ces technologies avec seulement ces tutos pour me débrouiller. Dans un sens, c'était une sorte d'autoformation, qu'il a été plus simple d'appréhender suite aux 4 mois d'autoformation openclassroom que j'ai fait avant de rentrer en bts SIO .

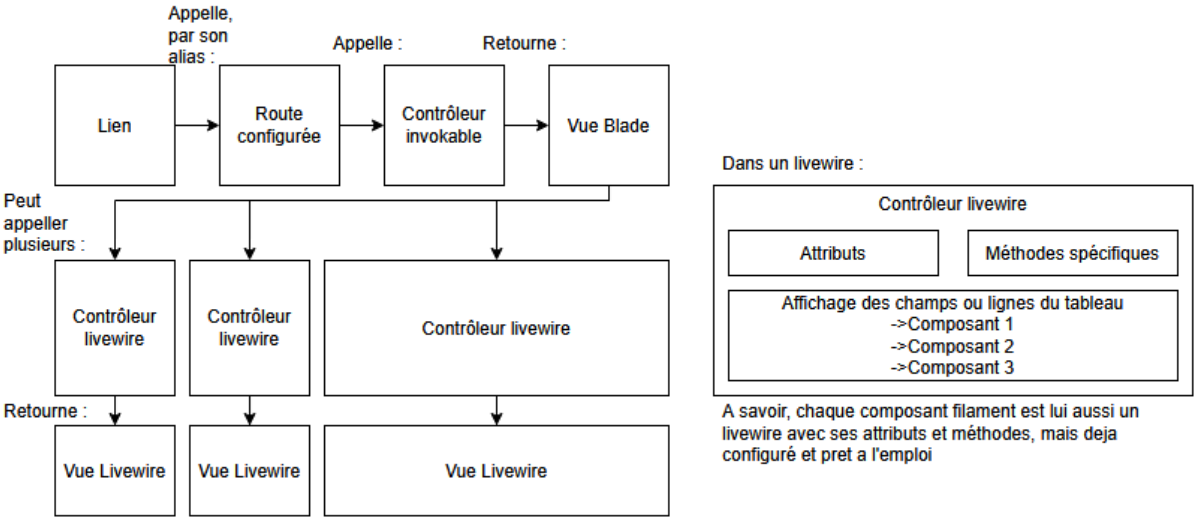
Certe , j’avais la possibilité de demander conseil à mon tuteur , mais son aide n'était jamais prise pour acquis a cause des nombreuses choses qui prennent l'attention d’un chef de projet (les demandes des autres collègues, les bugs, les demande d’évolution ou de changement du sav, les implémentation, etc). A ce sens, la possibilité d’utiliser chatgpt a été plus que bénéfique car cela m’a permis de poser toute les question que je voulais, sans limite, et de préparer des ‘mini-rapports’ a mon tuteur pour les fois ou il venait voir comment ca se passait. Cet équilibre entre tuto, mise en pratique, question chat gpt, prise de note a ma maniere, validation par tuteur m’a alors permit de comprendre ce qu’était filament et les livewire au bout de deux jours.

En Faite, Un livewire est une sorte de fenêtre vers un composant réutilisable et réactif. Il est composé d’un contrôleur livewire dans lequel se passe toute les logiques des données, et d’une vue livewire qui se contente simplement d’afficher ce que retourne le contrôleur. On appelle ainsi un livewire spécifique dans la vue blade, puis on configure le contrôleur pour afficher ce que l’on veut , et on peut formater cet affichage dans la vue livewire. Dans un sens, c’est un peu comme une fonction réutilisable un peu partout, mais dont on se sert comme base avant de modifier ce que l’on veut modifier, et d’afficher ce que l’on veut afficher. Et un composant filament ? Il s'appelle dans le controlleur livewire, et il permet de créer instantanément des choses relatives aux panels administrateurs. Par exemple, dans le cas d’une table qui affiche des utilisateurs, on peut lui demander d’afficher un filament table, et utiliser ses très nombreuses méthodes pour customiser cette table et afficher ce que l’on veut dans les colonnes. Par exemple, on peut demander à ce que la colonne id ne soit visible que si l’utilisateur est un admin. On va alors appeler la méthode ->visible() sur le composant TextColumn::make('user_id') .

A noter que le champ user_id est un champ de l’objet User, et que simplement en mettant le nom du champ en string dans le composant, celui-ci saura quelle valeur utiliser, et de quel objet. Il est bon de noter que filament suit le même paradigme que laravel dans sa structure : tout est à sa place. SI l’on veut que chaque page de résultat puisse être directement recherchée par exemple, on surcharge la méthode isTablePaginationEnabled pour lui faire retourner true . A ce sens, customiser la table devient aussi simple que de trouver la méthode à surcharger, et lui faire retourner la bonne valeur.

Cette philosophie s'applique aussi aux composants utilisés lors de l’affichage : chaque composant, d’une simple colonne de texte au checkbox, d’un filtre customisé à une action, possède de très nombreuses méthodes que l’on peut surcharger pour changer des choses dans leur affichage. On peut même utiliser des fonctions à l'intérieur des méthodes afin de rendre un affichage conditionnel par exemple. FIlament permet ainsi de faire des actions (type supprimer ou ajouter), des tables, des filtres, ou même des formulaires, puis il permet de structurer l’affichage ou d’ajouter de l’interactivité à ces composants. Certaines méthodes permettent même d’interagir directement avec la bdd. Par exemple, si l’on crée un composant toggable (un genre de switch ouvert/fermé) sur un champ booléen, cliquer sur le switch pour le faire passer sur on va faire passer la valeur correspondante à true dans la base de donnée.

C’est ce que j’entend par couche d'obfuscation : les commandes lourdes à utiliser et complexes sont remplacées par des lignes de codes simples d’utilisations, et qui sont compatibles/directement utilisables dans leur contexte, sans avoir à le préciser. Un peu comme des briques de légo, filament rend le front “trop facile” avec ces composants qui s’imbriquent les uns dans les autres et sont très fonctionnels sans même qu’on le spécifie. Cependant, il est bon de noter que la version filament utilisée dans le projet est la version 2.0, et la version de laravel utilisée est 9.0. Vu que ce ne sont pas les versions les plus récentes, certaines méthodes de la doc ne marchent pas, et certaines fonctionnalités ont dû être codées à la main, ou surchargées depuis d’autres méthodes. Si ces différences ne paraissent pas importantes au premier coup d'œil, vous allez comprendre dans les pages qui suivent, les obstacles et problèmes rencontrés qui ont dû être court-circuités en dur pour fonctionner sur des versions antérieures, et que l’on aurait pas eues si les techno utilisées étaient à jour.



II.3.β - Outil de suivi de commandes DAFY ORDER

Comme expliqué précédemment, le but de cet outil est de remplacer l'utilisation d'un tableau excel dans lequel était marqué toute les commandes. Ce tableau se caractérise par plusieurs fonctionnalités, qu'il a fallu implémenter dans l'outil, petit à petit.

La première étape de cet outil, c'est le tableau lui-même. Il possède plusieurs champs, tel que la date de création, le vendeur, le client, si la commande vient d'un autre magasin, de quel magasin elle vient, la marque, le produit, la quantité, le prix, l'acompte (si il y en a un), un commentaire et une série de flags qui décrivent l'avancée de la commande:

- En premier lieu, il a donc fallu créer un objet commande nommé Dafy Order. Comme expliqué plus haut, cela veut dire migration/factory/seeder/model .

- Ensuite, il a fallu appeler un livewire dans la vue qui affiche la table.

- Enfin, il a fallu utiliser la bonne syntaxe pour transformer ce livewire en table filament. Une fois ces étapes faites, on a disposé d'un objet lié à une base de données remplies d'informations factices, et d'un tableau dans lequel afficher ces informations. Il nous suffisait alors simplement d'utiliser filament et les composants qu'il possède afin d'afficher les informations.

Pour les textes, un simple textColumn suffit. Mais pour les flags (reçu, commandé, etc), il a été décidé en premier abord de les mettre en toggle. Des switch, donc, que l'on peut modifier directement dans le tableau et qui appliquent ces modifications dans la base de données. Une fois cette première version faite, il a été décidé de reprendre une autre fonctionnalité du tableau excel : la couleur.

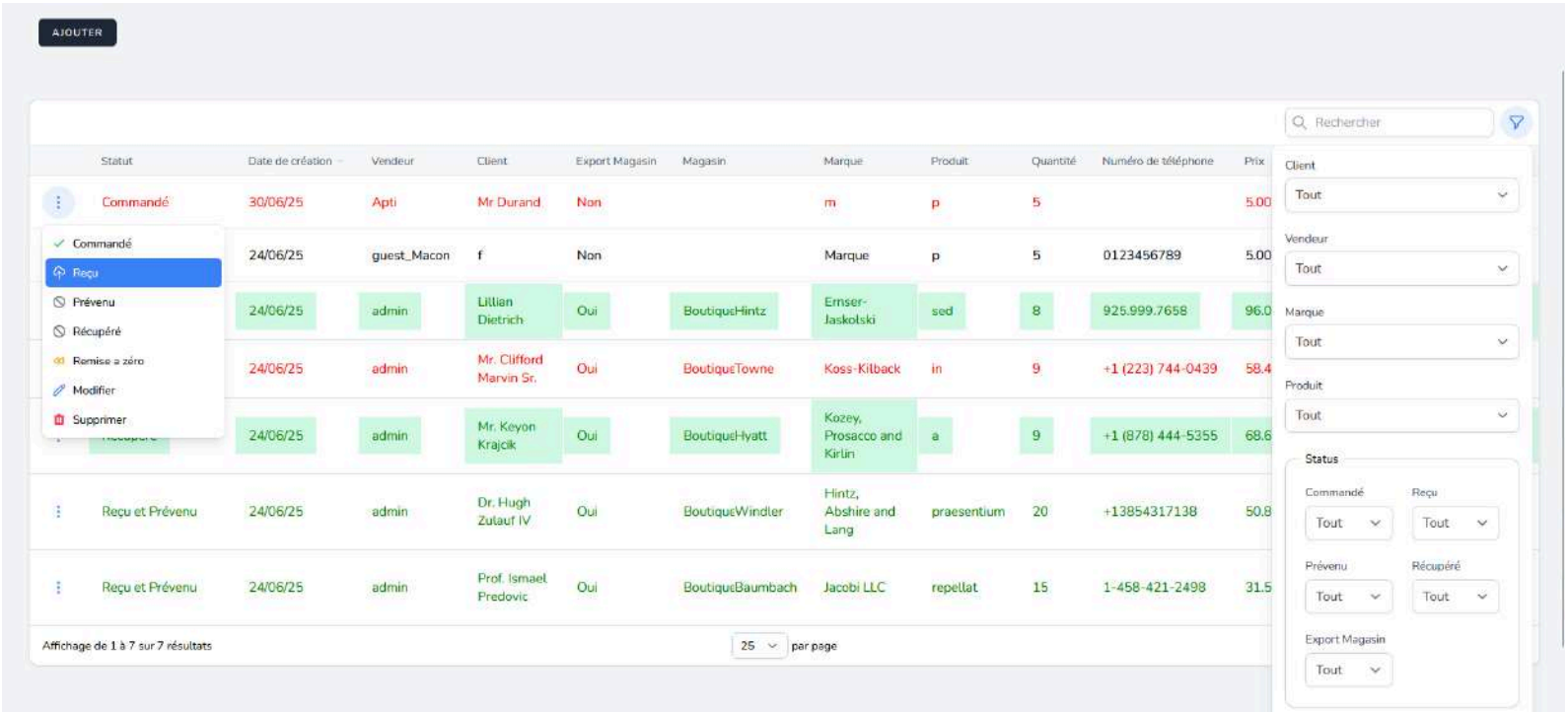
En effet, selon l'état des flags, la couleur de la commande change (on applique alors une couleur conditionnelle via balises css et fonction). Afin de rendre la chose plus facile à comprendre à l'œil nu, j'ai décidé de rajouter un autre champ, qui n'existe pas dans la base de donnée, et qui est donc créé de toutes pièces : le champ status, dont la valeur change selon l'état des flags.

Ce champ permet de donner la même information que les couleurs, mais d'une manière écrite. De plus, a l'aide de filtres, on peut maintenant directement filtrer selon le statut de la commande, chose qui aurait nécessité l'implémentation de fonctions conditionnelles de type "si le filtre "Reçu et Prévenu" est sélectionné, alors on tri par ceux qui ont le flag "commandé" à true, et le flag "reçu" à true, et le flag "prévenu" à true". Cette version-là avait beau avoir l'avantage de la fonctionnalité, on a eu l'idée de gérer les flags non pas avec des toggles, mais directement dans les actions.

De cette manière, on réduit la taille de la table, et on a alors plus besoin de scroller pour aller changer un flag. De plus, on peut maintenant affecter une logique conditionnelle au changement des flags, afin d'éviter qu'un vendeur mette un commande a Reçu alors que la commande n'a pas été commandée.

