

METZ NUMERIC SCHOOL

LOT 2

---

# Plateforme de covoiturage

---

*Auteurs :*

Alexandre GERARD

Vincent GIANGRECO

*Suivi de réalisation de la plateforme, d'un plan de sécurisation, et de  
tests / recettage*



30 janvier 2023



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Réalisation</b>	<b>3</b>
2.1	Descriptif du projet . . . . .	3
2.2	Étude du projet . . . . .	4
2.2.1	Gestion de compte . . . . .	4
2.2.2	Gestion des voyages . . . . .	4
2.2.3	Gestion des conducteurs . . . . .	5
2.2.4	Gestion de la messagerie* . . . . .	5
2.2.5	Gestion de l'administration . . . . .	5
<b>3</b>	<b>Sécurité</b>	<b>7</b>
3.1	CORS et en-têtes . . . . .	7
3.2	AWS S3 . . . . .	7
3.3	Gestion des dépendances . . . . .	7
3.4	Données sensibles dans la codebase . . . . .	8
3.5	Protocole HTTPS . . . . .	8
3.6	JWT / Cookie . . . . .	8
3.7	Mots de passe forts . . . . .	9
<b>4</b>	<b>CI/CD</b>	<b>11</b>
4.1	Tests et recettage . . . . .	11
4.1.1	Cahier de recette . . . . .	11
4.1.2	Tests unitaires . . . . .	11
4.1.3	Tests d'implémentation . . . . .	11
4.1.4	Tests E2E . . . . .	12
4.1.5	Validations . . . . .	12
4.1.6	Tests GitHub Actions . . . . .	12
4.2	Déploiement continu et maintenance . . . . .	13
4.2.1	Déploiement . . . . .	13

4.2.2	Cloisonner les environnements . . . . .	13
<b>5</b>	<b>Suivi de gestion de projet</b>	<b>15</b>
5.1	"Clean Code" . . . . .	15
5.1.1	"Pull request", assignation et revue de code . . . . .	15
5.1.2	Messages de "commit" . . . . .	15
5.1.3	Convention de nommage . . . . .	16
5.1.4	Une structure solide et rigoureuse . . . . .	16
5.1.5	Refléchir avant d'agir . . . . .	17
5.1.6	Restrictions Eslint . . . . .	17
5.1.7	Utilisation des bons statuts . . . . .	17
5.1.8	Commenter, avec parcimonie . . . . .	18
5.2	Comparaison avec la maquettes . . . . .	18
5.3	Difficultés . . . . .	18
<b>6</b>	<b>Bilan</b>	<b>19</b>
6.1	Apports . . . . .	19
6.2	Perspectives . . . . .	19
<b>Annexe</b>		<b>21</b>
A.1	Statistiques du repository Github . . . . .	23
A.2	CORS S3 . . . . .	24
A.3	Permissions IAM . . . . .	25
A.4	Diagramme de séquence : JWT . . . . .	26
A.5	Exemple d'un cahier de recettes (landing page) . . . . .	27
A.6	Exemple d'un schéma de validation (Drivers) . . . . .	28
A.7	Structure de client et de server . . . . .	29
<b>Bibliographie</b>		<b>31</b>

# Introduction

Après avoir choisi le projet de construction d'une plateforme de covoiturage, nous avons passé un moment non négligeable à se préparer à la réalisation de cette dernière. Avant de commencer la réalisation d'un point de vue technique, nous avons donc déjà commencé par mettre en place tous les éléments du premier lot, c'est-à-dire :

- Les besoins et objectifs du projet ;
- La mise en œuvre d'une méthodologie vis-à-vis des outils et de la structure de la plateforme ;
- La modélisation par une maquette et des diagrammes de séquences.

Ce lot 2 a donc pour objectif principal de faire un suivi de la gestion de projet préétablie lors de la réalisation de la plateforme. En plus de cela, ce rapport indiquera les moyens de sécurisation mis en place ainsi que les tests / maquetages présents.



# Réalisation

La partie réalisation s'est étendue sur plusieurs mois : de mi-juillet jusqu'à fin septembre. De la configuration de l'environnement au déploiement de notre site en ligne, en passant par le développement des fonctionnalités au niveau du back-end et du front-end, cette plateforme compte à l'heure actuelle plus de 700 commits ([A.1](#)) et une quarantaine de mises en productions.

## 2.1 Descriptif du projet

VroomMates, notre plateforme de covoiturage, est destinée à toute personne recherchant un trajet à partager, d'un point A à un point B, à une date donnée. Comme mentionné dans le rapport du premier lot, notre application a un but d'internationalisation : elle a donc été traduite intégralement en anglais, la langue universelle, afin d'être compréhensible par la majorité des personnes dans le monde. De plus, nous avons décidé de nous concentrer sur la partie mobile, car c'est pour nous le cas d'utilisation premier.

Sur la plateforme, il existe quatre types d'utilisateurs, chacun avec son propre accès aux différentes fonctionnalités disponibles :

**Visiteur** On définit un visiteur comme toute personne non-connectée sur la plateforme. Il peut concrètement visiter les pages où il n'est pas demandé d'authentification, comme la recherche et les détails d'un voyage. Cependant, il lui sera impossible d'interagir davantage avec la commande, le voyage ou le conducteur. Bien sûr, ce dernier aura accès aux termes et conditions, à l'accueil ainsi qu'aux pages d'inscription et de connexion.

**Passager** On considère un passager comme tel à partir du moment où le visiteur a créé son compte et s'est connecté à celui-ci. Dès lors, il lui est possible tout un tas de nouvelles fonctionnalités, comme pouvoir réserver un voyage ou encore noter le conducteur avec qui il a voyagé. De plus, il pourra renseigner certains champs de son profil, tout en changeant par exemple sa photo de profil, ses préférences de conduite, etc. Le passager aura un accès aux notes précédemment données et celles reçues, ainsi qu'à son historique de trajets. Il pourra aussi, après une confirmation manuelle des administrateurs du site, pouvoir devenir conducteur.

