Projet NoSQL

R4.03 – Qualité et non relationnel

LE NY Liam, PASQUIER Augustin

2D1

# Contextualisation et justification du sujet

Comme vu lors des TPs, le NoSQL se révèle être très intéressant pour manipuler des données en quantité massive et ne nécessitant pas de normalisation. Nous avons donc cherché une thématique qui exploite ces conditions.

Dans nos recherches, nous avons récupéré un immense jeu de données à propos de voitures à vendre. Le jeu de données comporte notamment 37 679 entrées, ce qui permet de bien exploiter la massivité des données.

En outre, la thématique des véhicules rend difficile la normalisation des données : de nombreux véhicules du même modèle avec la même couleur et la même motorisation sont présents dans le jeu de données. Il est donc difficile de satisfaire la contrainte d’unicité (surtout avec 37 679 éléments !).

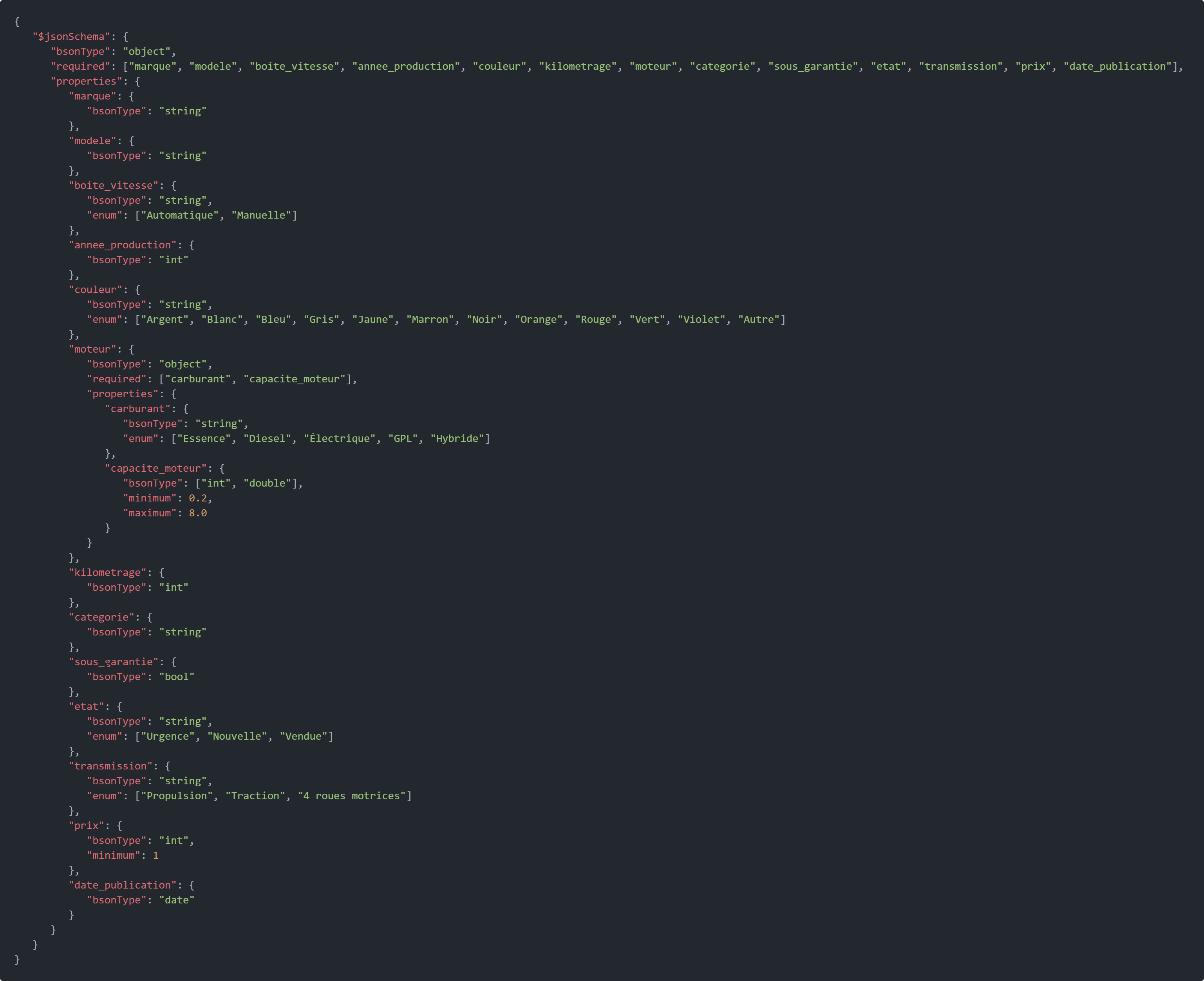
Par ailleurs, les champs renseignés (à savoir : marque, modèle, boite de vitesses, couleur, kilométrage, année de production, carburant, capacité du moteur, catégorie, garantie, état, transmission et date de publication) permettent de bien tirer profit des différents types mis à disposition pour formatter les données : String, Int, Double, Date et Objet (on aura pour cet attribut remanié les données en rassemblant le carburant et la capacité du moteur au sein d’un objet appelé « moteur »).

Sur le long terme, notre jeu de données sera plus facile à maintenir s’il est implémenté au sein d’une base de données NoSQL plutôt que dans un SGBDR. En effet, les opérations d’ajout, de suppression et de modification ne seront pas soumises aux contraintes de clés étrangères.

Dans le cadre du TPs, nous incarnerons donc un concessionnaire automobile qui dispose d’un très gros catalogue.