Si je parle de ces évolutions en grand détail, c'est pour faire comprendre que si l'outil est fonctionnel maintenant, c'est qu' il a eu plusieurs versions auxquelles on a rajouté des fonctionnalités selon les idées de mon tuteur, mes idées a moi, les idées de l'ancien outil, et le feedback des vendeurs. Car oui, à chaque nouvel ajout de fonctionnalité, l'outil a été présenté aux vendeurs, les principaux utilisateurs, pour savoir ce qu'ils en pensaient. Et c'est la combinaison du feedback, d'anciennes idées, et de nouvelles idées, qui rend la conception de cet outil si longue, mais si fonctionnelle.

Après cette dernière évolution donc, on disposait d'un tableau qui affiche tout ce dont on a besoin de savoir via des couleurs et des champs, des filtres qui permettent de cibler seulement certaines commandes, et les actions qui permettent de modifier les flags des commandes tout en évitant de mauvaises manipulations de flag. Une autre évolution viendra plus tard, mais on en parlera dans la partie sur le formulaire d'ajout, qui vient juste après.



Le formulaire d'ajout, donc, est une autre partie importante de cet outil.
En effet, à quoi sert un tableau d'affichage de commande si on ne peut pas créer de commandes ?

Pour arriver sur cette fonctionnalité, on a décidé de passer par un bouton au-dessus de la table d'affichage, et dans un menu présent dans la barre de navigation. Ce qui est intéressant ici, c'est que l'on dispose d'un fichier php appelé navigation.blade.php qui contient notre barre de navigation, utilisée directement par toute les pages de l'application, et affichant les boutons de navigation seulement aux rôles autorisés. Cette centralisation typique de laravel nous permet ainsi de rajouter le bouton tournant la route de l'ajout de commande sur toute les pages de l'application en ne modifiant qu'un seul endroit.

On arrive donc sur la page qui va devenir le formulaire d'ajout de commande. Sur cette page, on va pouvoir réutiliser une livewire en conjonction avec filament, car ce dernier possède des composants spécifiques aux formulaires et facile d'utilisation . On met ainsi des champs texte et de sélection dans la méthode getFormSchema, on met des règles de remplissage dans la méthode rules, et on met une étape de validation des données rentrée avant de créer le nouvel objet dans la méthode submit.

C'est simple, mais ce n'est pas très joli.

Il a donc été décidé de travailler un peu sur le front. En faisant des recherches, je suis tombé sur un magnifique composant filament : le Wizard. Un wizard en informatique, c'est simplement un logiciel ou une fonction qui permet d'automatiser certaines tâches en ne demandant des informations qu'à certains moments, étapes par étape. Dans notre cas, c'est la dernière expression qui va nous être utile : étape par étape. En utilisant le composant wizard de filament, on va pouvoir afficher une suite de champ à remplir dans une étape du wizard. Puis, quand l'utilisateur va cliquer sur suivant, la prochaine étape du wizard s'affiche sur la même page sans rechargement. On va pouvoir regrouper chaque champ similaire dans une même étape du wizard, et ainsi rendre plus simple d'utilisation le processus de rajout de commande ! De plus, on peut décider de rendre les étapes obligatoires, ce qui veut dire que l'on ne peut pas passer à l'étape 4 du wizard sans avoir correctement rempli l'étape 3 . Et par correctement, j'entend que l'on va pouvoir effectuer une étape de vérification des données remplies à chaque fois que l'on souhaite passer a la prochaine étape du wizard, évitant ainsi de remplir une dizaine de champs et de revenir sur chacun d'entre eux quand il y a une erreur.

Si on sacrifie un peu la fluidité du processus qui ne peut donc plus être fait d'un seul coup, on y gagne en compréhension et en sécurité. J'ai alors eu une bonne idée que je me suis empressé d'implémenter : la création d'une étape de récapitulatif à la fin qui affiche comment la commande va être enregistrée , et dans laquelle chaque champs d'un même wizard est regroupé dans une même carte (un genre de conteneur). Et pour aller plus loin, j'ai créé une méthode de désinfection des données avant placage dans la commande. En clair : le user va rentrer les données qu'il veut pour sa commande dans les champs appropriés, puis chaque champ du formulaire (appelé formData) va passer dans une méthode qui va, pour chaque champ et selon son type , enlever les espaces consécutifs, remplacer les lettres avec accent par leur version sans accent, formaliser les nombres pour avoir un espace en temps que délimiteur des milliers, etc . Les données ainsi 'lavées' vont

alors être remplies dans un autre formulaire invisible (le bien nommé cleanedData), et c'est ces données qui vont être utilisées pour l'affichage dans l'étape de récapitulatif. Et si l'utilisateur est content des données lavées et présentées, alors elles seront utilisées pour créer une nouvelle commande (on utilise bien les données contenues dans cleanedData et non formData pour la création de la nouvelle commande).

A noter que pour certains champs tels que le numéro de téléphone, il aura fallu tester la formalisation du contenu du champ avec des expressions regex différentes (on ne veut pas uniquement les numéros français, mais aussi peut être des pays bas ou de belgique, et pas forcément en 07 mais aussi peut être en +33 7. Tout cela nécessite de la regex). Heureusement, les composants filament disposent d'une méthode helpertext() pour marquer en dessous un paragraphe indiquant les différents numéros de téléphones pris en compte.

De la même manière , un problème de 'si on a pas d'export magasin alors on ne devrait pas pouvoir mettre de valeur au magasin d'export' est réglé par l'utilisation d'une méthode sur le composant filament : la méthode visible() . A l'intérieur, il suffit alors de récupérer la valeur de 'export magasin' (qui est un toggable, soit dit en passant) et de mettre une fonction dans la méthode visible que va renvoyer true ou false selon l'état de la valeur récupérée.

Tout cela pour dire : les composants filaments sont polyvalents grâce aux nombreuses méthodes dont ils disposent , et aussi grâce au fait qu'on peut leur donner une fonction en tant que valeur pour que chaque ligne de tableau ou champ sur objet aient des propriétés qui changent selon les valeurs d'autres propriétés. Tout peut s'interroger, tout est interchangeable, et tout peut réagir à des événements (tant que l'on met reactive() sur l'élément qui est censé effectuer un changement).

Cependant, un problème est apparu lorsqu'on a présenté le projet : la visibilité des commandes. En effet, un vendeur de Gap n'a pas besoin de voir les commandes de Sisteron. Cependant, un vendeur "vagabond" qui va travailler dans différents entrepôts va vouloir voir toutes les commandes. De plus, quand il va créer une commande, ca sera a lui de décider depuis quel magasin la commande sera visible, car son magasin associé change avec le temps. Enfin, lorsqu'on met un vendeur précis quand on crée la commande, un utilisateur de gap ne va pas vouloir voir les vendeurs de Sisteron.

On pourrait argumenter sur le fait que le vendeur sera toujours l'utilisateur, et qu'il n'y a donc pas de nécessité à créer un menu déroulant de sélection de vendeur avec le même magasin attitré que l'utilisateur, et on aurait raison. Mais c'est un de ces cas où la vision du développeur se heurte à la manière dont les utilisateurs se servent de l'outil. A la manière d'un chemin de volonté crée par les piétons lorsqu'ils coupent un chemin en passant par une zone herbeuse, un outil est utilisé de la manière qui prodiguera à l'utilisateur le plus de résultat en un minimum d'effort, quelque soit la volontée du créateur de l'outil en question. Ainsi, les vendeurs qui utilisent l'outil… ne se déconnectent pas. C'est à dire que le matin, un vendeur A va se connecter a l'outil, et quand un vendeur B va passer plus tard pour rentrer une commande, il ne va pas se déconnecter, pour se connecter avec son compte, et rentrer sa commande. Il va simplement rester connecté avec le compte de l'utilisateur A et rentrer sa commande. Il faut donc que le vendeur A ait accès à la liste de tous les vendeurs avec lesquels il travaille quand il rentre une commande, car c'est ces mêmes vendeurs qui

vont entrer une commande avec son compte et ils vont avoir besoin de pouvoir accéder à leur nom de vendeur.

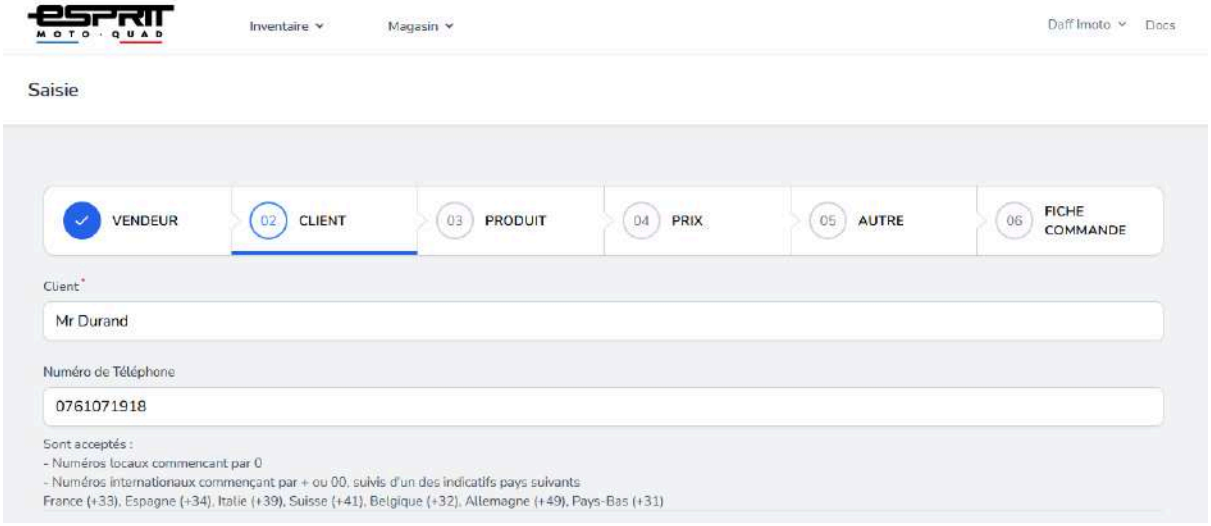
C'est ça, la fonctionnalité supplémentaire dont je parlais dans la partie ‘affichage tableau’ des commandes dafyorder. Car cette visibilité qui est configurée dans l’ajout et l'édition de commandes, on va l’utiliser dans la table d’affichage.

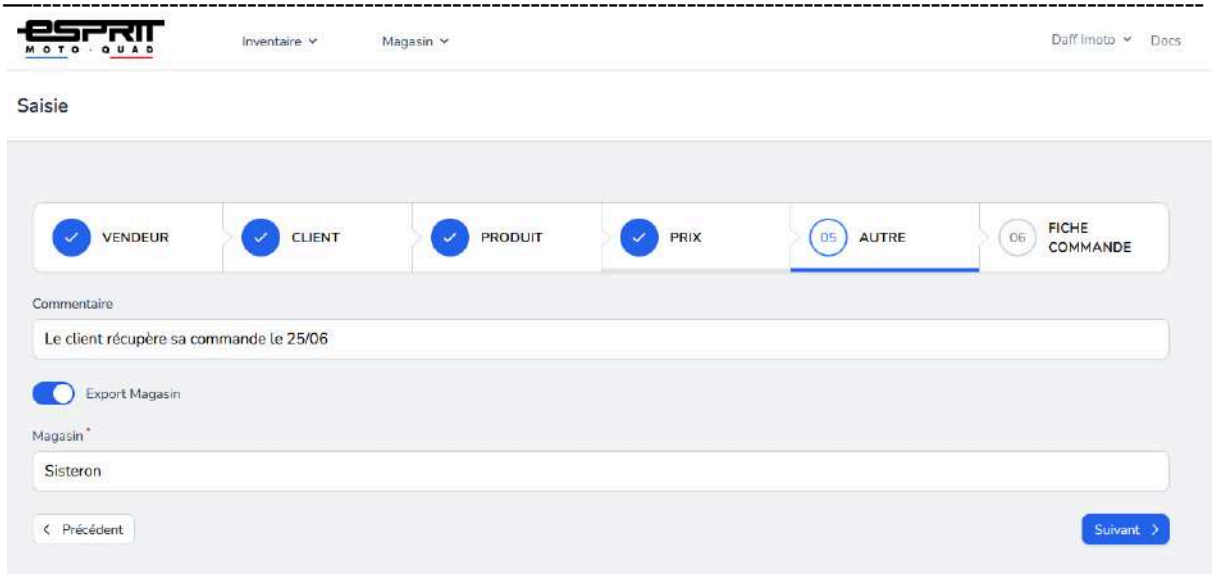
Ces histoires de visibilité et de gestion de rôle aura été une de ces fonctionnalités ‘yo-yo’ auxquelles on revient souvent pour des raisons de bug ou de perfectionnement. Et avec ce dernier point , l’outil était près a être utilisé… à un point près qui est un exemple de ‘filament prodigue des composants complets, mais si on veut faire un changement qui est pas prévu, c’est galère’.