**Conducteur** En plus de toutes les fonctionnalités précédemment énumérées, un conducteur peut créer ou encore gérer ses voyages. Ces utilisateurs seront indiqués d'un petit volant de voiture à côté du prénom.

**Administrateur** L'administrateur a la possibilité en plus d'afficher une liste d'utilisateurs et de les rechercher en fonction de leur nom et / ou prénom. En tant que "entité de modération", il leur est possible de bannir/débannir un membre. Une fois un compte banni, il ne sera plus possible d'interagir avec des trajets ou des utilisateurs.

## 2.2 Étude du projet

Cette partie a pour but de revenir sur les fonctionnalités de la plateforme. Au vu du nombre conséquent de features, nous allons segmenter l'étude de celles-ci en fonction du milieu opérant.

*NOTE : Toutes les parties ou features précédant un astérisque n'ont pas encore été réalisés / mises en production.*

### 2.2.1 Gestion de compte

Les visiteurs ont tous accès aux pages d'inscription et de connexion. Lors de la première connexion, l'utilisateur sera invité à entrer des informations complémentaires et facultatives via un modal. Finalement, l'utilisateur sera redirigé vers la page d'accueil. Dès lors, l'utilisateur pourra consulter sa page de profil et effectuer des modifications sur son compte. Si l'envie lui prend, ce dernier a la possibilité de se déconnecter voire de clore son compte.

### 2.2.2 Gestion des voyages

Concernant les voyages, n'importe quel type d'utilisateur a la possibilité d'effectuer une recherche de voyages en fonction :

- Du type (trajet simple, trajet régulier) ;
- De l'adresse de départ ;
- De l'adresse d'arrivée ;
- De la date de départ.

Dès lors, une liste de voyages sera proposé. Chaque voyage est donc clickable pour accéder aux détails. De cette page, en fonction de l'utilisateur, plusieurs actions peuvent apparaître. C'est donc d'ici que l'on peut s'ajouter / se retirer de la liste de passagers, ou retirer le voyage. Finalement, on peut noter une page de gestion des



voyages : une fois la participation confirmée, chaque voyage est répertorié dans une page énumérant les voyages créés, les voyages en cours et les voyages terminés.

### **2.2.3 Gestion des conducteurs**

Un utilisateur peut, du menu, faire sa demande pour devenir conducteur. Un modal s'ouvre et demande de fournir le permis de conduire et des informations sur la voiture du futur conducteur. Les informations seront envoyées dans la base de données et le document sera stocké sur le serveur, le temps de la demande. Une fois devenu conducteur, ce dernier a la possibilité d'ajouter / de retirer un voyage.

### **2.2.4 Gestion de la messagerie\***

La messagerie comprend les chats ainsi que les notifications. Pour les chats, il est possible de communiquer entre utilisateurs de façon instantanée. En terminant un voyage, un modal propose au passager de noter le conducteur. Concernant les notifications, chaque action du site (nouveau message non lu, nouveau passager s'ajoutant à son voyage, réponse de la demande de conducteur, ...) notifie l'utilisateur concerné.

### **2.2.5 Gestion de l'administration**

Les administrateurs ont plusieurs pages et fonctionnalités leur étant dédiées. Ces derniers ont premièrement accès à une simple page de statistiques\*, purement visuelle. Il existe ensuite une page qui liste tous les utilisateurs de la plateforme et enfin, la page de demande de conduite\*. On note aussi la possibilité de bannir/débannir les utilisateurs (non-administrateurs) et la possibilité de suppression de voyages.



# Sécurité

Conformément aux nombreuses recherches faites (on notera parmi elles le "OWASP top ten" [1]), nous avons réalisé une check-list exhaustive du sujet de la sécurité. Cette check-list sera, pour cette partie, séparée en plusieurs groupes :

## 3.1 CORS et en-têtes

Pour le back-end, le front-end et les services externes, **une forte politique CORS** a été mise en place.

De plus, il est nécessaire de passer à chaque requête (hormis quelques exceptions, comme le login) le token dans le header (**x-access-token**).

## 3.2 AWS S3

Initialement, les permis de conduire ainsi que les photos de profil devaient être stockés sur un serveur européen AWS S3 (eu-central-1). Cependant, lors de la sou-tenance du lot 1, il a été soulevé que le stockage des données sensibles (dans ce cas le permis de conduire) dans S3 peut être entièrement évité si ces dernières sont stockées directement sur le serveur, le temps de la vérification. Nous nous sommes tournés vers ce système.

Néanmoins, le stockage des photos reste tout de même idéal dans un bucket. Il a fallu donc paramétrer correctement les autorisations et accès accordés. En plus de la configuration CORS (→ A.2), vient donc s'ajouter la mise en place de policies avec AWS IAM. Nous avons verrouillé l'accès au bucket S3 en créant un profil utilisateur pour VroomMates qui va uniquement autoriser la lecture, la modification et la suppression des données (A.3). L'utilisateur IAM aura désormais une clef d'accès unique à passer lors d'une interaction avec S3 (stockée dans avec les variables d'environnement).

## 3.3 Gestion des dépendances

Nous utilisons des dépendances dans le back-end et le front-end. Afin d'éviter toute question de performance et de sécurité, nous avons décidé de mettre en place une maintenance des dépendances. Heureusement pour nous, sur GitHub, **Dependabot** nous permet de gérer au mieux ces problèmes.