# Schéma de la base de données

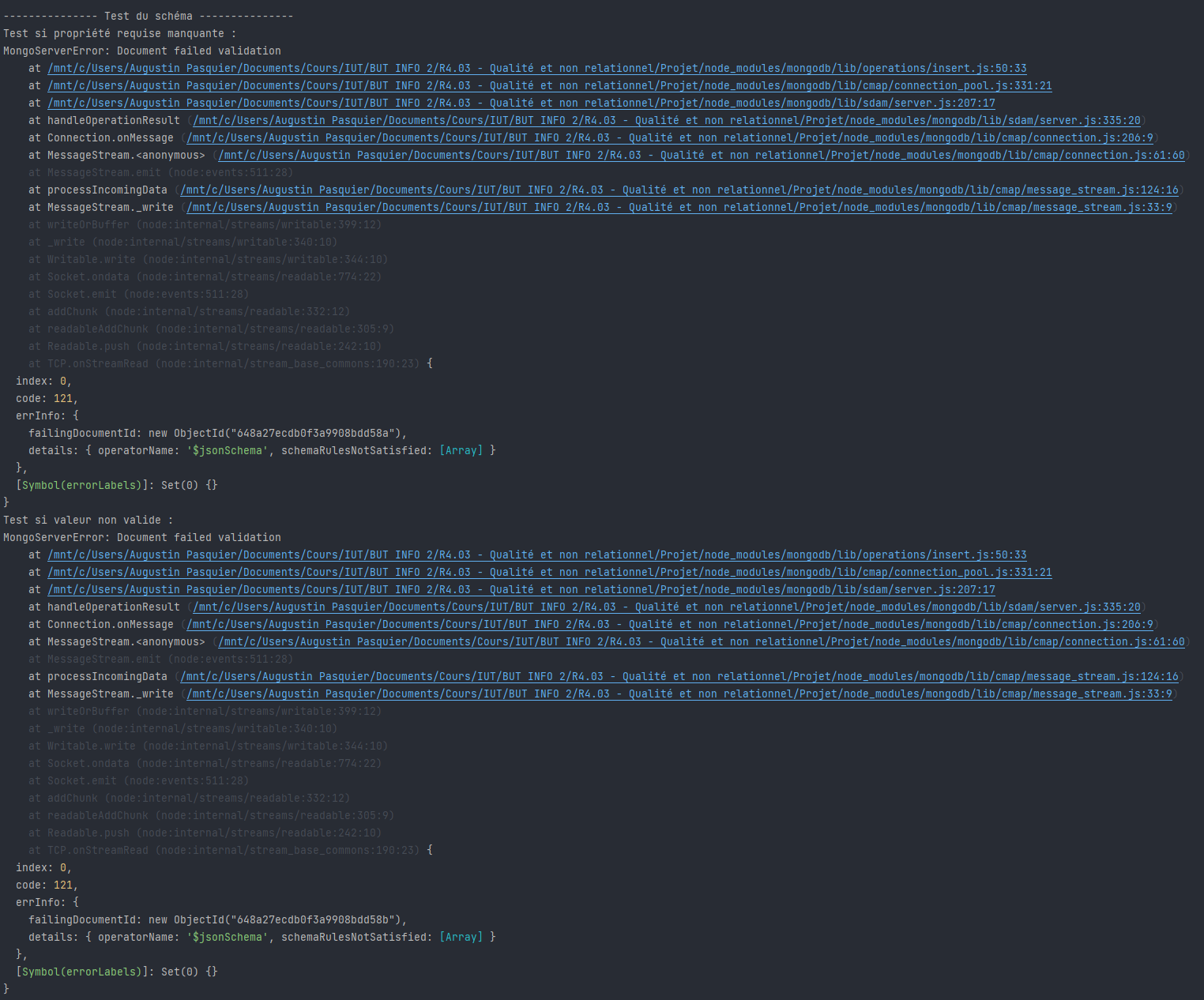
Voici le schéma JSON de la base de données :



# Vérification du schéma



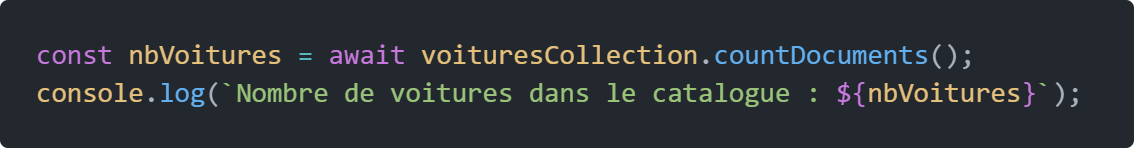
**Résultat :**



Commentaire : On a essayé ici d’insérer des documents invalides ; une erreur est renvoyée et le document n’est pas inséré.