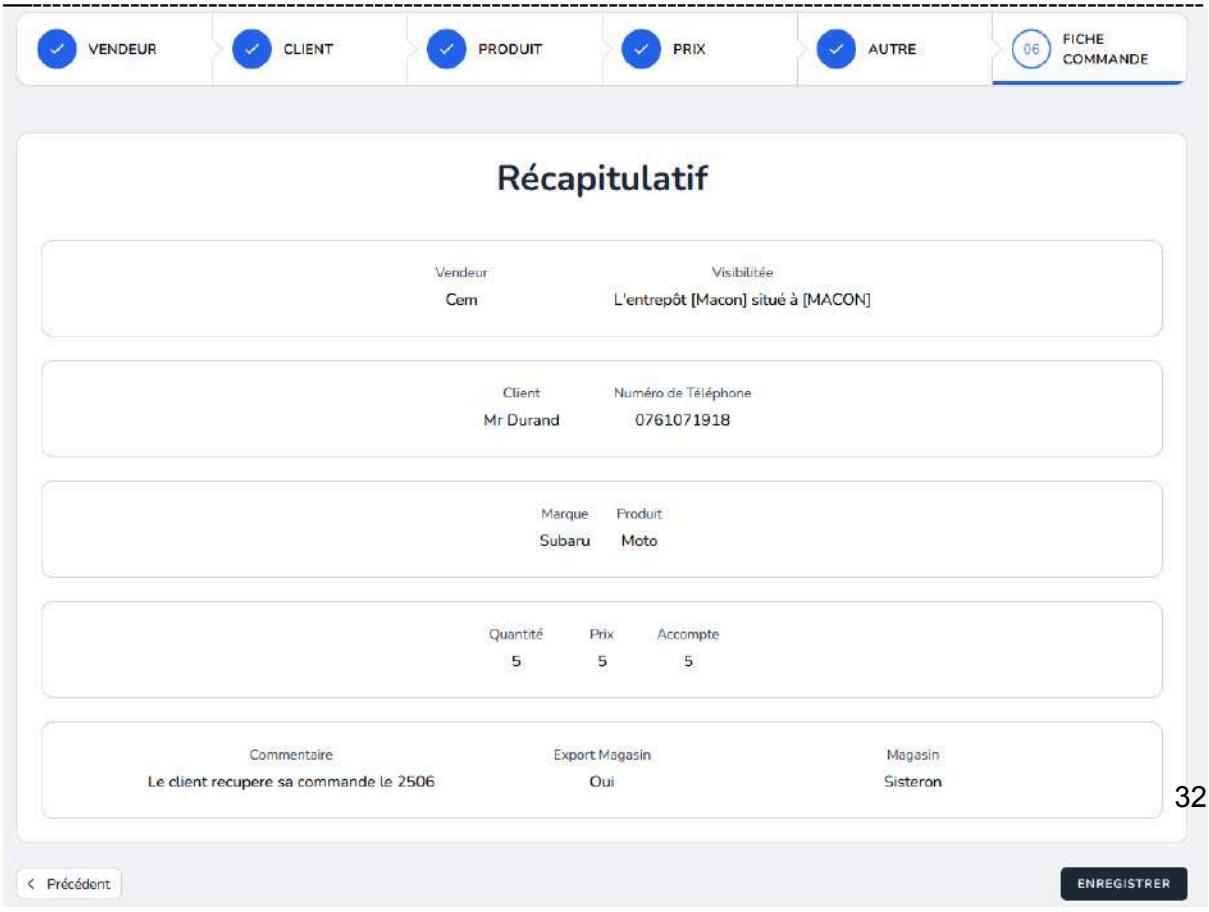
En effet, dans l’étape de récapitulatif du wizard, les noms et autres étaient décalés sur la gauche. J’ai donc simplement voulu les mettre au centre des cards, pour avoir quelque chose de beau et centré. Pour ce faire, j’ai d’abord changé des classes et appliqué des styles via l’outil d’inspection de la page , sur le navigateur. Une fois que le style et les cibles ont été trouvées, j’ai tenté de leur appliquer , sauf que la méthode extraAttribute() qui permet d'appliquer du style sur un élément ne pouvait faire ceci que sur la div de card, et non les éléments à l'intérieur.

La solution trouvée a été d'appliquer une classe spécifique à la div de card, puis de créer un fichier de style en .css contenant du style à appliquer non pas sur la classe spécifique, mais en se servant de la classe spécifique pour aller plus loin dans la matriochka de div. Pour expliquer ceci de manière plus claire, on donne a une div une classe spécifique qui n’existe nulle part autre. Puis, on se sert de cette classe pour appliquer du style à une div spécifique qui se situe à un niveau inférieur. On peut maintenant utiliser ce css et le style appliqué pour remettre tout au milieu. Cependant, on est obligé de compiler npm avec npm build car un gestionnaire de cache nommé vite s’occupe de charger les css de manière plus efficace. Il faut donc faire comprendre a Vite que ce nouveau css est à utiliser. Et de cette manière, nous avons enfin un outil d’ajout qui est fonctionnel, interactif, sécurisé, et agréable à prendre en main.

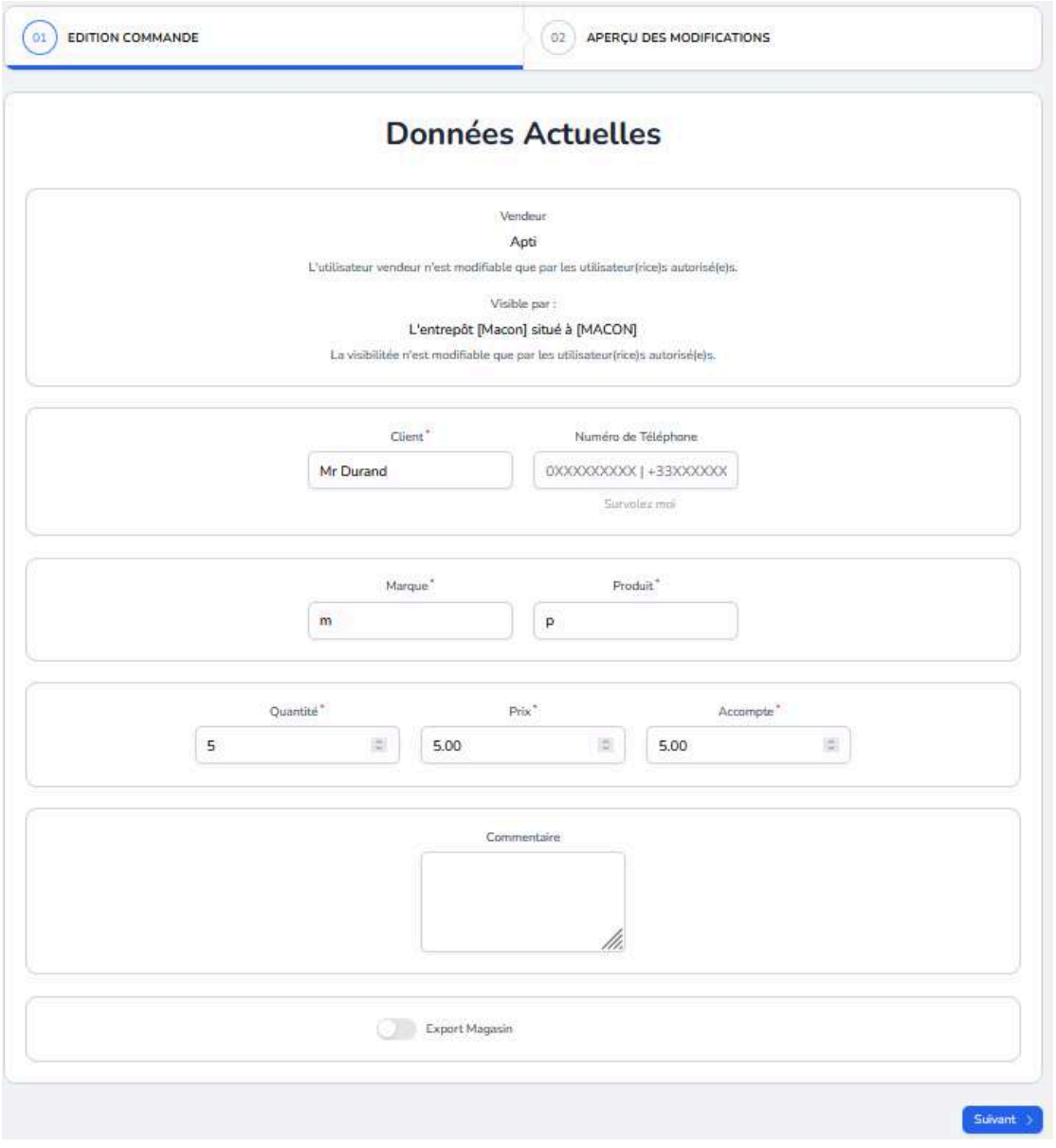
Mais si le problème de Vite et des gestions de rôle nous apprend quelque chose, c’est que changer un aspect de l’outil n’est plus aussi simple que de changer un bout de code. Cela requiert un certain nombre d'étapes intermédiaires, car chaque partie du projet interragissent entre elles. Cependant, il faut garder en tête que pour des si gros projets, une telle arborescence d’interaction est nécessaire pour rendre le tout efficient et maintenable.


Inventaire
Magasin
Daff Imoto
Docs
Saisie
VENDEUR
02 CLIENT
03 PRODUIT
04 PRIX
05 AUTRE
06 FICHE COMMANDE
Client *
Mr Durand
Numéro de Téléphone
0761071918
Sont acceptés :
- Numéros locaux commençant par 0
- Numéros internationaux commençant par + ou 00, suivis d'un des indicatifs pays suivants
France (+33), Espagne (+34), Italie (+39), Suisse (+41), Belgique (+32), Allemagne (+49), Pays-Bas (+31)


Inventaire
Magasin
Daff Imoto
Docs
Saisie
VENDEUR
CLIENT
PRODUIT
PRIX
05 AUTRE
06 FICHE COMMANDE
Commentaire
Le client récupère sa commande le 25/06
Export Magasin
Magasin *
Sisteron
Précédent
Suivant


VENDEUR
CLIENT
PRODUIT
PRIX
AUTRE
06 FICHE COMMANDE
Récapitulatif
Vendeur
Cem
Visibilitée
L'entrepôt [Macon] situé à [MACON]
Client
Mr Durand
Numéro de Téléphone
0761071918
Marque
Subaru
Produit
Moto
Quantité
5
Prix
5
Accompte
5
Commentaire
Le client recupere sa commande le 2506
Export Magasin
Oui
Magasin
Sisteron
Précédent
ENREGISTRER

Maintenant, que l’on peut afficher des commandes et en créer d’autres, il faut parler de la dernière partie de cet outil : l’édition. Déjà, on va mettre son accès dans les actions de la table d’affichage, c'est-à- dire dans le menu contextuel représenté par trois points et qui se trouve à la fin de chaque ligne du tableau, afin de pouvoir effectuer des actions sur chaque objet de manière indépendante. L’une de ces actions, par exemple, est la suppression de commande. Ou encore le fait de faire passer une commande à ‘Reçue’ . On crée donc une action ‘édition’ que l’on décore d’une icône heroicon, et qui va renvoyer la route menant à l'édition . Cette fois ci, on veut faire passer une information depuis une blade a une autre, on va donc la retrouver dans le fichier de route app.php, dans le controlleur qui affiche la blade, dans la blade qui appelle le livewire, et enfin dans le controlleur du livewire (graphique de ‘comment faire passer une information d’une blade a une autre’ en annexe). Une fois l'information d’objet à modifier reçue, on va pouvoir l’utiliser pour récupérer l’objet lui-même, et utiliser ses données dans un autre formulaire. Par souci d’ergonomie, le formulaire d’édition utilise le même principe que le formulaire d’ajout. On a ainsi une étape de wizard qui va afficher toute les données de la commande à modifier dans des champs appropriés, qui vont pouvoir être modifiés par l’utilisateur. Puis, quand il va passer à l'étape suivante, les informations vont être désinfectées/lavées, et affichées dans une étape de récapitulation afin que l’on puisse vérifier ce qui va être enregistré. Si on a pas refait plusieurs étapes obligatoires regroupant des champs similaires (comme pour le formulaire d’ajout) c’est pour une bonne raison : lorsque l’on veut modifier un objet, en général, on veut modifier une information en général, et on a pas envie de se retaper tout les champs pour arriver à l'information voulue.



II.3.γ - Outil de visualisation des stocks STOCK VÉHICULE

L'outil de visualisation des stock n'est pas si différent de la table de visualisation des commandes dafy order. On retrouve des composants qui utilisent un objet représentant des données stockées en base de données. On retrouve des filtres, des différents types de champs, mais aucune action. Les principaux problèmes que j'ai rencontrés lors de l'implémentation de cet outil sont l'accès à ladite base de données, la formalisation des données récupérées, et un des filtres .