### 3.4 Données sensibles dans la codebase

Pour éviter toute fuite de données sensibles de la part du développeur (ex. fuite de Keys), notre code contient toutes les clefs et mots de passe dans les fichiers `.env`. De plus, le code est continuellement scanné par **Github defender** qui nous signalera de toute faute de ce calibre.

### 3.5 Protocole HTTPS

Le client et l'api sont tous deux sécurisés avec le protocole HTTPS. Là où Heroku gère automatiquement le HTTPS, Netlify était un peu plus complexe à mettre en place. Les sous-domaines netlify utilisent par défaut https. Cependant, nous avons utilisé une adresse personnelle (agerard.dev). Il a donc fallu gérer la création du certificat TLS/SSL (via **Let's Encrypt**)

### 3.6 JWT / Cookie

Un système d'authentification et d'accès aux ressources par token a été mis en place. On utilise des token JWT encryptés en HS256 avec chacun son propre passphrase secret, afin d'éviter toute faille. Lors de l'authentification, l'api va donc générer deux tokens :

- Le premier est un token d'authentification, contenant la meta du token (date d'expiration, de création, ...) ainsi que le strict minimum de payload (id, nom, url de photo, type d'utilisateur, ...). Ce dernier ne contient aucune information sensible puisque ce qu'il sera retourné au client est stocké dans un cookie (avec les flags secure et http only). Sa durée de vie varie entre la durée de la session et 15 minutes, en fonction des choix de connexion de l'utilisateur (case "Se souvenir de moi").
- Le second jeton est un token de rafraîchissement. Le client n'aura jamais accès à ce token et a une durée maximale de 1 mois avant qu'une tâche CRON supprime ce dernier de la base de données.

Chaque appel d'api privée nécessite d'envoyer ce jeton, sans quoi, la ressource est bloquée. Quand un jeton envoyé à une route expire, des intercepteurs Axios vont déclencher le rafraîchissement du token avec le second token dans la base de données ; Si la manoeuvre rafraîchit bien le token client, Axios retente la requête précédente avec le nouveau jeton.

Diagramme de séquence du process : [A.4](#)

## 3.7 Mots de passe forts

Notre site force une certaine complexité à l'utilisateur, ce qui restreint l'utilisateur à générer un mot de passe de minimum 10 caractères, avec minimum une minuscule, une majuscule, un chiffre et un caractère spécial lors de la création de son compte. nous demandons de rentrer une seconde fois le mot de passe afin de le confirmer. Dès lors, les mots de passe sont cryptés avec le module bcrypt qui va saler plusieurs fois et hasher le mot de passe avant de le stocker dans la base de données.



# CI/CD

## 4.1 Tests et recettage

Réaliser des tests pour l'application permet entre autres de ne pas perdre certaines fonctionnalités à cause d'un développement annexe, cela permet de vérifier constamment la disponibilité de celles-ci. Nous avons réalisé plusieurs séries de tests à plusieurs niveaux différents.

### 4.1.1 Cahier de recette

Nous avons fait un cahier de recette sous forme de fichier Excel ([A.5](#)). C'est ici que nous avons regroupé la majeure partie des features de l'application, les soucis de performance, les éventuels dysfonctionnements ou encore les fautes d'orthographe ou de traduction. Nous avons aussi réalisé manuellement le test de nos fonctionnalités (comme un utilisateur aurait pu le réaliser). C'est une pratique que nous avons pu voir en entreprise, surtout lors de la recherche à la reproduction d'un dysfonctionnement. Ce fut, tout au long de notre projet, un réel cahier de bord, facilitant ainsi la visualisation des tâches effectuées, ou de ce qu'il nous restait à faire.

### 4.1.2 Tests unitaires

Tout d'abord, les tests unitaires. Ce sont des test qui permettent de vérifier le bon fonctionnement d'une partie précise et définie de notre plateforme, comme par exemple la bonne traduction de celle-ci au moment du changement de langue, ou les tests de conversion de miles en kilomètres. Pour cela, nous avons décidé de mettre en place Jest : c'est un framework de test JavaScript. Nous l'avons couplé à Babel afin de définir au mieux sa scope d'utilisation, comme les fichiers à ne pas transpiler.

### 4.1.3 Tests d'implémentation

A la fin de chaque feature, on effectue des tests d'implémentation afin de savoir si en plus des features testées individuellement, ces dernières fonctionnent bien entre-elles et celles pré-existantes.

#### 4.1.4 Tests E2E

Les tests E2E consistent à vérifier que le visiteur puisse parcourir les principaux scénarios d'utilisation de l'application. Pour se faire, nous avons configuré "puppeteer" afin de lancer automatiquement une instance headless de chromium (sans interface graphique) pour qu'il puisse tester certaines fonctionnalités. De plus, des tests de non-régression ont été mis en place afin de vérifier, en comparant avec un "snapshot" du résultat attendu, le bon fonctionnement des pages. Un "snapshot" est un fichier où l'entièreté du contenu d'une certaine page est enregistré en format JSON.

#### 4.1.5 Validations

Chaque table de notre base de données est strictement définie avec des patterns de validation de schéma. Cela permet d'être sûr qu'un format respecte les attentes avant d'insérer un document dans la base de données. -> [A.6](#)

#### 4.1.6 Tests GitHub Actions

GitHub Actions constitue une sécurité supplémentaire avant un merge. En effet, à chaque commit, des tests automatisés se lancent afin de tester la mise en forme du code (avec eslint et prettier). Ces tests sont fait côté back-end et côté front-end.