# Requêtes simples

### Combien y a-t-il de véhicules dans le catalogue ?

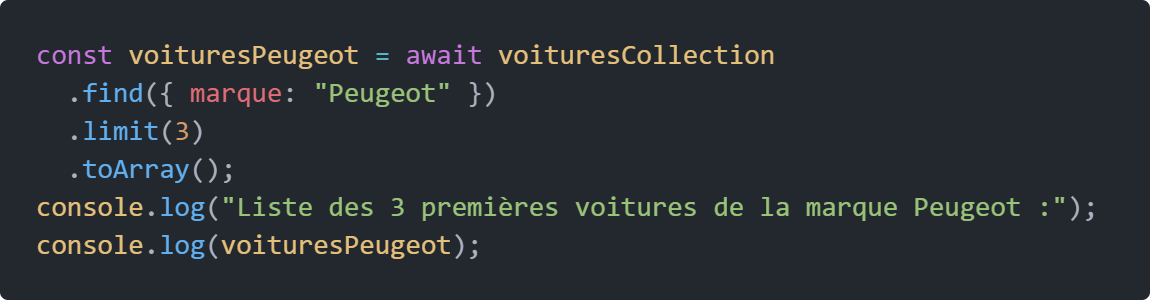


**Résultat :**

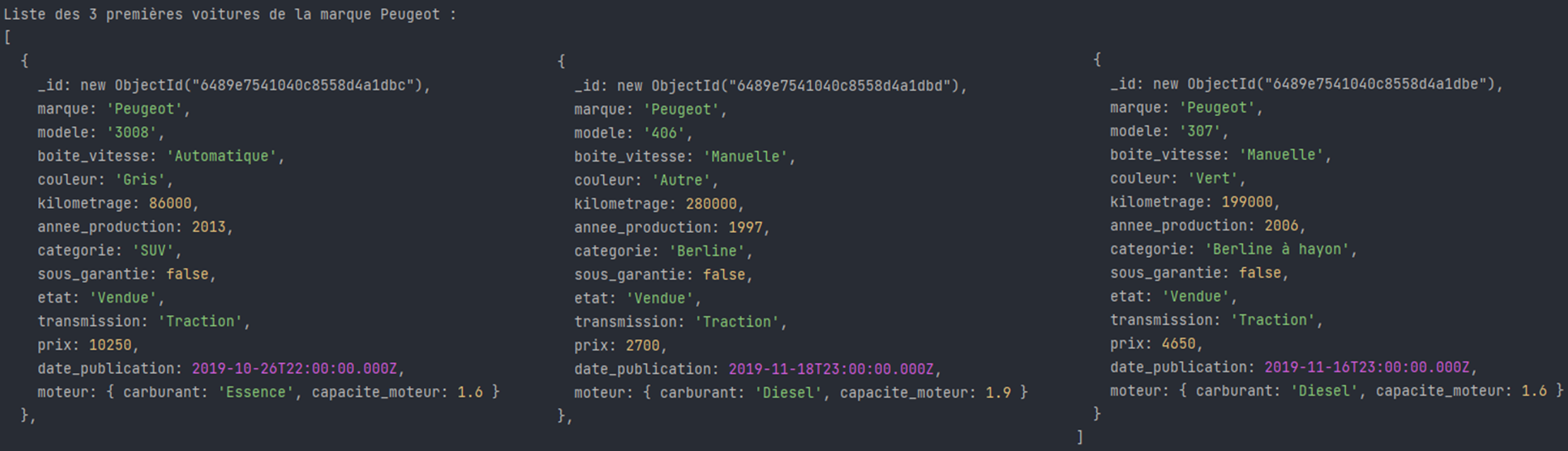


**Commentaire :** La requête est simple, on cherche à savoir le nombre de véhicules (de documents) inscrits dans le catalogue (la collection). On en compte donc 37 679.

### Quels sont les 3 premiers véhicules de la marque Peugeot dans le catalogue ?

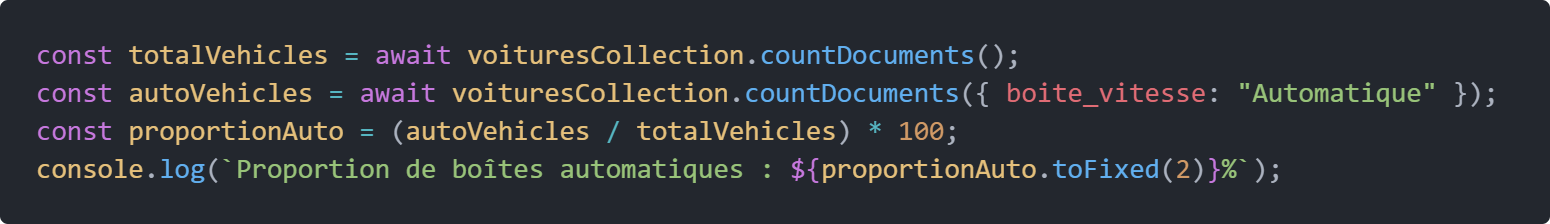


**Résultat :**



**Commentaire :** Là aussi la requête est simple, on se contente de sélectionner les trois premiers véhicules avec une clause de restriction.

## Quelle est la proportion de boites auto par rapport au nombre total de véhicules ?

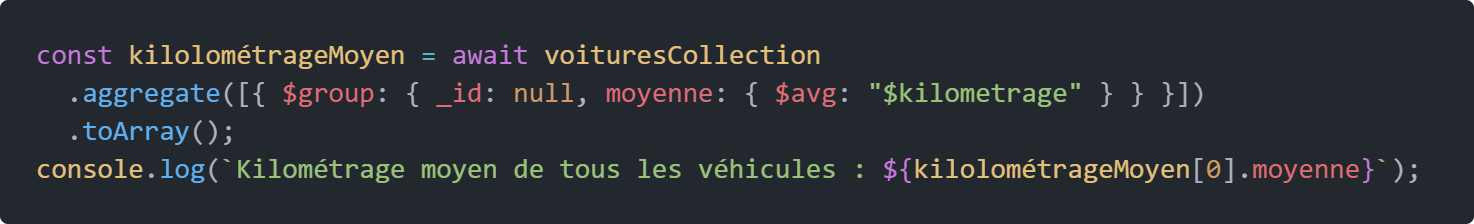


**Résultat :**



**Commentaire :** On prend ici le nombre boites automatiques (restrictions) et le total de véhicules, puis on fait le rapport entre les deux valeurs.