Le problème de l'accès à la base de données, c'est que celle-ci est particulière : elle se présente sous la forme d'une bdd hfsql. Ce type de base de donnée crée un fichier pour chaque table, et demande donc à être traduite pour en récupérer les informations sous une forme utilisable.

De plus, ce n'est pas une base de donnée qui se situe sur le cloud avec les autres machines virtuelles (schéma d'explication de l'architecture en annexe), elle se trouve plus proche des locaux, derrière un firewall qui n'accepte que les connexions venant d'une adresse ip en particulier.

Enfin, le logiciel traducteur de BDD hfsql est disponible sous Windows. La bdd est bien sous windows, seulement la machine virtuelle web sur le cloud tourne sous linux.

En clair, pour récupérer les informations et les afficher dans la table, il faut trouver un moyen pour une machine virtuelle sous linux d'utiliser un traducteur sous windows. Et pour ça, on va utiliser un api installé sur un serveur qui peut communiquer avec la bdd.

Cet api, il va écouter certaines routes, faire des choses selon la route appelée, puis retourner des informations, souvent sous forme d'un json. On appelle ce genre de framework un fournisseur d'api, tandis que la machine virtuelle web sur laquelle se trouve le site, s'appelle un consommateur d'api. Dans ce genre de structure, la question d'un os différent ne se pose pas : un ordinateur sous windows peut communiquer avec un api sous linux , et inversement, car la seule chose qui est faite, c'est un retour de json. On a donc configuré un api qui écoute sur la route getVehicleStock, va utiliser un logiciel obcd et son driver odbc hfsql (qui vont servir d'interface de connection couplé a traducteur pour aller récupérer des informations sur la BDD hfsql) pour exécuter une requête sur la bdd, formater les informations récupérées sous forme de json, et renvoyer ce json à l'ordinateur qui a appelé la route (pour plus d'information sur les api et hfsql, mon driver de note est disponible en annexe).

Pour tester ce genre de route et voir le type de données récupérées , mon tuteur m'avait fait installer insomnia, un logiciel qui permet de configurer une route, des actions à faire pré ou post connexion à la route, et de lancer le fetch de l'api pour tester des messages d'erreurs, ou des noms de champs retournés par exemple. Un autre logiciel aura été très utile : le centre de contrôle hfsql. C'est un logiciel qui traduit de manière native les bases de données hfsql, et qui nous permet de parcourir les bases de données pour savoir quelles tables utiliser dans nos requêtes et quels champs demander (ce qui se sera révélé particulièrement utile au vu de certains noms de champs incompréhensibles).

Ceci étant dit, pour ce genre de projet, il ne suffit pas de récupérer des données d'une api, il faut sécuriser cette connexion, car c'est un point d'entrée potentiel. Pour cela, j'ai implémenté un système de token valable 5 min, et une méthode sur laravel qui permet, si on lui donne une route, de d'abord se connecter à l'api pour récupérer les token, puis d'utiliser le token pour se connecter à la route donnée. Comme pour les autres missions présentées ici, on commence par implémenter quelque chose de basique, et on ajoute des fonctionnalités dessus. Ce système a donc évolué et j'y ai implémenté un système de clé symétrique, en dur, stocké dans le .env du laravel et le .env de l'api. Laravel envoie ainsi sa clé à l'api lorsqu'elle cherche à récupérer un token, et l'api compare la clé reçue avec sa clé a lui. Si c'est la même , le token est généré puis renvoyé. l'api est maintenant fonctionnelle, faisant le pont de manière sécurisée entre la machine virtuelle sous linux et la base de donnée + traducteur sur windows. Mais que fait on des données récupérées ?

Comme dit lors de l'explication de la mission pour le premier outil, le livewire couplé à filament peut créer tout un ensemble de tables et form à partir d'un objet relié à une base de donnée, base de donnée que l'on configure via migration/factory/seeder . Alors comment utiliser un retour d'api en json dans un composant qui demande un model lié à une table de la base de donnée ?

La réponse, c'est Sushi .

Sushi, c'est un module de laravel qui permet justement de prendre une réponse en json et de la faire passer pour une base de donnée . Il suffit de le require dans le model de l'objet que l'on veut utiliser, ici 'Véhicule', et de surcharger la méthode getRow pour lui faire retourner la réponse en json. Sushi va ainsi récupérer les informations, créer une fausse table sqlite de cache, table qui sera utilisée par les composants filament comment si elle était une base de donnée ordinaire.

A noter que sushi, a la base, c'est plus pour les csv accessibles directement en local, dans les répertoires. Pourquoi ? Car si le csv en question change, alors sushi recharge sa table sqlite avec les nouvelles données du csv. C'est ce point là qui aura posé un problème, car jamais sushi ne rechargera les informations de l'api par lui-même , même si la base de donnée (et donc les données retournées en json) changent. Tant que le fichier pointé par sushi dans sushiCacheReferencePath ne change pas, la méthode getRow ne sera pas appelée, et donc le fetch des données api à l'intérieur ne sera pas fait.

Ce qu'il faut donc, c'est un moyen de demander à sushi de recharger le cache quand il y a un changement dans les données retournées de l'api. La méthode que l'on a décidé d'appliquer, c'est celle de rajouter une route dans l'api qui retourne le compte de tous les objets des tables que l'on requête. Ensuite, on a implémenté une commande qui permet de récupérer les comptes de l'api, et de le stocker en mémoire s' il est différent de celui qui est déjà stocké en mémoire. Cette commande a été mise en cron, ce qui veut dire qu'elle s'effectuera une fois toute les x minutes (ici 5), ce qui veut dire que l'on possède maintenant un fichier qui est mis à jour toute les 5 minutes, et qui contient le compte total des objets retournés par la requête que l'on fait lorsqu'on veut les informations à afficher dans la table. Il suffit maintenant de faire en sorte que Sushi pointe vers ce fichier de compte et le tour est joué.

En clair : si le compte retourne des nombres différents de ce qu'il y avait il y a 5 minutes en arrière,le fichier est mis à jour. Sushi se rend compte de cette mise à jour, et va ainsi lancer sa méthode getRow qui va , entre autres, appeler l'api pour recharger ses données. Si il n'y a eu aucun changement dans le count, alors le fichier de count n'est pas changé, et sushi continue d'utiliser son ancien cache.

Cette méthode a un défaut, c'est que si le changement de la base de donnée s'effectue sur une modification des informations d'un des véhicules en stock, notre méthode qui ne fait que compter les véhicules ne changera pas le fichier de compte, et sushi ne rechargera rien (le compte des objets n'a pas changé, c'est l'une des informations d'un des objets qui change, et ca notre commande ne le voit pas). Pire encore, si un vendeur supprime un véhicule et en rajoute un autre dans la même base de donnée, la commande ne le verra pas (encore une fois, ce qui nous intéresse avec notre commande laravel custom, c'est le nombre d'objets de la bdd, rien d'autre). Cependant, ce problème a été jugé non important. L'utilisation de sushi dans cette partie était donc nécessaire, et n'a nécessité qu'une implémentation d'un système de rechargement de cache par vérification de compte pour marcher. Ce qui veut dire qu'on possède un moyen de récupérer des données depuis la base de donnée au travers de l'api, et qu'on possède un moyen de transformer ces données en informations utilisable par les composants filament.

Comme je l'avais fait avant pour l'outil de visualisation des commandes, afficher les informations dans une table filament n'était pas compliqué. Ce qui l'a été , par contre, c'est de créer un des filtres. En effet, dans le cadre d'une demande de mon professeur a mon tuteur référent, il fallait que je 'fasse beaucoup de front'.

On m'a donc demandé de faire un filtre qui ressemble à ce que l'on peut trouver sur Amazon quand on veut trier par prix. Ce filtre paraît simple : une barre horizontale, deux handles (des cercles que l'on peut déplacer avec la souris)a chaque extrémité, et on peut bouger les sphères pour modifier les valeurs de prix minimum et maximum. L'implémenter et trouver un moyen de faire marcher ça en tant que livewire, par contre, m'aura pris 4 jours.

J'ai déjà commencé par faire un filtre plus basique, car je me suis dit qu'il vaut mieux avoir quelque chose de moche et fonctionnel, que l'inverse. Ce filtre là, c'est la combinaison d'un simple champ numérique pour une valeur minimum, et un autre pour une valeur maximum.

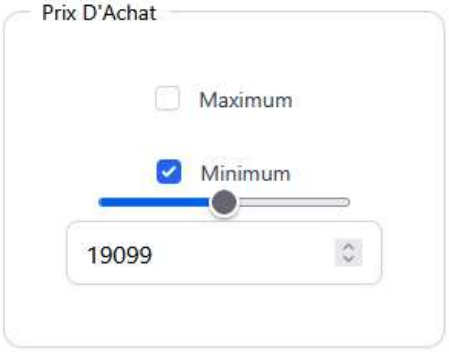
La seconde évolution de ce filtre, c'est l'utilisation d'un composant 'price slider' proposé par tailwind. C'est comme le filtre que l'on souhaite avoir, sauf que celui la ne contient qu'une seule handle. Implémenter cette deuxième version aura été plus dur car cette fois-ci, ce n'est pas un composant filament, il faut donc créer un genre de composant hybride livewire-tailwind. on fait ainsi appel , dans le controlleur livewire de la table à afficher, à un autre contrôleur livewire dans lequel faire passer toute nos méthodes et nos attributs, et qui va retourner la vue blade de ce livewire. Dans cette vue blade, on va pouvoir mettre en place le code du slider de prix de tailwind, et encapsuler le script js pour éviter que, si on ré-utilise ce composant custom, les script s'entremêlent. Pour un peu plus de sécurité, un string haché utilisant tous les attributs donnés lors de la construction du composant est affecté à chaque balises du composant, avant d'être utilisé par le script. De cette manière, on peut être sûr que l'on peut réutiliser le composant plusieurs fois.

La troisième évolution de ce filtre, c'est de faire sortir les valeurs du composant slider pour les mettre dans des champs du filtre de notre table. En clair, faire en sorte que les sliders et les champs puissent interagir entre eux (si je baisse la hande minimum , le champ minimum baisse pour refléter la nouvelle valeur. Si je mets une valeur dans le champ minimum , la valeur de la handle du slider bouge pour refléter la nouvelle valeur du champ. et entre tout ca, la valeur utilisée pour le filtre est celle qui se trouve dans le champ minimum) Après avoir tenté de faire remonter les informations, sans succès, j'ai décidé d'y aller comme une brute : j'ai crée un champ minimum dans les filtres, au niveau de la table filament, puis je suis allé regarder le code généré, et j'ai copié entièrement le html généré pour le champ afin de le mettre dans mon composant custom. J'ai ensuite enlevé toute les informations superflues, et utilisé le string unique pour rendre ces champs la uniques et réutilisables.

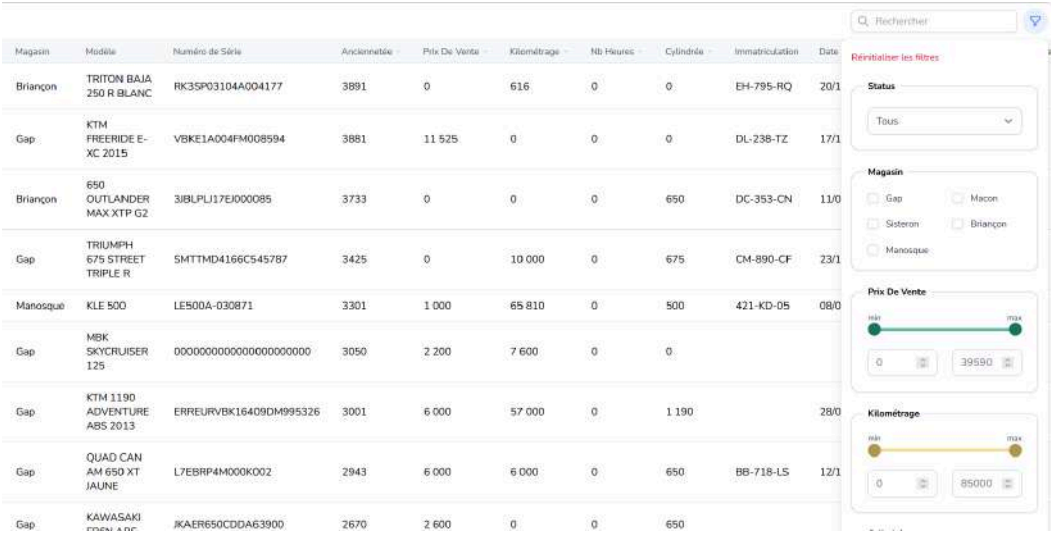
Pour la quatrième et dernière évolution du filtre, j'ai recherché sur le net d'autres personnes qui ont fait des filtres à double slider. Après une après-midi misérable, mon tuteur m'a aidé à en trouver un qui pouvait être remodelé à l'image que l'on attend de lui. C'était un véritable slider a double handle , avec certaines variables de couleurs codées à l'intérieur. J'ai réutilisé la même technique que pour le slider simple afin de rajouter les champs minimum et maximum , et j'ai modifié un peu la logique de javascript afin d'empêcher que les champs se chevauchent ou se dépassent. De plus, vu que la couleur était codée dans une balise style plus haut, j'ai pu relier ces variables au contrôleur de ce composant pour pouvoir changer librement les couleurs et ombres du slider et des handles.