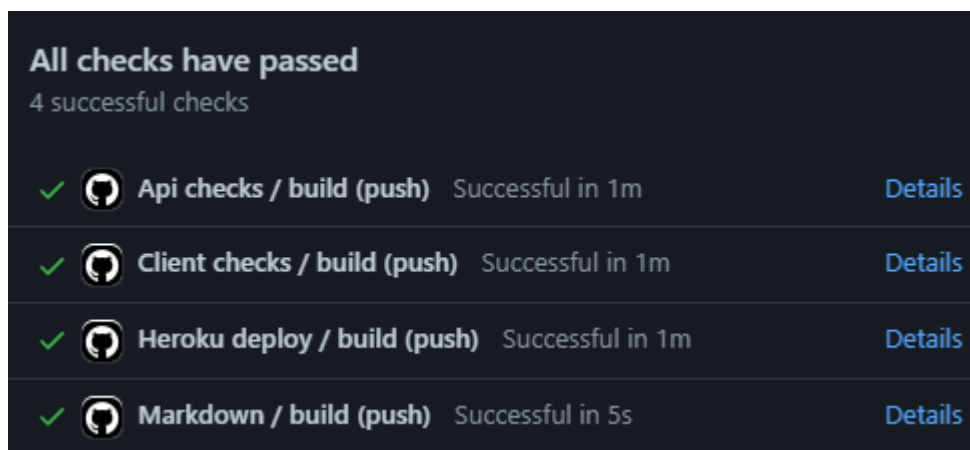


FIGURE 4.1 – Checkmark de Github Actions

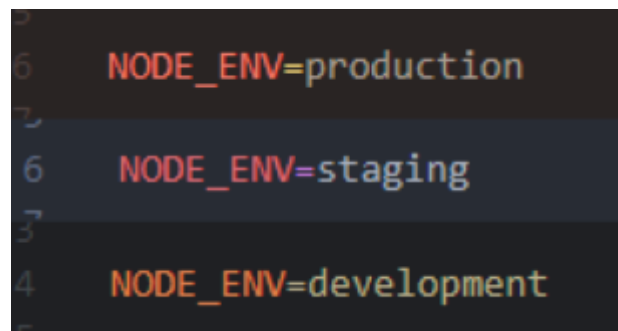


## 4.2 Déploiement continu et maintenance

### 4.2.1 Déploiement

Nous avons déjà parlé de notre méthode de déploiement dans le lot 1 ainsi que dans 3.5. Nous avons bien mis en place et suivi notre méthode de déploiement du serveur et du client tout au long de notre réalisation de projet. Pour rappel, l'api est accessible à l'adresse <https://vroommates-agerard57.herokuapp.com> et le client se situe sur <https://www.vroommates.agerard.dev>.

### 4.2.2 Cloisonner les environnements



```
6 NODE_ENV=production
6 NODE_ENV=staging
4 NODE_ENV=development
```

FIGURE 4.2 – Exemple d'un fichier .env sous 3 environnements différents (dev / staging / production)

Nous avons cloisonné les différents environnements pour chaque partie de la plateforme. Au build du code, ce dernier détecte l'environnement sous forme de variable stockée dans les deux fichiers **.env** du projet (un dans /server, un dans /client).

Il existe au total 3 environnements différents :

- Nous avons donc l'environnement de "**development**", qui correspond à la version locale du code. Il s'agit de la version que les développeurs utilisent, afin d'avoir accès aux logs et aux dépendances de développement.
- Ensuite, l'environnement intermédiaire de "**staging**", correspondant à la branche develop sur Github. C'est ici que l'on teste la compatibilité de nos nouvelles features avant de les publier. Avec chaque pull request de staging vers master, on génère une version test "en ligne", afin de tester dans des conditions se rapprochant plus de l'environnement d'un utilisateur. Cette méthode s'est avérée très efficace pour détecter des bugs ou des erreurs seulement remarquables dans ces conditions.

- Enfin, une fois que nous sommes sûrs qu'il n'y ait ni bugs ni régressions, on déploie notre site, avec l'environnement de "**production**", correspondant à la branche master.

# Suivi de gestion de projet

Suite à la mise en place du diagramme de Gantt, nous avons respecté minutieusement ce dernier afin de suivre le chemin critique et proposer une version de la plateforme qui soit fonctionnelle, dans les temps. Nous pensons qu'avec quelques méthodes de discipline simples et efficaces, la production d'un travail conséquent concret en un temps restreint devient réaliste. Aussi, des méthodes Agile ont été mises en place afin d'avoir une meilleure communication sur les points essentiels et les besoins de notre application et ainsi avoir un meilleur suivi personnel. Pour respecter ces méthodes, nous nous devons de nous auto-imposer ces quelques règles.

## 5.1 "Clean Code"

### 5.1.1 "Pull request", assignation et revue de code

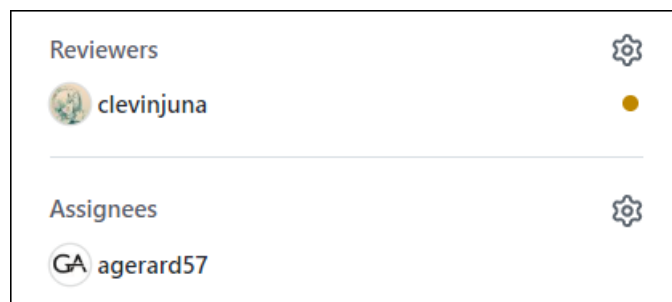


FIGURE 5.1 – Exemple de demande de review d'une pull request (Github)

Nous créons une "pull request" (ou PR) à chaque branche avec une nouvelle feature/fix/refacto/..., avec une description de la feature. Une revue de code consiste à étudier les changements effectués dans une branche d'un membre de l'équipe. Cette opération permet d'avoir un oeil neuf sur le code proposé afin de mutuellement s'améliorer.