### Quel est le kilométrage moyen du catalogue ?

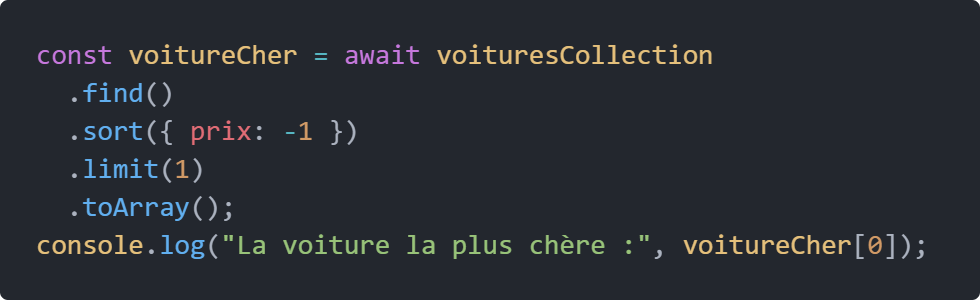


**Résultat :**

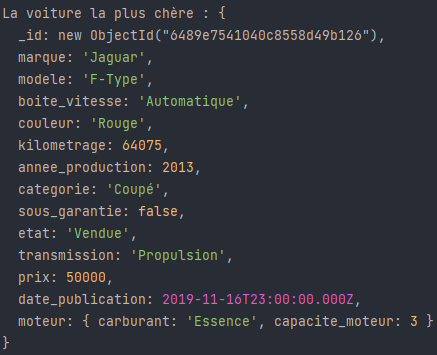


**Commentaire :** On utilise ici une agrégation pour trouver le kilométrage moyen du catalogue.

### Quelle est le véhicule le plus cher ?



**Résultat :**



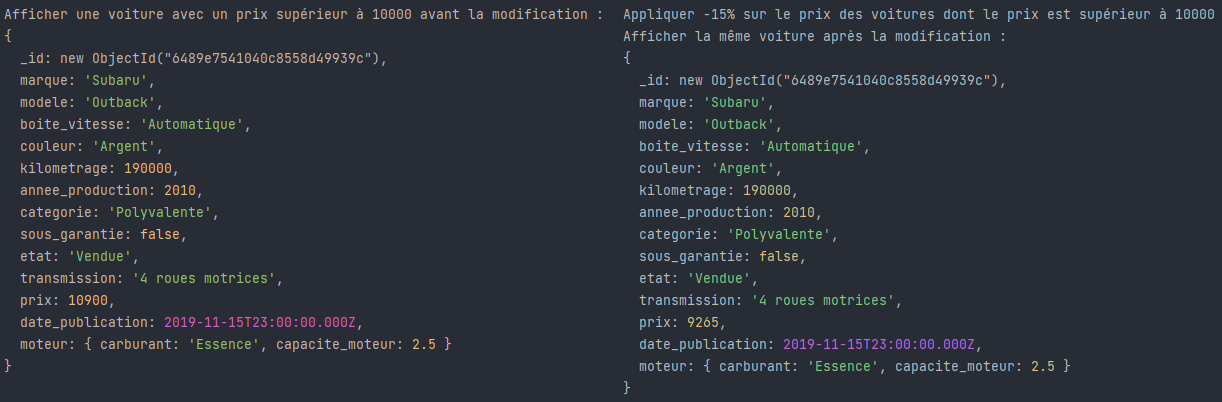
**Commentaire :** On récupère la liste des voitures, on trie par prix décroissant puis on sélectionne uniquement le premier élément.

# Requêtes recherchées

### Appliquer une réduction de -15% sur le prix des voitures vendues à plus de 10 000 €

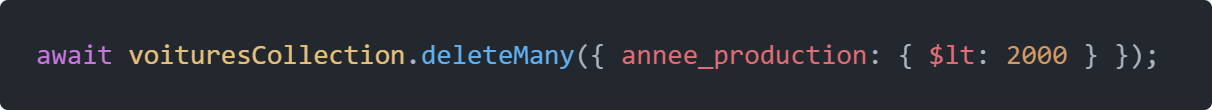


**Résultat :**

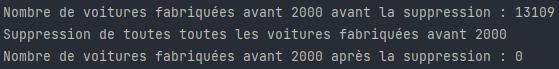


**Commentaire :** L’opération a lieu en trois temps. On commence par sélectionner les voitures dont le prix est supérieur à 10 000 €, puis on modifie le prix de chacune d’elles pour finalement mettre à jour les documents avec le nouveau prix.

### Supprimer toutes les voitures fabriquées avant 2000

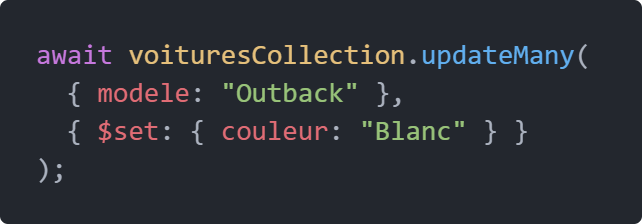


**Résultat :**

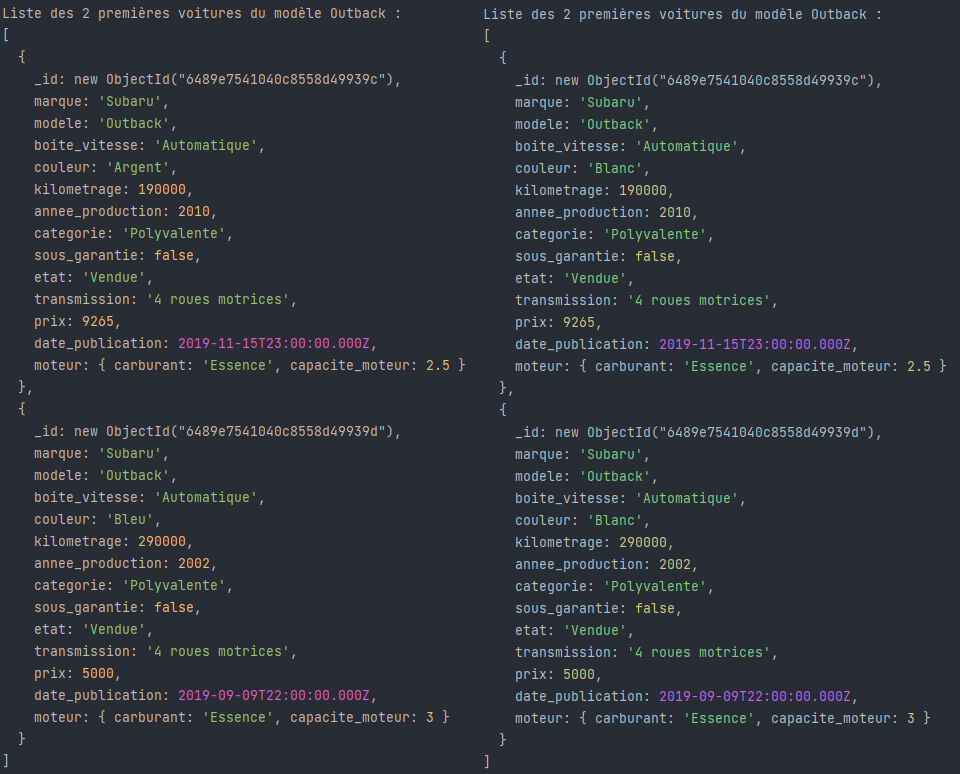


**Commentaire :** On utilise tout simplement deleteMany pour supprimer les documents. La quantité de véhicules concernés est comptée avant et après l’opération afin de vérifier la bonne exécution de la requête.