J'avais maintenant un double slider réutilisable/dynamique pour filtrer des champs numériques au bout de plusieurs jours d'intégrations. Un passage sur le discord du groupe pour poser des question de taille de police et de choses a changer, et le filtre était prêt. Avec un api configuré et sécurisé, un modèle utilisant Sushi et gérant la mise à jour des caches, et des jolis filtres dynamiques, l'outil était finalement terminé (jusqu'à la prochaine demande des vendeurs).

Trois évolutions



Aperçu du tableau



III - Les bénéfices et inconvénients
III.1 - Ce que le stage m’a apporté

J’aime bien le dev web maintenant. Ce n’est pas une option que j’avais véritablement envisagé , car faire du front me rend misérable. Les règles se chevauchent, prennent priorité les unes sur les autres, ne font jamais ce qu’on veut qu’elles fassent, en clair : les pages internet ne se prêtent pas à ma vision et se refusent à mon doigté. Je pensais plus partir en développement d'application, ou de logiciel, avec une équipe a mes coté.

Et pendant le stage ?
C'est exactement ce que j’ai fait !

J’ai travaillé au côté d’une équipe pour créer des outils agréables à utiliser pour d’autres personnes. Il y a eu de la réflexion au niveau de la prise en main de l’outil, de son apparence, de ses possibles évolutions, mais aussi des personnes qui sont censées l’utiliser, et de la frontière entre usage prévu et praticité (le problème qu’on a eu avec “le vendeur est censé se connecter avec son compte, mais enfaite il crée ses commandes sur le compte d’un autre car il veut juste accéder à l’outil”). J’ai beaucoup apprécié le va et vient avec les vendeurs pour toujours rajouter quelque chose de plus, et perfectionner l’outil au fur et à mesure des itérations. C’est très agréable aussi, quand on code quelque chose et qu’on voit les gens l’utiliser.

Une autre chose que ce stage m’a apporté, c’est dans ma manière de coder. Comme le dit un grand homme , un développeur se doit d'être feignant, pas fainéant. Cela veut dire qu’il faut trouver un moyen pour que les changements puissent se faire sans avoir à entrer dans le code, en changeant une valeur stockée quelque part ou autre (et pas… ne rien faire). Ça veut aussi dire que si un bout de code est ré-utilisé, alors il faut le rendre réutilisable et dynamique, pour que si on veut rajouter des fonctionnalités , cela puisse se faire sans avoir à modifier le fonctionnement du bout de code, et sans toucher aux autres fonctionnalités.

De plus, j’ai appris à mieux organiser mon travail, en faisant du workflow git. Je fais plus souvent des commits pour enregistrer mon travail, et je prend le temps de faire les choses. Un des principes que j’ai cultivé en cours, par exemple, et que j’ai pu mettre en application ici, c’est de passer du temps avec la technologie et de faire des comptes rendus/notes sur la technologie jusqu'à être assez confiant pour pouvoir me lancer corps et âme sur une feature. Avec laravel par exemple, le tuto a beau durer 40 minutes, j’y ai passé 1 jour et demi. J’ai, à ce jour, fait plus de 30 pages de notes entre laravel, les api, hfsql et filament, et ca ma permi de comprendre d'où venaient les problèmes quand il y en avait, et les étapes préliminaires à effectuer pour résoudre le problème.

En clair : prendre le temps de connaître les solutions que l'on veut implémenter ainsi que les techniques utilisées pour les implémentations.

D’une manière plus personnelle, j’ai appris a accepter l’échec lors de la recherche de méthodes . Quand un développeur veut créer quelque chose, il a à sa disposition de nombreuses manières d’arriver à ses fins. Et au contraire d’un TP où tout est déjà préparé, ça va être à lui de tester les différentes méthodes jusqu'à trouver celle qui conviendra le mieux à ses attentes. Le problème que j’ai eu par exemple avec les api (et qui au final n’en

était pas un, vu que c'est tout de même la solution qu'on a implémenté au niveau de l'api,) c'est que j'ai passé un après midi entier à chercher des méthodes et des solutions et des fonctionnements que l'on pourrait utiliser, tout ca pour au final utiliser la méthode qui a déjà été implémentée. C'est dur de se dire qu'on a écrit des centaines de lignes pour rien, mais au final, il faut se dire que toutes les méthodes que l'on a testé, même si elles ne marchent pas, nous rapprochent des méthodes qui vont marcher. On fait le tri, petit à petit, et tout ce qui est mis de côté est gardé pour plus tard. C'est ça finalement, être un développeur : avancer dans son code, ligne par ligne, problème par problème, attente par attente, petit à petit. On teste, on re-teste , on se pose des questions, on regarde pourquoi ça ne marche pas, et avec chaque échec on se rapproche de quelque chose qui fonctionne. Tout est une question de patience.

Et enfin, l'une des choses que ce stage m'a apporté, et qui est pour moi la plus importante d'un point de vue personnel, c'est l'appréhension du travail et la gestion de l'effort. J'ai déjà travaillé avant, dans ce genre de boulot ou on compte les secondes, et j'avais un gros apriori sur la manière dont le stage allait se dérouler. Entre les jours à attendre, le travail non satisfaisant, et juste l'appréhension de rentrer tous les soirs en me disant que jamais je ne trouverai satisfaction dans un travail, et que ce ras le bol général serait le seul sentiment que je subirais jusqu'à la retraite. Finalement, je me suis retrouvé dans un environnement agréable, libre , dans lequel j'ai pu apprendre en sécurité et mettre en pratique ce que j'ai appris dans des choses qui apportent du bonheur aux autres. J'ai pu utiliser ma créativité, me lancer dans des projets que j'avais envie de continuer et d'améliorer. Et au final, la plupart du temps, je me lançait dans une feature et ne levais les yeux de l'écran qu'au bout de 3 heures, avant l'impression d'avoir passé 10 minutes à coder. C'était une joie de pouvoir faire quelque chose qui me satisfait, ou je ne vois pas le temps passé, qui profite à la société, et qui rapporte de l'argent (pas pour l'instant, bien entendu). J'ai pu appréhender le travail de manière sereine, et c'est très précieux pour moi.

Le seul problème, c'est que j'avais l'impression de ne pas avancer assez vite, et de ne pas assez travailler. J'avais cette boule au ventre de me dire que j'étais lancé dans le grand bain avec des technologies que je ne maîtrisait pas du tout et des objectifs qui semblaient impossibles à atteindre. Pour une fois , j'étais devant un ordinateur avec des choses à faire, et je ne voyais pas directement la manière et les outils dont j'allais avoir besoin pour réussir.

Ce sentiment de mal-être, j'ai décidé de le refuser, et je me suis alors forcé à prendre mon temps et à accepter que , en tant que parfait débutant dans le domaine de l'informatique, j'allais faire les choses à mon rythme. Sans me frustrer du manque de résultat immédiat, sans prendre peur devant les lignes de codes que je ne comprenais pas, sans m'en vouloir de passer 10 minutes à prendre l'air toutes les 1h et demies. Et ce sentiment a fini par disparaître au bout de quelques semaines.

Si je devais choisir seulement 2 apports importants de ce stade, c'est que je me suis découvert une flamme de passion pour l'informatique, qui a tout autant brillé en cours que dans le monde du travail, et que j'ai appris à gérer cette flamme en éloignant la frustration et en décorant cette motivation d'un tempo régulier, mais tranquille. On le sait : ça ne sert à rien de travailler corps et âme dans ses projets, sans pause, pendant trop longtemps, sauf si on veut faire un burnout, détruire sa santé mentale, et finir par élever des lama en Dordogne quelques années plus tard.

III.2 - Les problèmes rencontrés et solutions proposées

- *Problème* : Un mal-être causé par le manque de confort avec les technologie compliquée et inconnues, un manque de résultat , et mes précédentes interactions avec le monde du travail
  Solution : Accepter l'échec, Accepter qu'un débutant va naturellement moins vite, prendre/perdre son temps, Prendre fierté dans le travail accompli

- *Problème* : Faire remonter les informations depuis le composant custom jusqu'à la table filament
  Solution : Plutôt que de déplacer les informations à travers le livewire, faire venir les champs à remplir dans le livewire. Copier le html en dur des champs dans lesquels je voulais mettre les informations,le coller dans le composant, et le rendre unique

- *Problème* : Passer des heures sur des solutions qui ne seront pas implémentées
  Solution : Accepter que cela fait parti du travail, et que ces heures là ne seront jamais perdues car les compétences apportées seront utilisées ailleurs

- *Problème* : Impossible d'utiliser un traducteur de bdd hfsql sur windows alors que le serveur web est sous linux
  Solution : Créer un api sous windows qui a accès à la bdd hfsql, lui donner une route qui va aller chercher l'information et la formater, et appeler cette route depuis le serveur web

- *Problème* : Jambes lourdes et mal de dos
  Solution : Se forcer à prendre des pauses pour aller marcher et se dégourdir les jambes, aller faire du sport 2 fois par semaine

- *Problème* : Trop de choses dans la documentation des techno utilisées, ne sais pas quoi prendre et comment l'implémenter
  Solution : Poser tout un tas de question a chat gpt jusqu'à maîtriser entièrement les méthodes utiles dans le contexte, quitte commenter ligne par ligne les solutions implémentées

- *Problème* : Les données renvoyées par l'api sont difficiles à utiliser, et je n'arrive pas à m'y connecter
  Solution : installer un logiciel comme insomnia qui permet de tester les routes et de voir le type de réponse renvoyées

- *Problème* : Chaleur empêche de dormir, la fatigue s'accumule
  Solution : Café, au maximum une fois par jour pour éviter l'addiction

- *Probleme* : Perte de cheveux a cause des émotions fortes causées par un code qui devrait marcher et ne marche pas
  Solution : Écouter de la musique, prendre son temps à expliquer son code (méthode du canard en plastique), faire des pauses régulières pour boire de l'eau, implantation capillaire en Turquie.

Conclusion

Le +

Je n'appréhende plus le monde du travail
Je suis content de me dire que j'ai trouvé une voie dans laquelle je vais pouvoir m'engager
J'ai pu faire joujou avec des technologies diverses et variées (de la gérance de cache avec CloudFlare jusqu'à l'architecture web Microsoft Azure)
J'ai pu apprendre deux trois trucs en dehors du stage qui vont me servir plus tard
J'ai fait du bon boulot
J'ai bien géré mon stress et les attentes qu'on avait de moi
J'ai appris une nouvelle manière de coder, de gérer un projet, et de ranger les fonctionnalitées
J'ai bossé avec des gens sympa
La glace offerte par le patron, pendant la canicule
Mon filtre en double slider que j'ai mis tant de temps à coder, mais qui marche tellement bien
J'ai appris à contourner les obstacles , des fois de manière extrême
Le discord pour communiquer
Le tableau blanc pour expliquer
La machine à café gratuite
Le frigo pour la nourriture du midi
Les pauses quand on veut tant que c'est raisonnable et qu'on bosse bien à côté
Le lien avec les usager du code que je crée (donne un sens à ce qu'on fait), à la fois quand on va les voir pour demander du feedback, ou qu'ils viennent nous voir pour qu'on leur fasse la démo d'un outil
La musique en travaillant
La proximité a mon logement

Le -

Le rapport de stage, quel outrage
La chaleur, quelle horreur
J'ai quelques fois utilisé chat gpt comme raccourci plus que professeur, je me demande bien où j'en serai arrivé sans cet outil