### 5.1.2 Messages de "commit"

Tous les messages de commit sont normalisés selon **Conventional Commits** [2]. Nous suivons ce pattern afin de mieux comprendre le type / l'utilité / le scope du commit. La transparence est ici vitale pour comprendre plus facilement le code et faciliter le travail des reviewers dans la PR.

### 5.1.3 Convention de nommage

Un nommage clair des fichiers, des fonctions et des variables dans le code permet de rendre compréhensible un élément du code au premier coup d'oeil, sans pour autant avoir à documenter et expliciter ce que chaque partie du code accomplit. Par ailleurs, JavaScript et TypeScript obligent, nous avons normalisé le code en "camelCase" pour tout le projet, sauf pour tout ce qui concerne la base de données (snake\_case). Les fonctions ont un contexte plus verbeux, ce qui signifie que tout le temps de réflexion et d'écriture supplémentaire ont pour effet de bord de faire gagner du temps en termes de compréhension ultérieure du code. Enfin, on note que toute la codebase est en anglais (hormis ces rapports de lots).

### 5.1.4 Une structure solide et rigoureuse

La rigueur du code et son uniformisation permet de se repérer au mieux. En règle générale, nous faisons en sorte de segmenter notre code de telle façon à ne pas dépasser les 200 lignes par fichier. Pour des cas plus précis :

**/client** Toute notre partie front-end est organisée dans le dossier `/src` par packages. Ces derniers contiennent tous plus ou moins la même structure. Cette architecture en packages nous permet de ranger nos imports et exports grâce à la "barrel method" : il s'agit de fichiers index présents dans chaque dossier qui va nous permettre d'y réunir tous nos imports/exports. Lors-ce qu'un composant d'un autre package aura besoin d'un de ces composants exportés, il aura juste à importer sans préciser le fichier en particulier. Cette implémentation est enforced par eslint, mais nous y reviendrons plus tard (5.1.6). Enfin, l'utilisation de TypeScript nous oblige à typer l'intégralité de notre code, demandant donc de la régularité. → [A.7](#)

**/server** Notre back-end a une structure similaire au front, en partie due aux similarités entre JavaScript et TypeScript. Les modèles, contrôleurs, services, routes, etc... sont rangés dans leur packages respectifs. Chaque fichier dans un package correspond à une feature précise. → [A.7](#)

**/scripts** Ce folder met à disposition des scripts python permettant d'automatiser et donc faciliter nos actions et opérations, comme par exemple la création d'un package template pour client.

**/database** Ce folder est destiné à un usage exclusif aux développeurs, il sert à générer une base de données locale pour mongoDB (afin de faire les tests en environnement local). Le fichier "index.js" va donc générer la DB, tandis que les fichiers dans /generate vont la peupler avec de la de mock data (en créant par exemple un document Users).

**/docs** cette partie comprend notre rapport du lot 1, celui du lot 2 et la structure des documents de la base de données. En ce qui concerne la documentation, nous avons aussi donc des commentaires directement dans le code comme mentionné précédemment, mais aussi de la documentation inline sur certaines fonctions indiquant les paramètres, le type attendu ou encore possiblement le comportement de celle-ci.

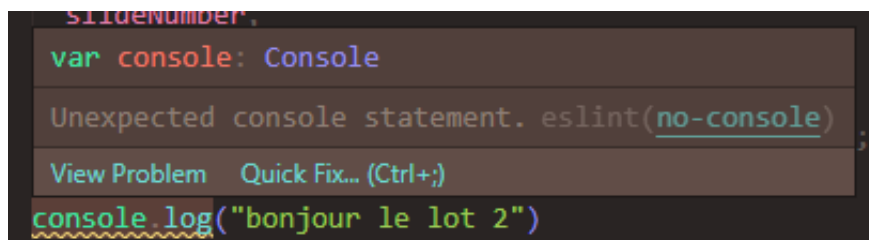
Pour les développeurs, il est mis en place un fichier VS Code **.workspace** à la racine du repository qui va agencer les différentes parties explicités plus haut.

### 5.1.5 Réfléchir avant d’agir

Un moyen simple mais efficace d’écrire du code convenablement. Évidemment, il se peut que le code ne soit pas parfait, même après une partie conséquente de réflexion. C’est pour cela qu’il faut donc passer assez souvent par une étape de refactoring. Ce concept consiste à retravailler le code de l’application, sans pour autant ajouter de nouvelles fonctionnalités, ni en corriger les éventuelles défaillances.

### 5.1.6 Restrictions Eslint

Les restrictions d’Eslint imposent d’avantage de normalisation dans la codebase. Nous avons défini quelques règles, comme par exemple l’impossibilité de mettre des "console.log()", l’obligation de ne pas mettre d’accolades dans une fonction ou une condition "if" avec un seul retour, ou l’obligation de camelCase.



### 5.1.7 Utilisation des bons statuts

A chaque retour d’appel d’api, on envoie le bon statut en fonction de la demande : 200 pour une requête "get", 204 pour "put", 201 pour un "post", 401 pour un "non autorisé", etc.

### 5.1.8 Commenter, avec partimonie

Placer des commentaires dans le code est une bonne source de documentation rapide expliquant par exemple le fonctionnement d'un petit bout de code, son résultat attendu, quelques erreurs qui pourraient arriver à cause de celui-ci, etc. Dans notre cas, dû à la clarté du code, peu de commentaires ont été nécessaires. Cependant, il ne faut pas oublier de mentionner certains commentaires très utiles qui ont quand-même trouvé leur place dans notre code, comme ceux qui vont vulgariser des opérations contenant plusieurs opérations "ET" / "OU".