### Changer la couleur de tous les véhicules du modèle Outback

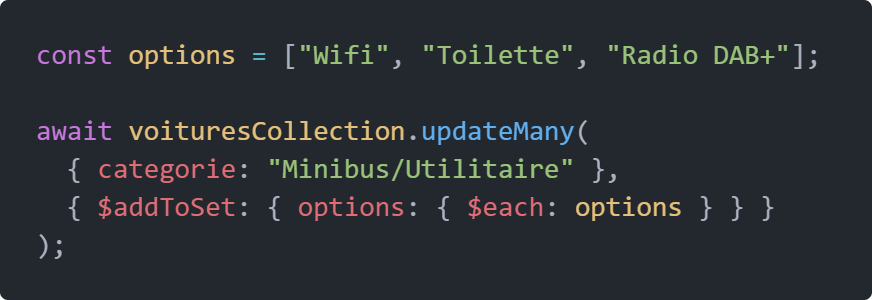


**Résultat :**



**Commentaire :** On peut ici mettre à jour tous les véhicules dont la valeur de modèle est « Outback ». Pour vérifier, on sélectionne les deux premiers véhicules du modèle avant et après l’opération.

### Ajouter des options pour les véhicules de la catégorie Minibus/Utilitaire



**Résultat :**

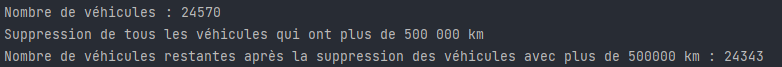


**Commentaire :** On cherche ici à manipuler un nouveau type de données qui n’a pas encore été introduit dans notre jeu de données : les tableaux. On insère ici un tableau de strings pour tous les documents sont la catégorie est « Minibus/Utilitaire ».

### Supprimer les véhicules qui ont plus de 500 000 km



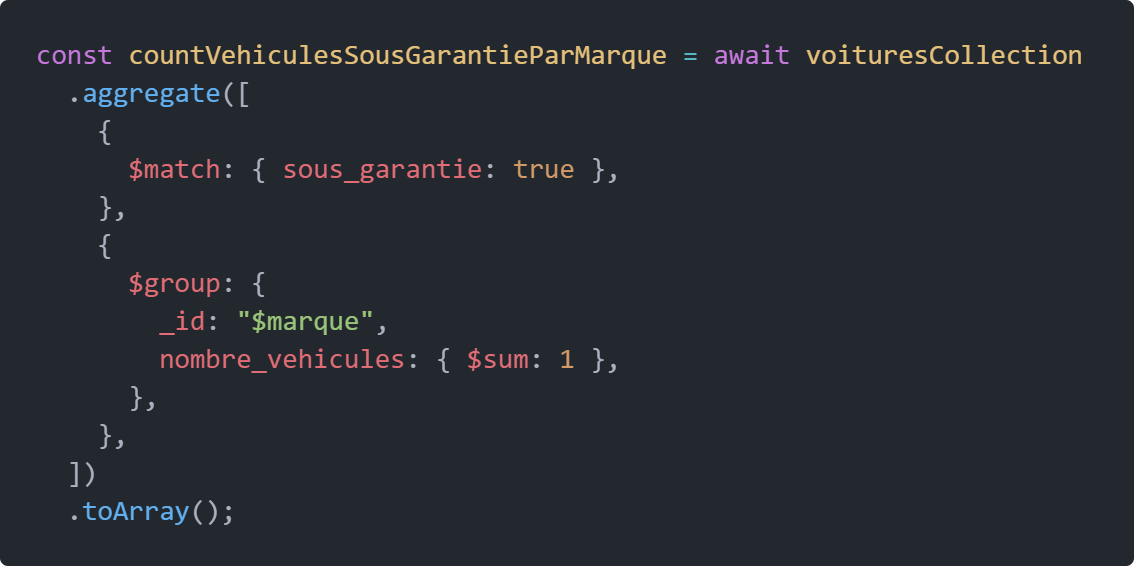
**Résultat :**



**Commentaire :** On supprime ici tous les véhicules qui remplissent la condition. Pour vérifier, on compte le nombre de véhicules concernés avant et après l’opération.

# Requêtes complexes

### Compter le nombre de véhicules sous garantie pour chaque marque

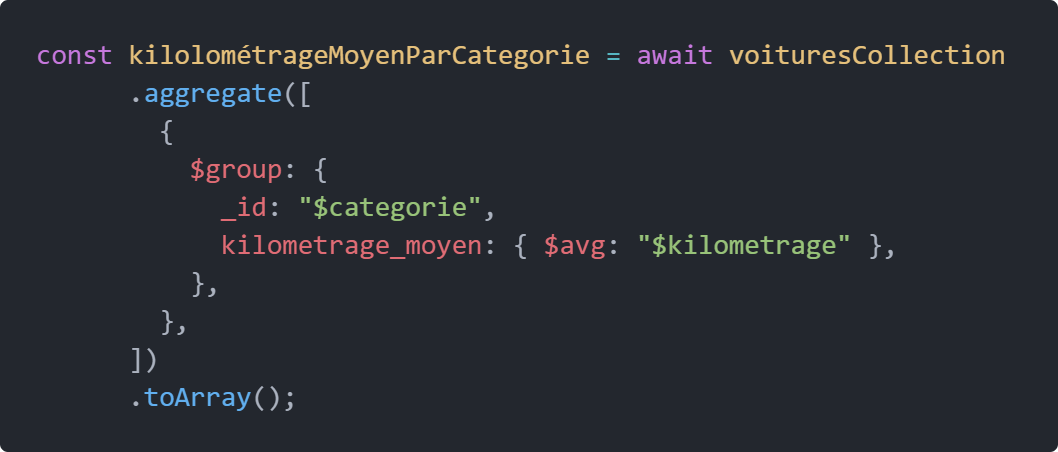


**Résultat :**

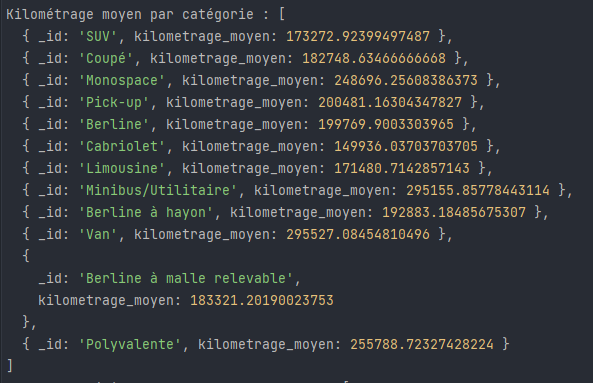


**Commentaire :** On utilise une agrégation pour regrouper par marque, sélectionner la garantie et sommer les valeurs correspondantes.