Si c'était à refaire

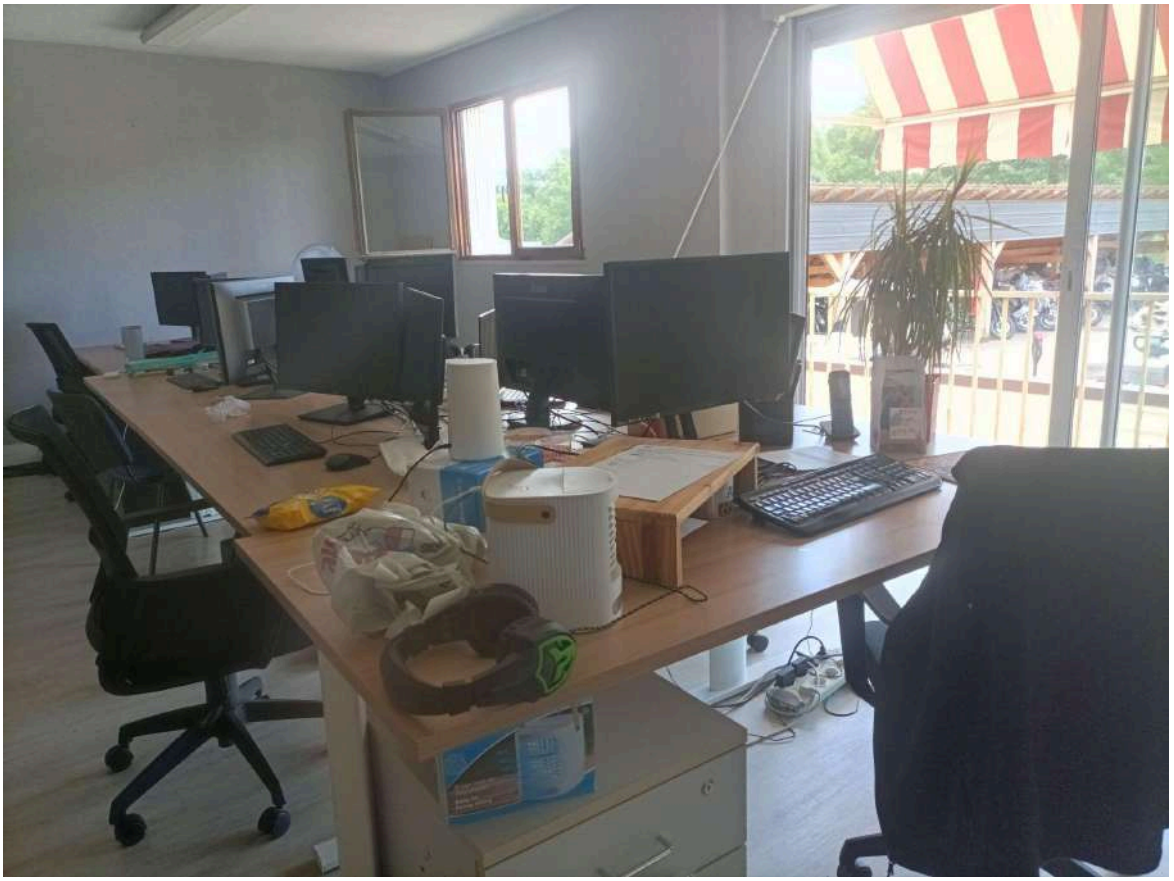
J'amène 4-5 bouteilles d'un litre d'eau que je garde dans le frigo, ainsi que ma tasse et un ventilateur rien que pour moi
Je ne me force pas à venir en jean quand il fait 39 dehors. Être bien habillé, ca peut aussi se faire avec un short
Vu que je sais que je suis capable de fournir quelque chose, essayer de n'utiliser chat gpt que l'après midi (ce qui me force à me dépatouiller sans pendant toute la matinée)
Mariner le poulet avant de le mettre dans le wrap du midi. Je suis sûr que ça aurait été meilleur
Des mister freeze dans le congélateur (Chimique ? Évidemment. Frais ? C'est l'idée !)
Aller plus souvent voir les vendeurs le matin/pause café, pour leur dire bonjour, taper la discute ou juste récupérer plus de feedback pour les outils
Amener d'autres capsules pour la machine à café (le latté de l'ours, c'est un latté au spéculoos, c'est divin et ca change du simple espresso. source : j'en ai à la maison)

Annexes

Le latte de l'ours. C'est vraiment bon !

Les Photos

Les locaux, vu de l'intérieur



Le couloir d'entrée des locaux, caché derrière une grille

Le Code

Un exemple de code pour la table filament. On voit que cela consiste en une méthode pour lister les lignes et colonnes à afficher, et des composants spécifiques pour chaque colonne, selon ce que l'on veut

```php
/**
 * Liste et type des colonnes à afficher
 */
0 references | 0 overrides
protected function getTableColumns(): array
{
    return [

        // Split pour mettre tout les champs les un au dessus des autres quand on dépasse md (une balise d'écran, quand l'écran est trop petit quoi.)
        // Split permet surtout d'afficher les éléments de manière horizontale tant que c'est possible, et les empile verticalement en dessous du breakpoint
        Split::make(schema: [

            // Premiere colonne : l'opérateur, avec le nom du chap au dessus (above)
            TextColumn::make(name: 'operator')
                ->description(description: 'Operateur', position: 'above'),

            // Deuxieme colonne : le sku et le fournisseur, dans un stack.
            // Le stack fait en sorte que les deux champs soient dans la meme colonne,
            // l'un au dessus de l'autre (2 espaces dans ce stack, comme indiqué plus loin)
            Stack::make(schema: [

                // Premier champ de la deuxieme colonne : le sku
                // le parametre searchable fait que le filtre de texte peut aller taper dans ce champ
                TextColumn::make(name: 'sku')
                    ->description(description: 'SKU', position: 'above')
                    ->searchable(),

                // Deuxieme champ de la deuxieme colonne : le fournisseur
                // le parametre wrap fait que la valeur de ce champ continuera en dessous si il est trop long
                TextColumn::make(name: 'supplier')
                    ->description(description: 'Fournisseur', position: 'above')
                    ->wrap(),

            ])->space(space: 2),

            // troisieme colonne : la description
            TextColumn::make(name: 'name')
                ->description(description: 'Description', position: 'above')
                ->wrap()
```

Le regex utilisé pour forcer l'utilisateur à donner un numéro de téléphone convenable lors de l'ajout d'une commande

- **Un peu de regex : les numéros de téléphone**

```php
->regex('/^(?:0|(?:\+|00)(33|34|39|41|32|49|31))[1-9][0-9]{8}$/')
```

?:0	doit commencer par un zéro
|	ou alors
(?:\+|00)	doit commencer par un + ou un 00
(33|34|39|41|32|49|31))	et avoir juste après un des couples de nombres marqués ici
[1-9]	et avoir juste après un nombre entre 1 et 9 compris
[0-9]	et avoir juste après un nombre entre 0 et 9 compris
{8}	(8 fois)

On a alors une expression qui autorise les numéros de téléphones qui *commencent par 0* ou alors *commencent par un agrégat de + ou 00 suivi d'un des couples de nombres correspondant aux indicatifs* , le tout suivi d'un nombre entre 1 et 9 inclus, et de 8 nombres entre 0 et 9 inclus.

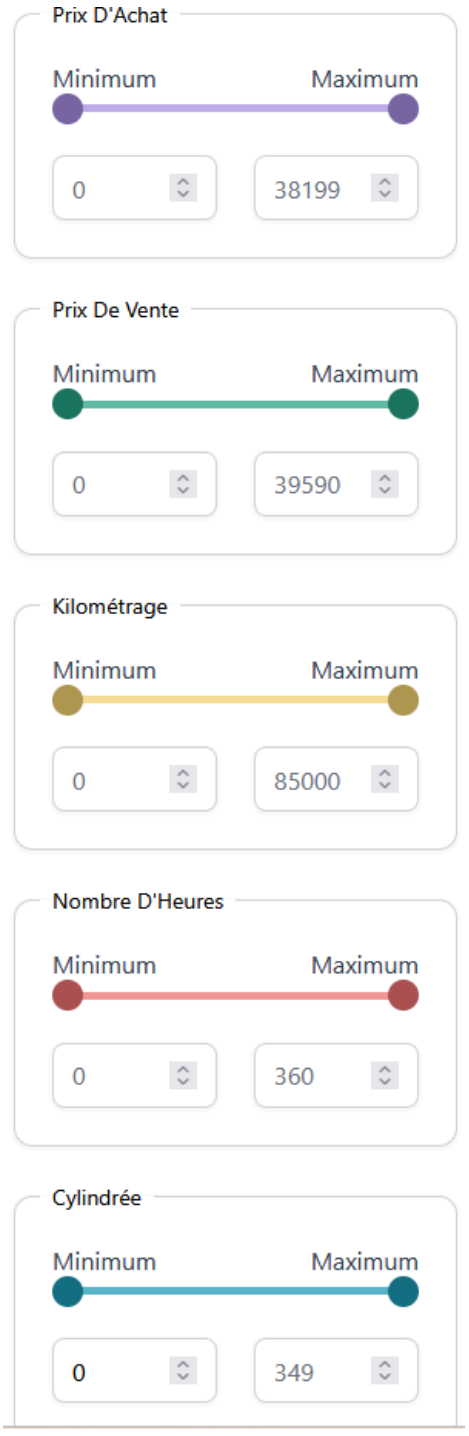
La méthode qui s'occupe de nettoyer les données rentrées par l'utilisateur lors de l'ajout de commande. Le regex ici est aussi est obligatoire :

```php
$cleanedValue = $value;
if (in_array(needle: $key, haystack: ["client", "brand", "product"]) or (in_array(needle: $key, haystack: ["store"]) and $value != NULL)) {

    // Supression des espaces au début et a la fin de la chaine
    $cleanedValue = trim(string: $cleanedValue);

    // Remplacement des accents par des caractères simples (é → e)
    $cleanedValue = Str::ascii(value: $cleanedValue);

    // Suppression des caractères non latins et de l'ascii invisible
    $cleanedValue = preg_replace(pattern: '/[^\x20-\x7E]/u', replacement: '', subject: $cleanedValue);

    // Nettoyage espace multiples → 1 seul espace
    $cleanedValue = preg_replace(pattern: '/\s+/', replacement: ' ', subject: $cleanedValue);

    // Autorise lettres, chiffres, espaces, tirets, apostrophes, etc.
    $cleanedValue = preg_replace(pattern: '/[^a-zA-Z0-9 \-\'&]/', replacement: '', subject: $cleanedValue);

} elseif (in_array(needle: $key, haystack: ["price", "deposit"])) {

    // Supression des espaces au début et a la fin de la chaine
    $cleanedValue = trim(string: $value);

    // Nettoyage espace multiples → 1 seul espace
    $cleanedValue = preg_replace(pattern: '/\s+/', replacement: ' ', subject: $cleanedValue);

    // Garde seulement les chiffres
    $cleanedValue = preg_replace(pattern: '/\D/', replacement: '', subject: $cleanedValue);

    // Remplace les virgules par des points
    $cleanedValue = str_replace(search: ',', replace: '.', subject: $cleanedValue);

} elseif (in_array(needle: $key, haystack: ["phone_number"])) {

    // Supression des espaces au début et a la fin de la chaine
    $cleanedValue = trim(string: $value);

    // Supprime les tirets, virgules, espaces et points (devrait pas y en avoir, mais au cas ou)
    $cleanedValue = str_replace(search: ['-', '.', ',', ' '], replace: '', subject: $cleanedValue);

}

// Limite à 255 caractères (au cas ou les champs de la bdd soient limités)
$cleanedValue = substr(string: $cleanedValue, offset: 0, length: 255);

$this->cleanedData[$key] = $cleanedValue;
```

Le Component Custom Double Slider

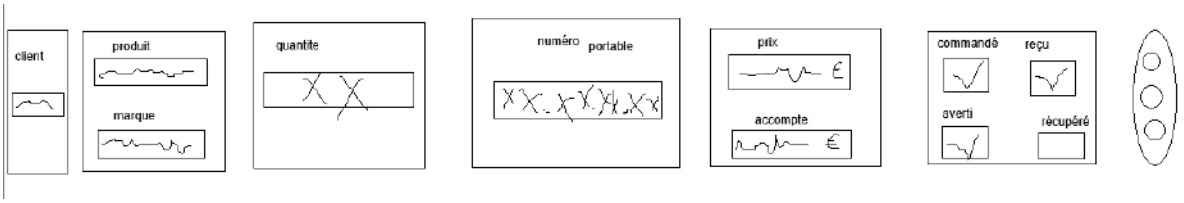
Une capture d'écran que j'avais prise pour illustrer que pratiquement toute les parties du composant double filtre que j'ai créé sont customisables au niveau de la couleur. A noter que j'ai aussi implémenté des méthodes pour changer la police d'écriture et taille des labels.