## 5.2 Comparaison avec la maquettes

Lorsque nous visualisons le travail réalisé avec celui que nous avions prévu de faire, nous pouvons nous rendre compte que le résultat actuel de la plateforme coïncide grandement. Tout ce qui nous a été demandé en priorité a plus ou moins pu être établi (hormis la messagerie et des pages peu lourdes en tâches). Le style graphique et les placements des éléments dans le DOM tient exactement de ce que nous avions prévu et beaucoup de pages ressemblent traits pour traits à la maquette. De cette réalisation naît tout de même une certaine satisfaction personnelle sur cet aspect du travail.

## 5.3 Difficultés

Dû à des complications, un manque de temps nous aura impacté. Ce projet n'était tout d'abord pas la seule tâche qui nous était demandé, ceux à quoi il a fallu rajouter nos contrats en entreprise. Une gestion du temps qu'il nous a fallu gérer.

Certaines features ont forcément posé plus de soucis que d'autres : on retient respectivement, l'initialisation de l'environnement de dev, des tests et la version desktop du site pour Vincent et l'intégration du système d'authentification par jetons et toute la partie déploiement des plateformes (CI/CD) pour Alexandre.

Enfin, en quantifiant les features manquantes (version desktop, page de paiement, messagerie, pages administrateur, certains tests), on pourrait estimer que quelques sprints supplémentaires de travail d'environ 1 semaine (soit 2-3 semaines) seraient suffisants pour **entamer totalement la backlog actuelle**.

# Bilan

## 6.1 Apports

Finalement, ce projet a été très riche en connaissances acquises. Pour Vincent, travailler sur une telle architecture avec des langages comme TypeScript, une base de données NoSQL, ou encore toute la partie développement de site internet était une grande première. C'est une grande plus-value, étant donné que pour sa prochaine alternance, de l'expérience en React lui sera demandé. Pour Alexandre, travailler par exemple sur des services comme AWS lui sera utile pour sa prochaine année d'alternance en tant que développeur sur ces mêmes services. D'un point de vue plus général, les deux lots avaient tous deux quelque-chose à apporter, non seulement en terme technique, mais aussi en termes de gestion / conception de projet. Nous sommes tous deux très fiers du travail qui a été réalisé, mais aussi dans la manière dont s'est déroulé l'achèvement de celui-ci.

## 6.2 Perspectives

Si le projet dans le cadre "scolaire" est bien terminé, pour autant, nous souhaiterions continuer le projet après coup afin d'arriver au résultat que nous concevions initialement. La mise en place des fonctionnalités manquantes établies dans [5.3](#) reste à faire afin d'arriver à un résultat qui nous semblera entièrement finalisé, ainsi nous laissant prendre le large sur de nouveaux projets que nous avons d'ores et déjà en vue.