### Quel est le kilométrage moyen par catégorie de véhicules

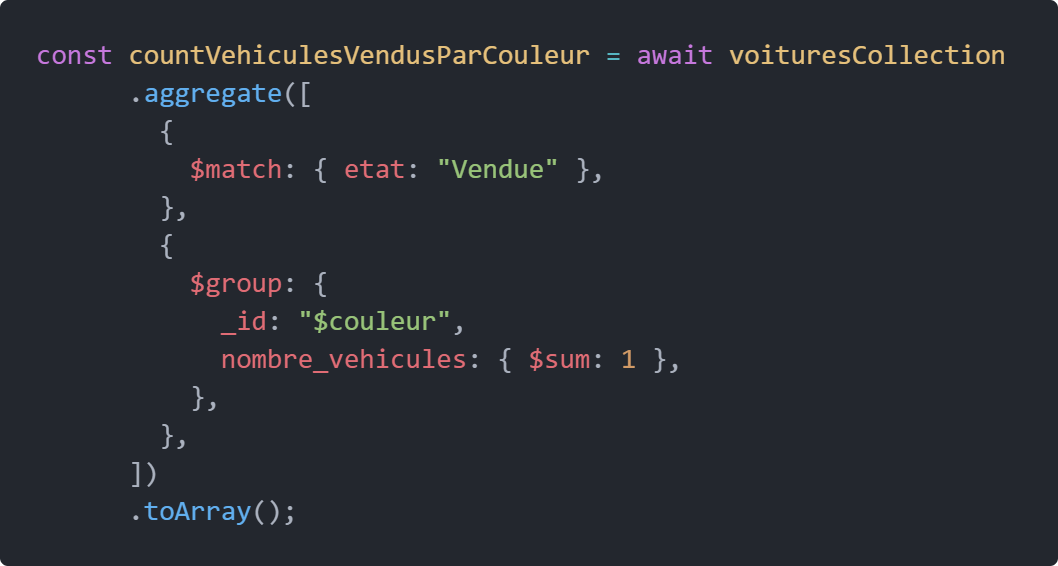


**Résultat :**

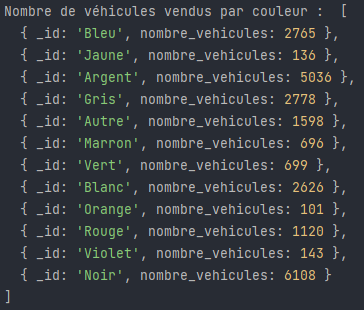


**Commentaire :** Ici on regroupe par catégorie puis on utilise la fonction d’agrégation de moyenne.

### Quel est le nombre de véhicules vendus pour chaque couleur ?

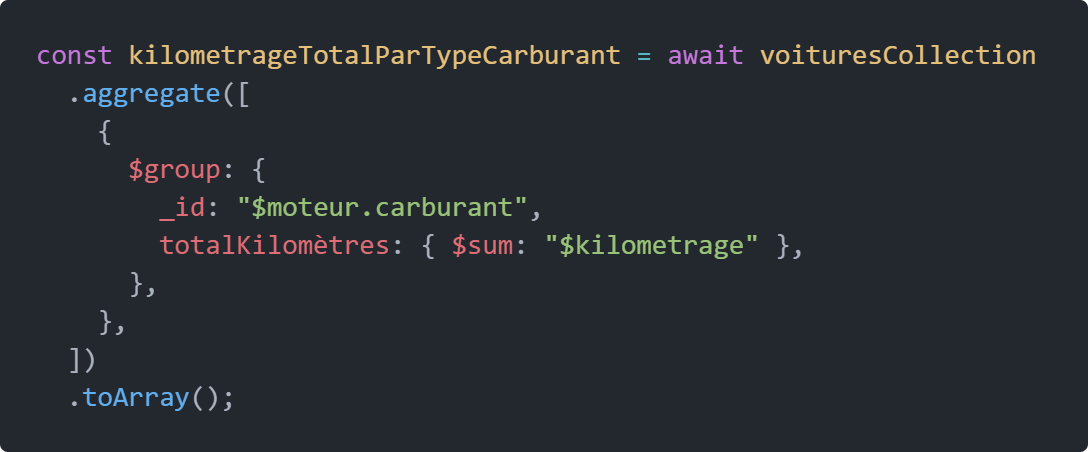


**Résultat :**

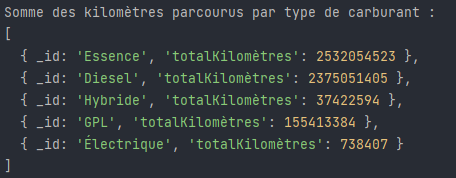


**Commentaire :** Ici on groupe par couleur, on filtre par état « Vendue » puis on somme les véhicules concernés.

### Quelle est la somme des kilomètres parcourus pour chaque carburant



**Résultat :**

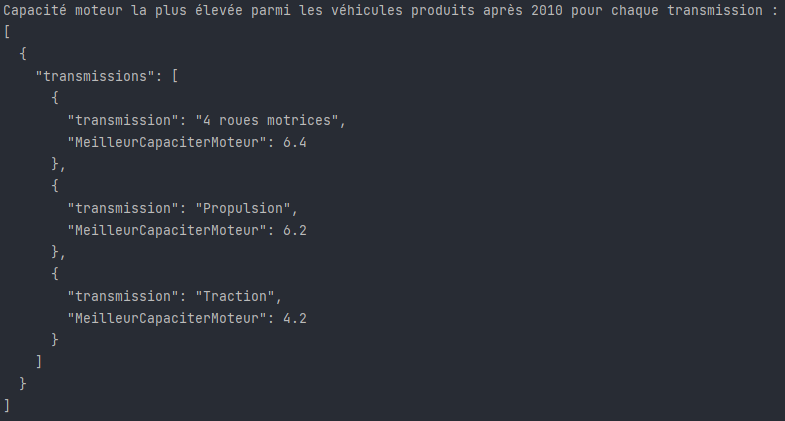


**Commentaire :** Ici on groupe avec l’attribut « carburant » de l’objet « moteur », puis on somme les kilomètres.

### Quelle est la capacité du moteur la plus élevée pour les trois types de transmissions pour les véhicules produits après 2010