Un aperçu des filtres tels qu'il apparaissent dans la version finale, et utilisant le contrôle des couleurs montré ci dessus

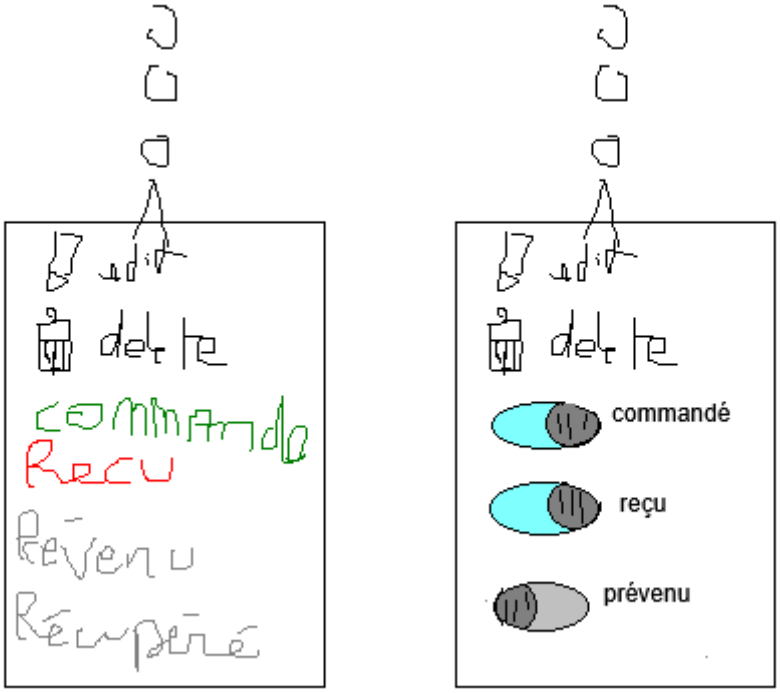


Les Prototypes

L'outil de visualisation des commandes



La gestion des flags de commandes (reçu, récupéré, etc) à l'intérieur des actions plutôt qu'avec des toggables



Premiers essais lors de la création de mon composant filament double slider custom. Illustre la notion de 'le livewire est une fenêtre vers une version d'un component déjà configuré, configurable, et réutilisable.'

La première version de la solution pour la gestion de cache Sushi. Comme expliqué plus haut, on utilisera au final le 'compte de véhicule des bdd distants, et si il diffère du dernier count, alors on met à jour un fichier qui va provoquer une récupération de toutes les données des bdd distants par sushi'. Cette version, cependant, aura été implémentée au préalable, et marche aussi.

- **Sushi : les caches**

Sushi met en cache les données d'objet. Ce qui veut dire que tant que le cache n'est pas supprimé, sushi n'ira pas récupérer les informations de la bdd. Cela pose un problème, car les changements de la bdd ne seront jamais pris en compte si le hash (ordre et nom des objets stockés). Pour éviter ce problème , on peut :

- Invalider le hash en ajoutant un champ bidon à tous les objets récupérés. Dans ce cas, on n'utilisera jamais le cache, ce qui peut être problématique/lent s' il y a beaucoup de données à récupérer.
- Ajouter le temps dans le nom du cache sauvegardé. Quand Sushi va aller voir s' il n'a pas déjà les données en cache, il va regarder le nom du fichier de cache. Si ce dernier comporte le temps sous la forme d'heure, alors il va regarder par exemple cache_9_04_06_2025 . si il est 9h02 ou 9h45, sushi ira tout de même regarder cache_9_04_06_2025, trouver des données, et utiliser ces infos. Si par contre il est 10h, il ne trouvera pas de données dans cache_10_04_06_2025 et refera le cache. Ce qui veut dire qu'on a un cache qui se refait toutes les heures. Cependant, il faudra manuellement supprimer tout les caches crées (par cron ou autre)

La meilleure manière de faire, c'est de surcharger getCacheFilename pour faire en sorte que Sushi utilise un slot basé sur le temps (par exemple entre 0 et 30 min : slot a , slot b le reste du temps) pour créer le nom du cache. Ensuite, on définit le nom du cache en utilisant le slot, et on supprime l'ancien cache en utilisant le nom du cache ainsi que l'autre slot. Cela permet d'avoir un cache qui se fait détruire et reconstruire toutes les 30 minutes. Il ne faut pas oublier de rajouter un champ slot dans les objets retournés par la bdd , ceci afin de changer le hash toute les 30 min et d'être ainsi sur qu'une ancien cache ne sera pas utilisé (le slot changeant aussi toute les 30 min, on aura alors une suppression de l'ancien slot et un changement de hash qui va provoquer la création d'un nouveau cache toute les 30 minutes). Ce système permet d'avoir un refresh garanti sur les données toutes les 30 minutes, mais on peut l'adapter pour en avoir un toutes les 5 minutes par exemple.

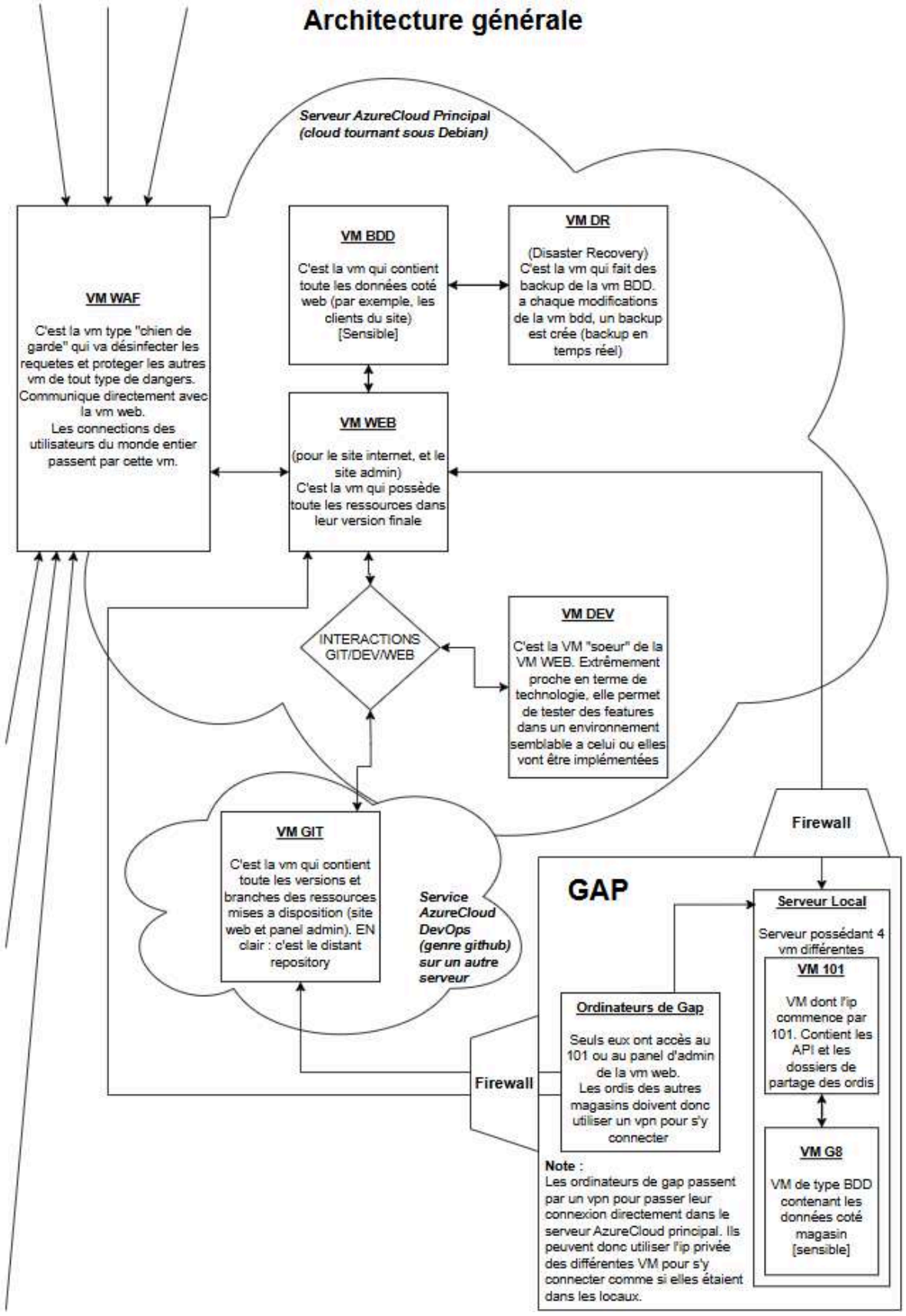
En clair :

3. Meilleure pratique : système de *slots* (🧠 efficace et propre)

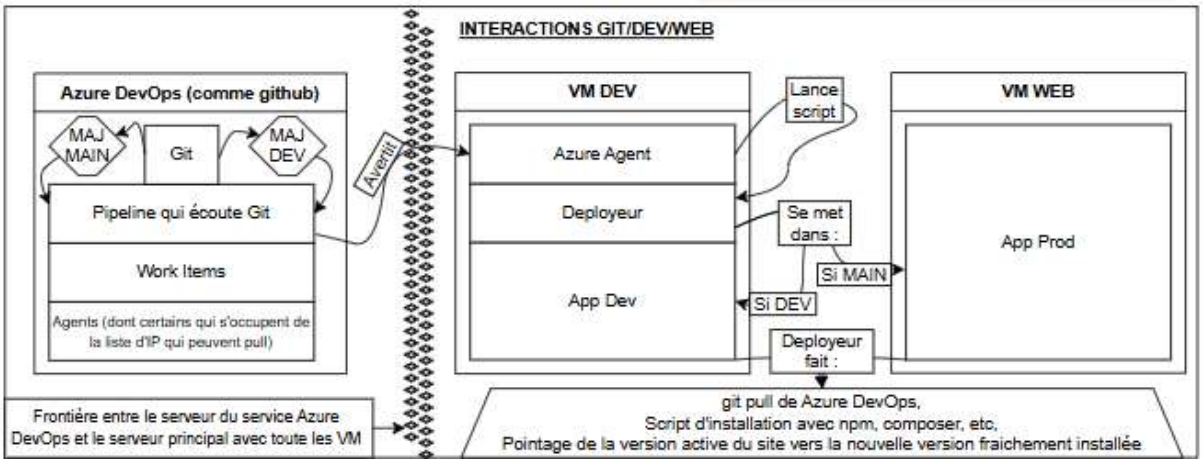
- Divise ta journée en **slots fixes** (ex : A pour 0-30 min, B pour 30-60 min)
- Dans `getCacheFilename()`, renvoie un nom basé sur le slot courant
- Dans `getRows()`, ajoute une clé `_slot` aux données (ex : `'slot' => 'A'`)
- Supprime l'ancien fichier de l'autre slot au moment du changement