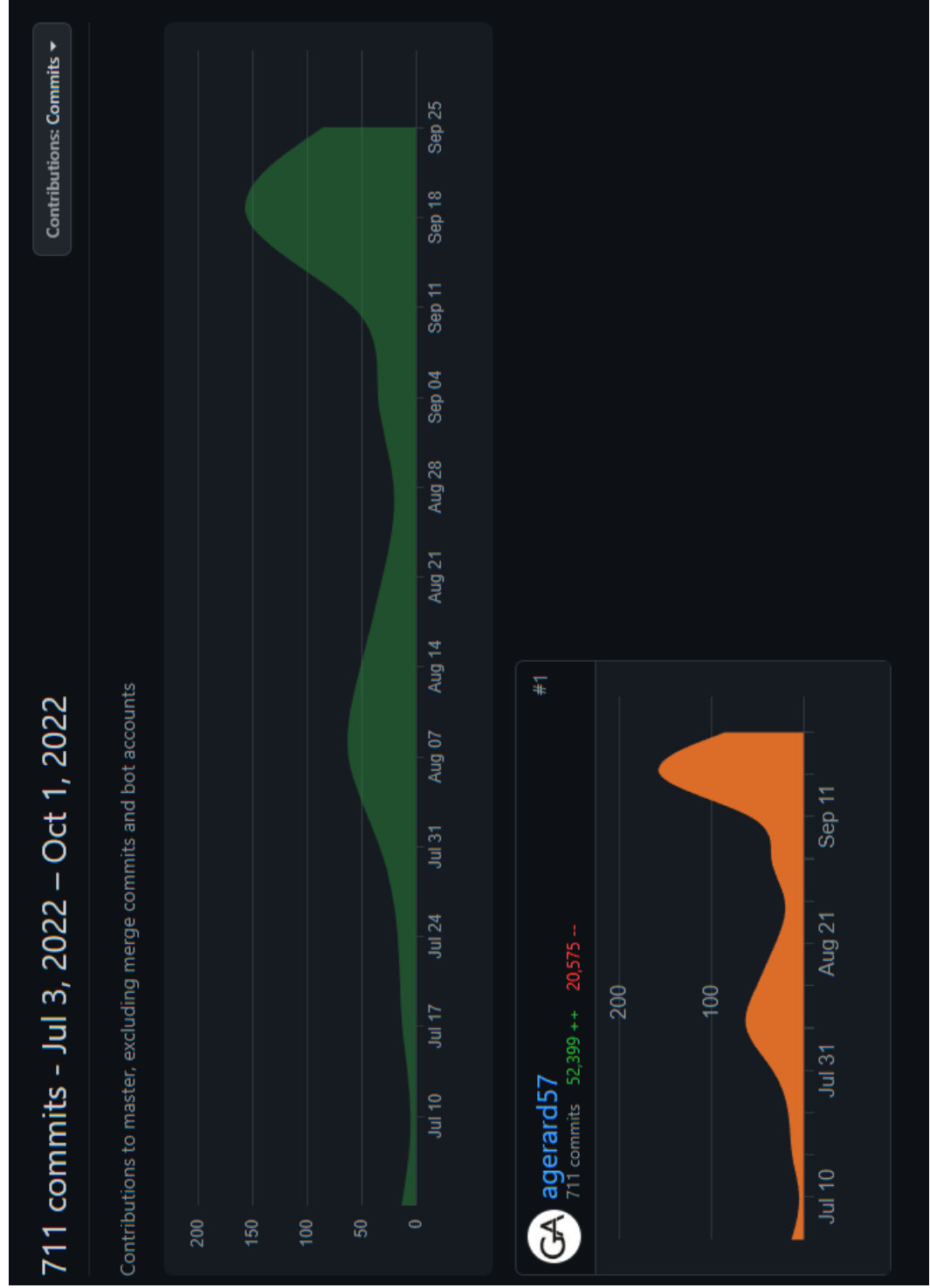




## **Annexe**



## A.1 Statistiques du repository Github



## A.2 CORS S3

### Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. [Learn more](#) 

```
[
  {
    "AllowedHeaders": [
      "x-access-token"
    ],
    "AllowedMethods": [
      "GET",
      "PUT",
      "HEAD",
      "DELETE"
    ],
    "AllowedOrigins": [
      "vroommates.agerard.dev"
    ],
    "ExposeHeaders": [
      "Access-Control-Allow-Origin"
    ]
  }
]
```

### A.3 Permissions IAM

▼ Permissions (1)

Attach this policy to an IAM entity to apply its permissions to the entity. [Learn more](#)

Attach

Detach

Filter: Filter ▼

Q Search

☐

Name ▼

☐

VroomMates

< Back

S3

Policy summary

{ } JSON

Edit policy

Q Filter

Action (2 of 128) Show remaining 126

Resource

Write (2 of 41 actions)

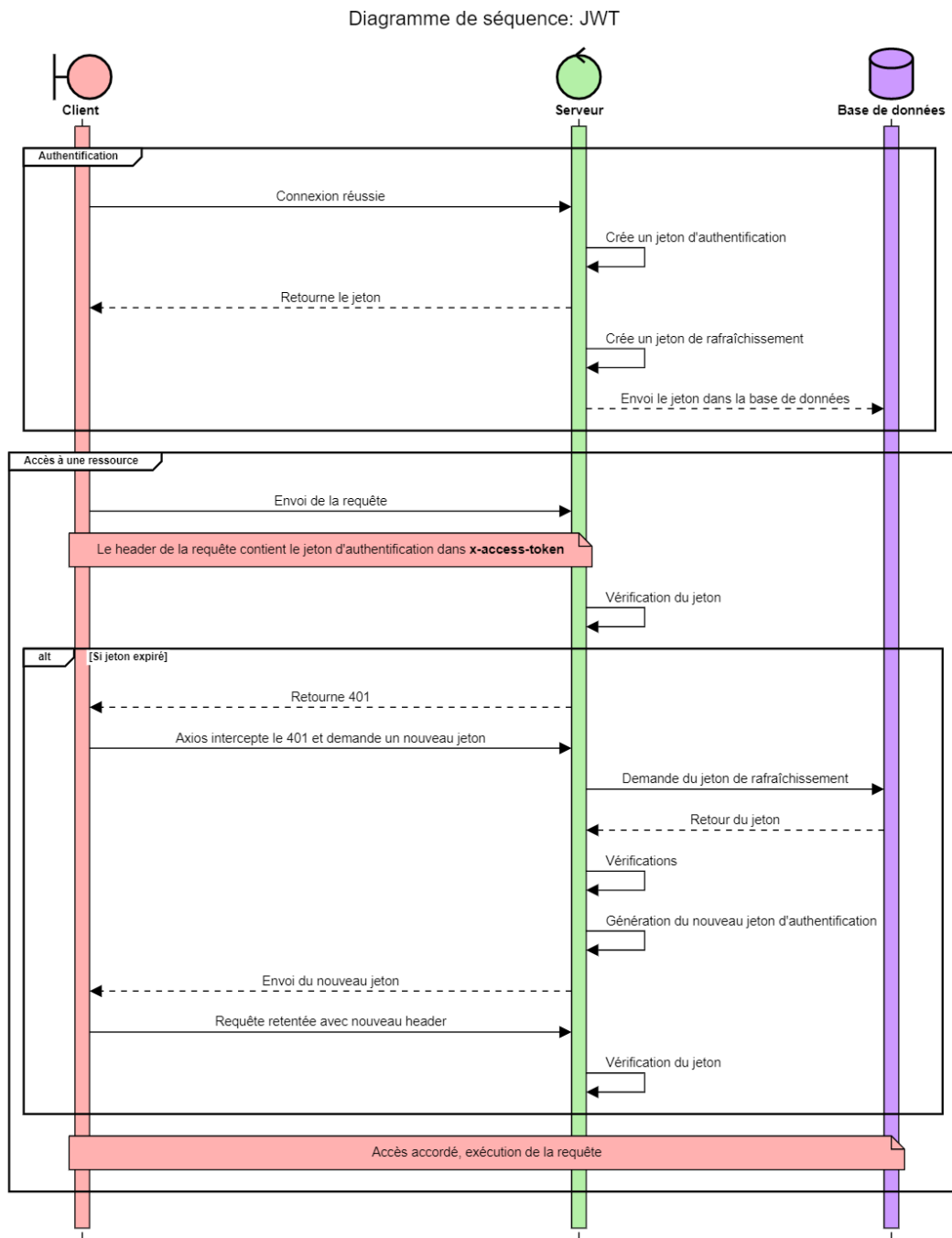
DeleteObject

PutObject

BucketName | string like | vroommates-profile-pictures, ObjectPath | string like | All