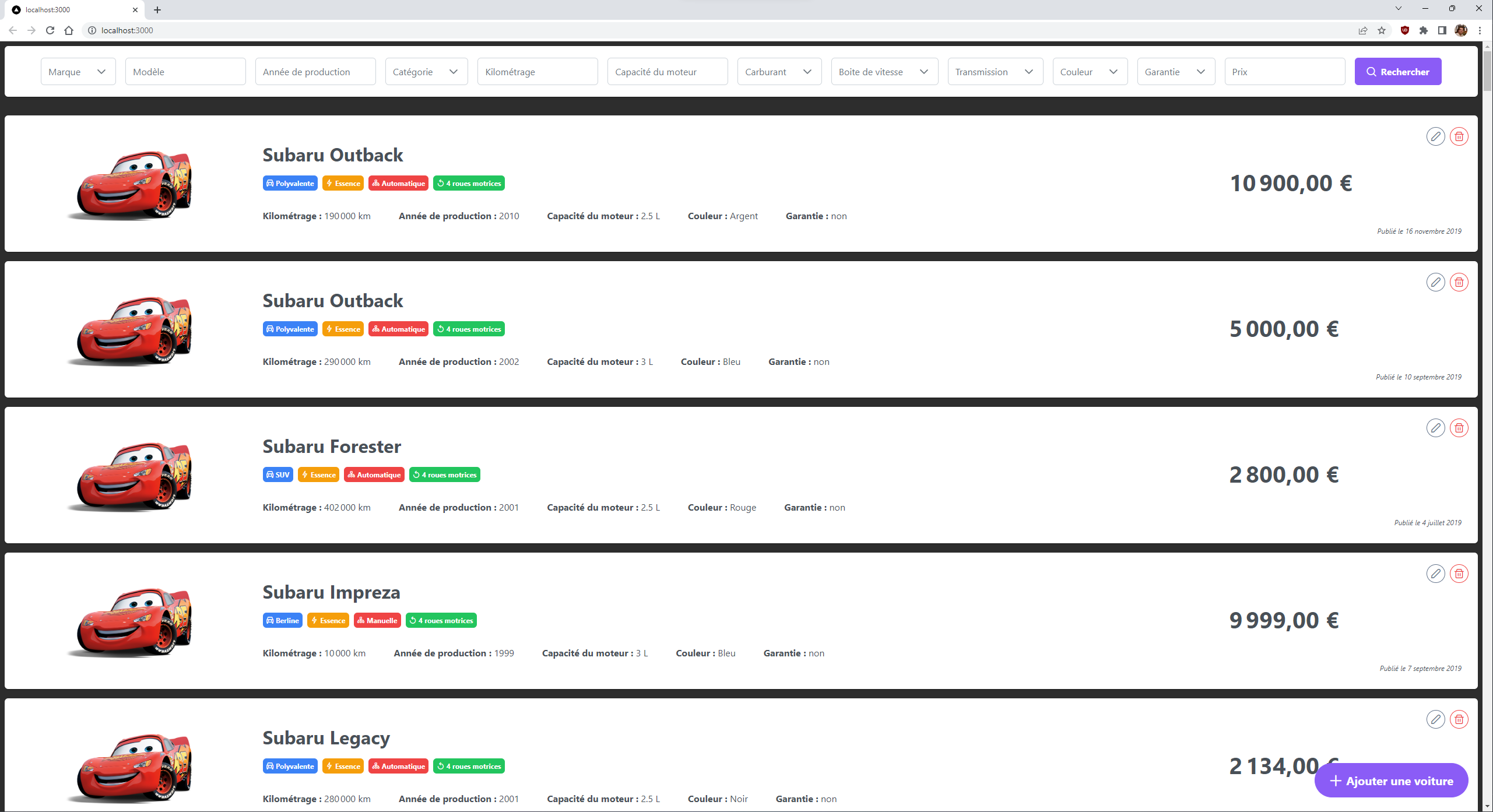
**Résultat :**



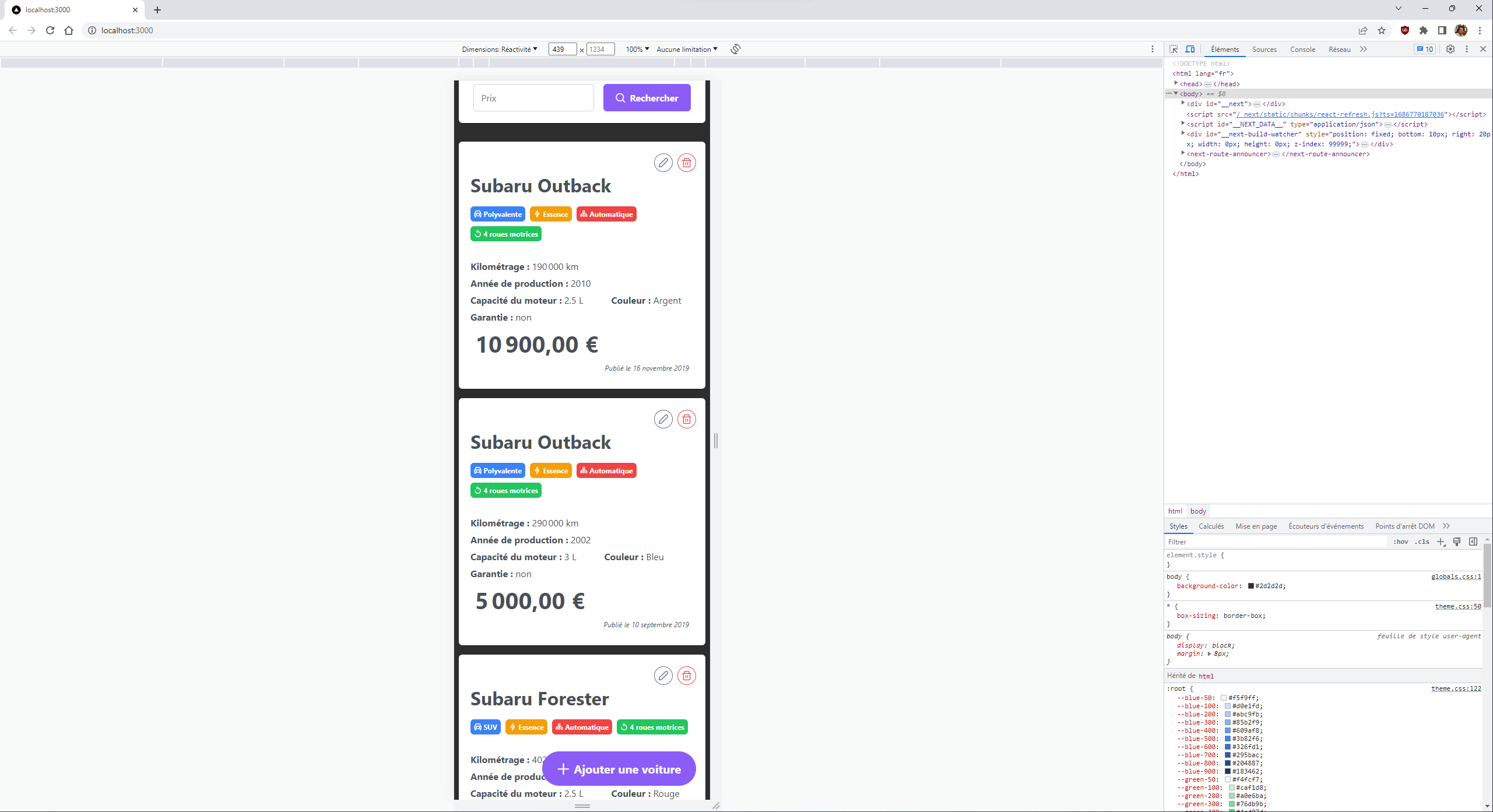
**Commentaire :** On groupe les véhicules par leur transmission et on extrait la capacité du moteur maximale dans chaque groupe. On filtre également selon l’année de production. On trie ensuite les résultats par ordre de transmission. On crée ensuite un groupe unique avec un identifiant nul qui utilise l'opérateur $push pour ajouter chaque transmission et sa meilleure capacité moteur au tableau "transmissions". Puis on projette uniquement sur la transmission.