Les Schémas

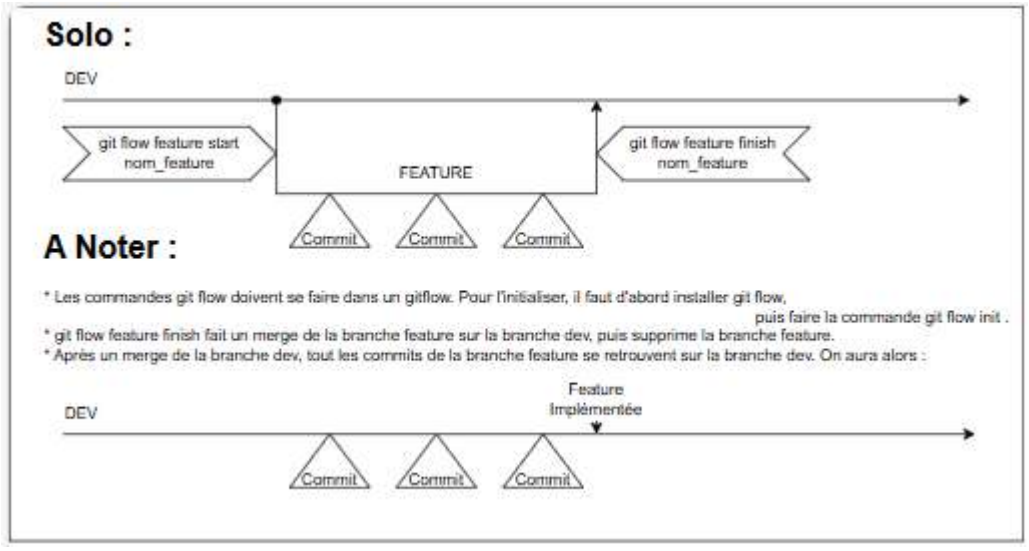
L'architecture Web



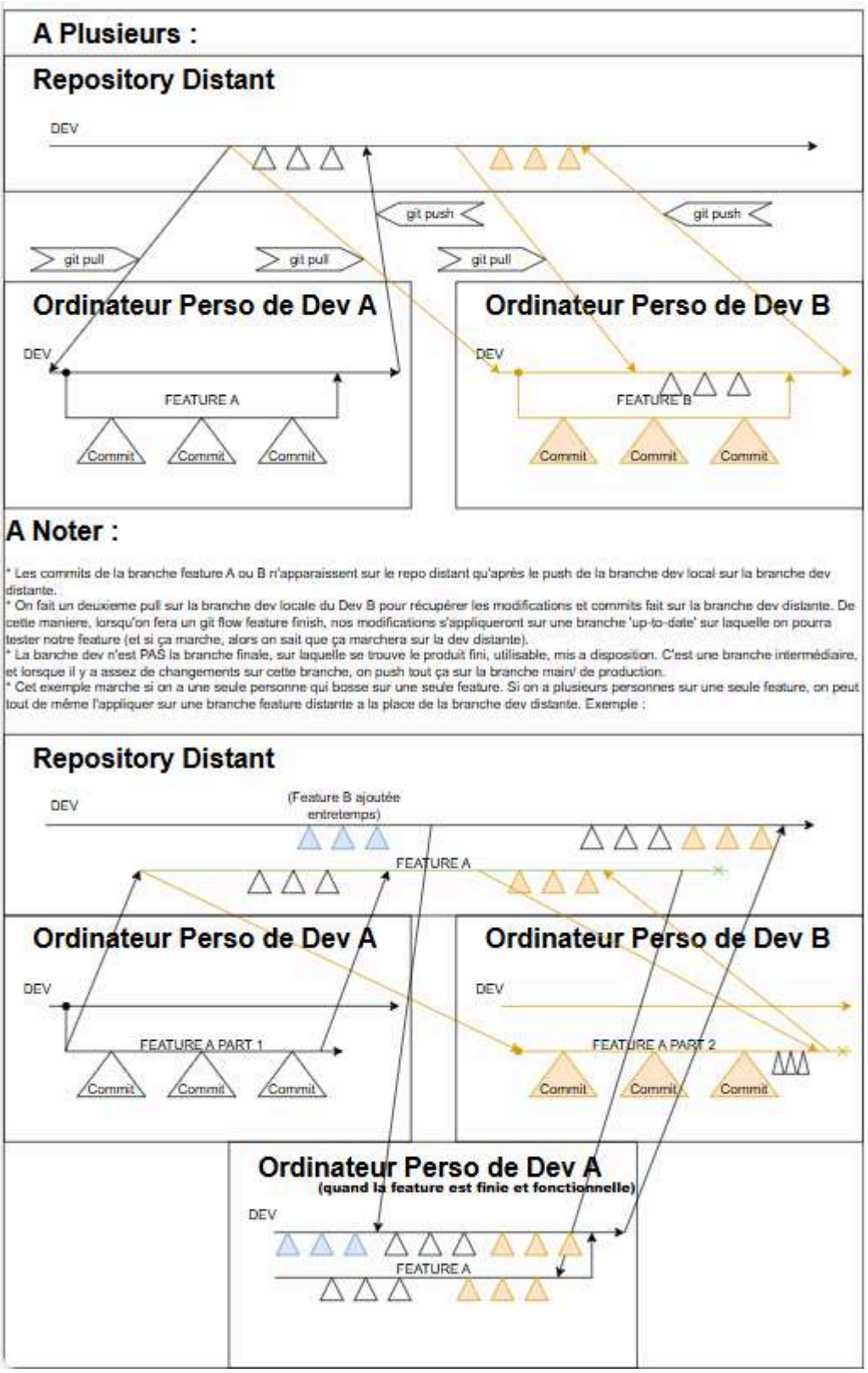
Le Déploiement, de git au web



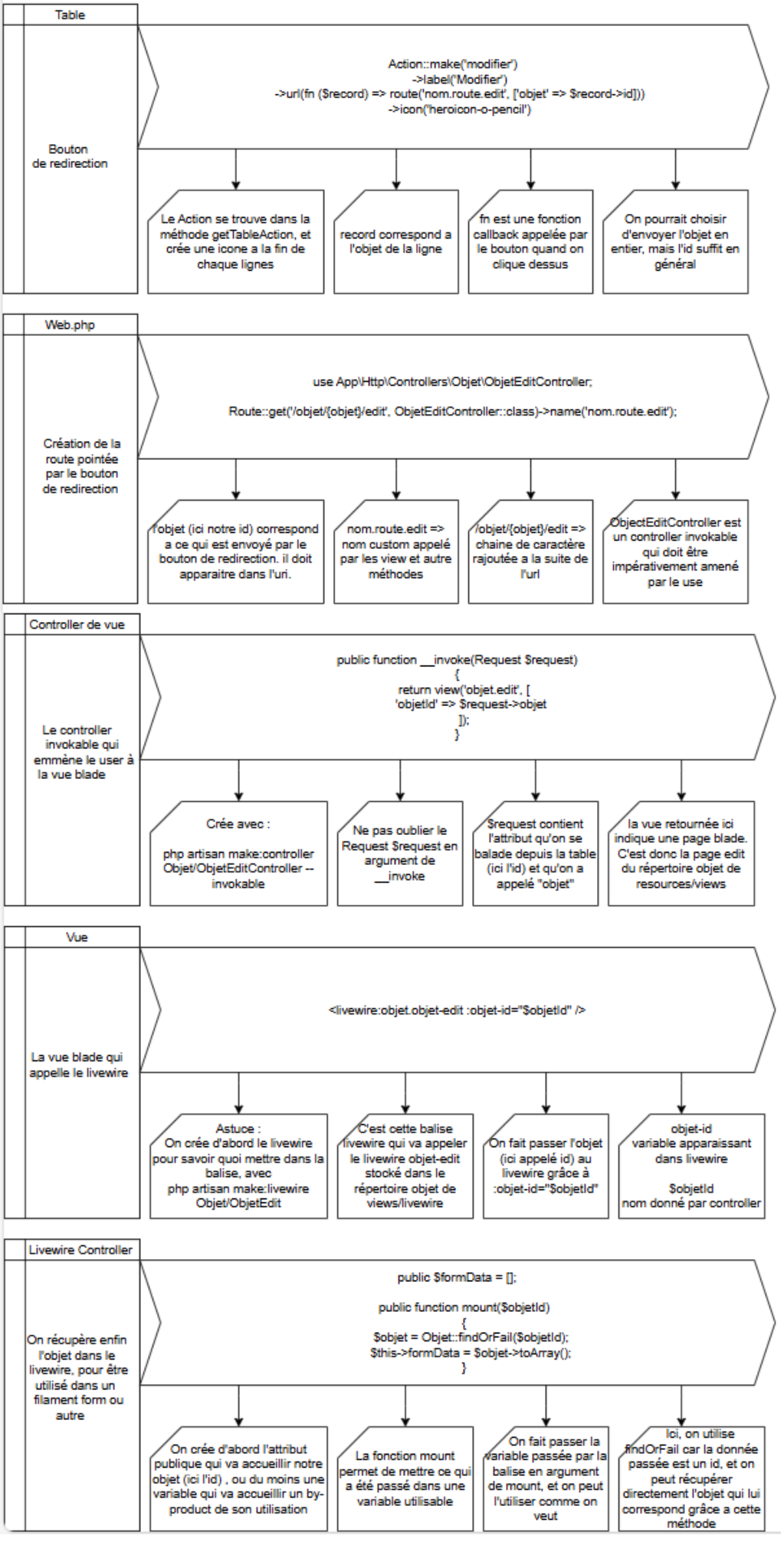
Workflow Git : en Solo



Workflow Git : à Plusieurs

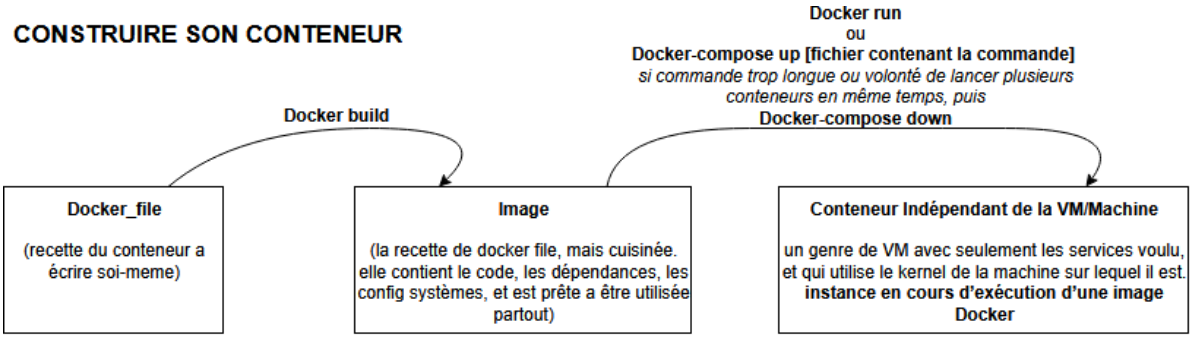


Livewire : Faire passer des informations d'une page à un composant



Docker et déployeurs : c'est toujours bon à savoir

CONSTRUIRE SON CONTENEUR



A noter :

* Il faut installer docker pour transformer le docker_file en image, ou juste lancer l'image

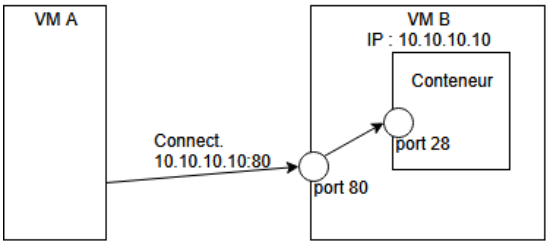
* Dans certains conteneurs, on peut vouloir des services qui tournent en permanence. Mais si c'est juste pour lire un fichier (ex : docker run node app.js) ou pour lancer une seule instance de l'application dessus, on peut rajouter -rm pour que le conteneur crée s'efface après que l'application a l'intérieur aie finie

* Un service dans le conteneur n'apparaitra pas dans la vm, et inversement

INTERACTION DES CONTENEURS

Utiliser -p dans la commande de docker run permet de rediriger les connections qui arrivent depuis un port de la vm, jusqu'a un port du conteneur.

Ex : -p 80:28

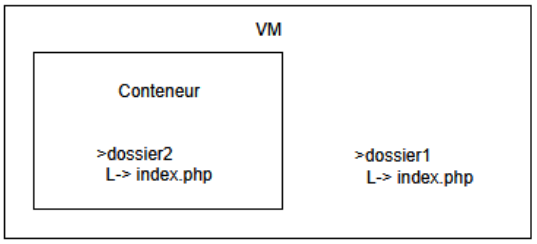


A noter :

On peut aussi configurer les conteneurs pour qu'ils aient une adresse ip et soient comme dans une serveur, a l'interieur de la vm. Cela permet de les faire communiquer entre eux, par exemple pour qu'apache puisse donner a php du contenu a traduire

On peut utiliser -v pour faire pointer des dossiers dans le conteneur (volume monté) , sur des dossiers a l'extérieur (genre dans la vm). Cela permet d'accéder a ces dossier dans le conteneur, et a lancer des processus qui se situent dans le conteur , sur des dossiers a l'extérieur du conteneur, tout en gardant l'isolation des conteneurs

Ex : -v dossier1:dossier2



Les deux index.php sont exactement les mêmes, sauf que le conteneur y a accès, donc si il a un service, on peut l'utiliser. Sans ça, le service dans Conteneur n'aurait jamais pu être utilisé sur l'index.php (puisqu'il n'y a pas accès, le conteneur étant isolé)

DOCKER ET DEPLOIEMENT

Un des points forts de docker, c'est que les conteneurs sont les mêmes, et marchent de la même manière. En clair, si ça marche dans un conteneur, alors ça marchera dans tout les copeis de ce conteneur. De plus, on peut intégrer des logiques différentes dans l'architecture :
* Vue applicative : on crée un conteneur dans lequel on met tout les services pour une application, et on recommence pour chaque application
* Vue service/microservice: on crée un conteneur par service, et les applications peuvent se servir

Dans le cas dafy, on avait avant une vue applicative, et donc plusieurs conteneurs qui possédaient les mêmes services et prenaient de la place. maintenant, chaque conteneur possède son service, et il est rare de voir des conteneurs avec plusieurs services, sauf cas exceptionnel (le conteneur de déploiement, par exemple, a besoin de l'application de déploiement custom pour récupérer les derniers maj du repo distant git, faire les installations npm, etc. Mais elle a aussi besoin d'un service php pour pouvoir lire les instructions. On pourrait cependant mettre l'application docker dans le conteneur du service php, pour éviter d'avoir deux php a deux endroits).

Aujourd'hui, la vm web ne contient véritablement que les différentes versions de l'application (jusqu'à 3 en arrière, contre 10 dans la vm dev), et un current qui désigne la version en cours, afin que même lors du déploiement de nouvelles versions le site soit toujours en ligne (c'est du déploiement en 0 downtime). Tout le reste, ce sont des conteneurs qui ont leurs ports de redirection, des adresse ip pour communiquer entre eux , et un conteneur de déploiement qui est déclenché par l'agent azure lorsqu'il y a une Maj sur le repo distant git (déploiement continu : 1 commit = 1 déploiement sur le serveur). Dans ce cas, le déployeur récupère les dernières maj, puis installe les dépendances. Ce déploiement est actif sur la branche develop de magento et laravel, et main de laravel..

Les Liens

Site à propos de LE CIRCUIT : ici

Site MesInfos sur l'ouverture de la branche à Sisteron : ici

Données du gouvernement sur l'historique de Team Moto Quad : ici

L'ensemble des travaux, notes, et schémas sont sur un drive qui vous est mis à disposition :
[Juste là]

Sinon, vous pouvez accéder de manière indépendante aux notes :

- Filament
- Api et HFSQL
- Laravel
- Journal de Bord

Merci d'avoir lu mon rapport de stage, et bonne journée !

Favre Sylvain