BucketName | string like | vroommates-profile-pictures, ObjectPath | string like | All

## A.4 Diagramme de séquence : JWT



## A.5 Exemple d'un cahier de recettes (landing page)

Titre	<b>Landing page</b>
Route	/ ou /home
État	Faite ▼
Commentaires	n/A

Affichage de la page	<input checked="" type="checkbox"/>
Affichage du contenu statique	<input checked="" type="checkbox"/>

Feature	Affichage des nombres	Searchbox	Graphiques	Footer
État	Faite ▼	Faite ▼	Faite ▼	Faite ▼
Comportement attendu	Affiche les nombres	Sélection des types, bouton déverrouillé quand rempli, redirection recherche	Affichage des graphiques	Switcher la langue, boutons et liens fonctionnels
Problèmes rencontrés				

Tests à faire	Test unitaire			Test E2E
État des tests	En cours ▼	Faite ▼	Faite ▼	En cours ▼
Description du test	Tester la conversion des miles en kilomètres			Clicker sur le bouton et vérifier si la langue change

## A.6 Exemple d'un schéma de validation (Drivers)

**VroomMates.Drivers** 3 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Explain Plan Indexes **Validation**

Validation Action ERROR Validation Level STRICT

```

1 {
2   bsonType: 'object',
3   title: 'Drivers',
4   required: [
5     'user_id',
6     'car',
7     'license_card_filename',
8     'status'
9   ],
10  properties: {
11    user_id: {
12      bsonType: 'objectId'
13    },
14    car: {
15      bsonType: 'object',
16      title: 'car',
17      required: [
18        'type',
19        'model'
20      ],
21      properties: {
22        type: {
23          bsonType: 'string'
24        },
25        model: {
26          bsonType: 'string'
27        },
28        colour: {
29          bsonType: 'string'
30        }
31      }
32    },
33    license_card_filename: {
34      bsonType: 'string'
35    },
36    status: {
37      bsonType: 'string'
38    },
39    request_date: {
40      bsonType: 'date'
41    }
42  }
43 }

```

✓ Sample Document That Passed Validation
✗ Sample Document That Failed Validation

```

_id: ObjectId('6327ec56166e90b2fa27d748')
user: ObjectId('63263fdabb85a73e2267f4e')
status: "pending"
license_card_filename: "71bf82b9e3182b5"
request_date: 2022-09-19T04:13:10.989+00:00
__v: 0
> car: Object

```

No Preview Documents



## A.7 Structure de client et de server

```

└─ client
  │
  │ └─ ...
  │
  │ └─ public
  │   │
  │   │ └─ ...
  │
  │ └─ src
  │   │
  │   │ └─ @types
  │   │   │
  │   │   │ └─ ...
  │   │
  │   │ └─ App.css
  │   │
  │   │ └─ App.test.tsx
  │   │
  │   │ └─ App.tsx
  │   │
  │   │ └─ ${FEATURE}
  │   │   │
  │   │   │ └─ assets
  │   │   │   │
  │   │   │   │ └─ index.ts
  │   │   │   │   │
  │   │   │   │   │ └─ ...
  │   │   │
  │   │   │ └─ components
  │   │   │   │
  │   │   │   │ └─ index.ts
  │   │   │   │   │
  │   │   │   │   │ └─ ...
  │   │   │
  │   │   │ └─ hooks
  │   │   │   │
  │   │   │   │ └─ index.ts
  │   │   │   │   │
  │   │   │   │   │ └─ ...
  │   │   │
  │   │   │ └─ i18n
  │   │   │   │
  │   │   │   │ └─ en.json
  │   │   │   │
  │   │   │   │ └─ fr.json
  │   │   │   │   │
  │   │   │   │   │ └─ index.ts
  │   │   │
  │   │   │ └─ index.ts
  │   │   │
  │   │   │ └─ interfaces
  │   │   │   │
  │   │   │   │ └─ index.ts
  │   │   │   │   │
  │   │   │   │   │ └─ ...
  │   │   │
  │   │   │ └─ services
  │   │   │   │
  │   │   │   │ └─ index.ts
  │   │   │   │   │
  │   │   │   │   │ └─ ...
  │   │
  │   │ └─ tsconfig.json
  │   │
  │   │ └─ ...
  │
  └─ server
    │
    │ └─ ...
    │
    │ └─ index.js
    │
    │ └─ src
    │   │
    │   │ └─ config
    │   │   │
    │   │   │ └─ ...
    │   │
    │   │ └─ controllers
    │   │   │
    │   │   │ └─ ${FEATURE}.controller.js
    │   │
    │   │ └─ middlewares
    │   │   │
    │   │   │ └─ ...
    │   │
    │   │ └─ models
    │   │   │
    │   │   │ └─ ${DB_DOC}.models.js
    │   │
    │   │ └─ routes
    │   │   │
    │   │   │ └─ index.js
    │   │   │
    │   │   │ └─ ${FEATURE}.routes.js
    │   │
    │   │ └─ services
    │   │   │
    │   │   │ └─ ...
    │   │
    │   │ └─ utils
    │   │   │
    │   │   │ └─ ...
    │   │
    │   │ └─ ...
    │
    └─ ...

```



# Bibliographie

- [1] OWASPFUNDATION. « OWASP Top Ten ». In : *OWASP Project* (mai 2017).  
URL : <https://owasp.org/www-project-top-ten/>.
- [2] Damiano PETRUNGARO. « A specification foadding human and machine readable meaning to commit messages ». In : *Conventional Commits* (mars 2018).  
URL : <https://www.conventionalcommits.org/en/v1.0.0/>.

Lot 2, par —

GERARD Alexandre : <https://github.com/agerard57>  
GIANGRECO Vincent : <https://github.com/clevinjuna>