# Démonstration de l’interface

Accueil :

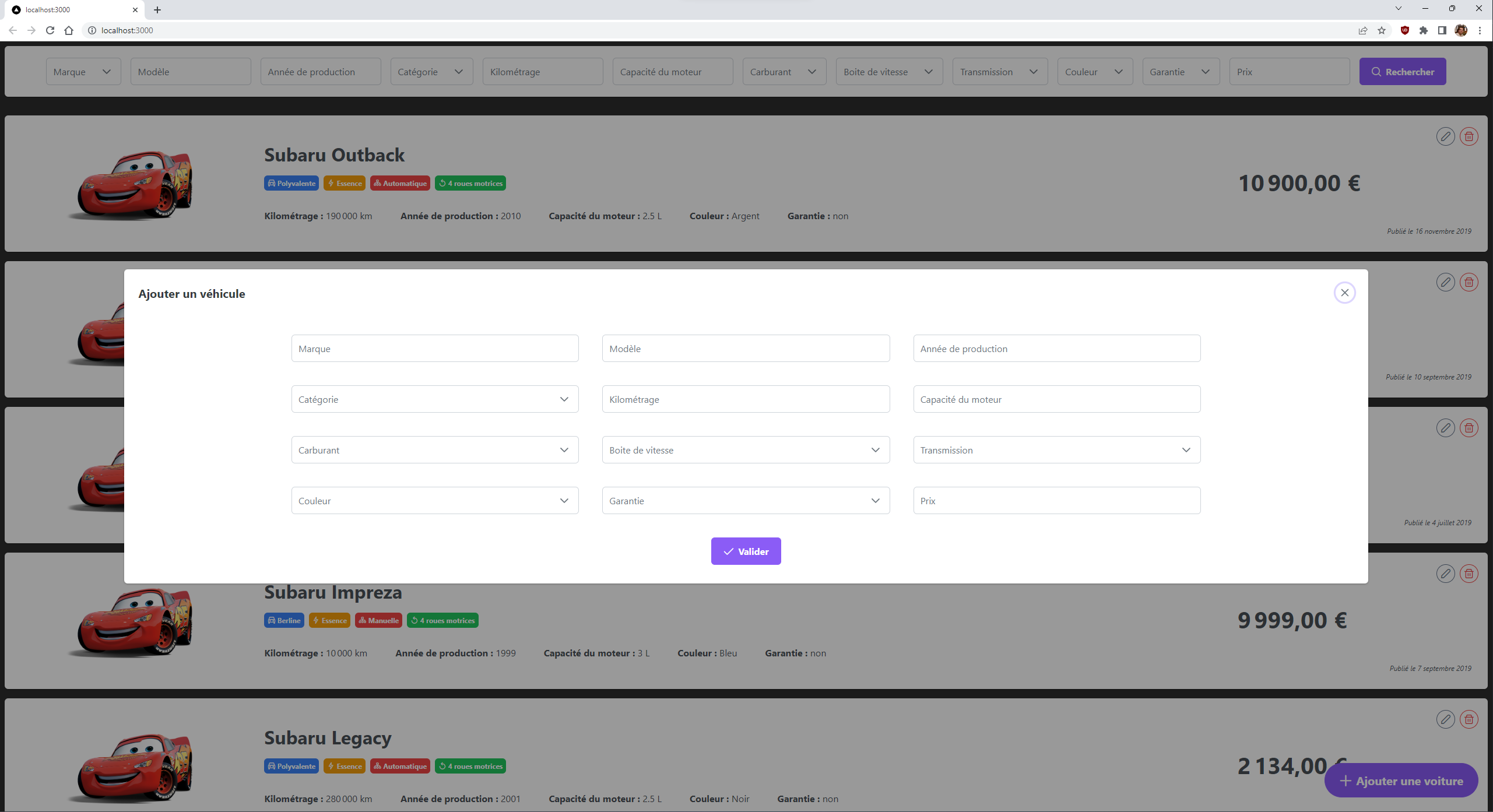


On peut voir ici la liste des véhicules. Tous les attributs apparaissent d’une manière ou d’une autre.

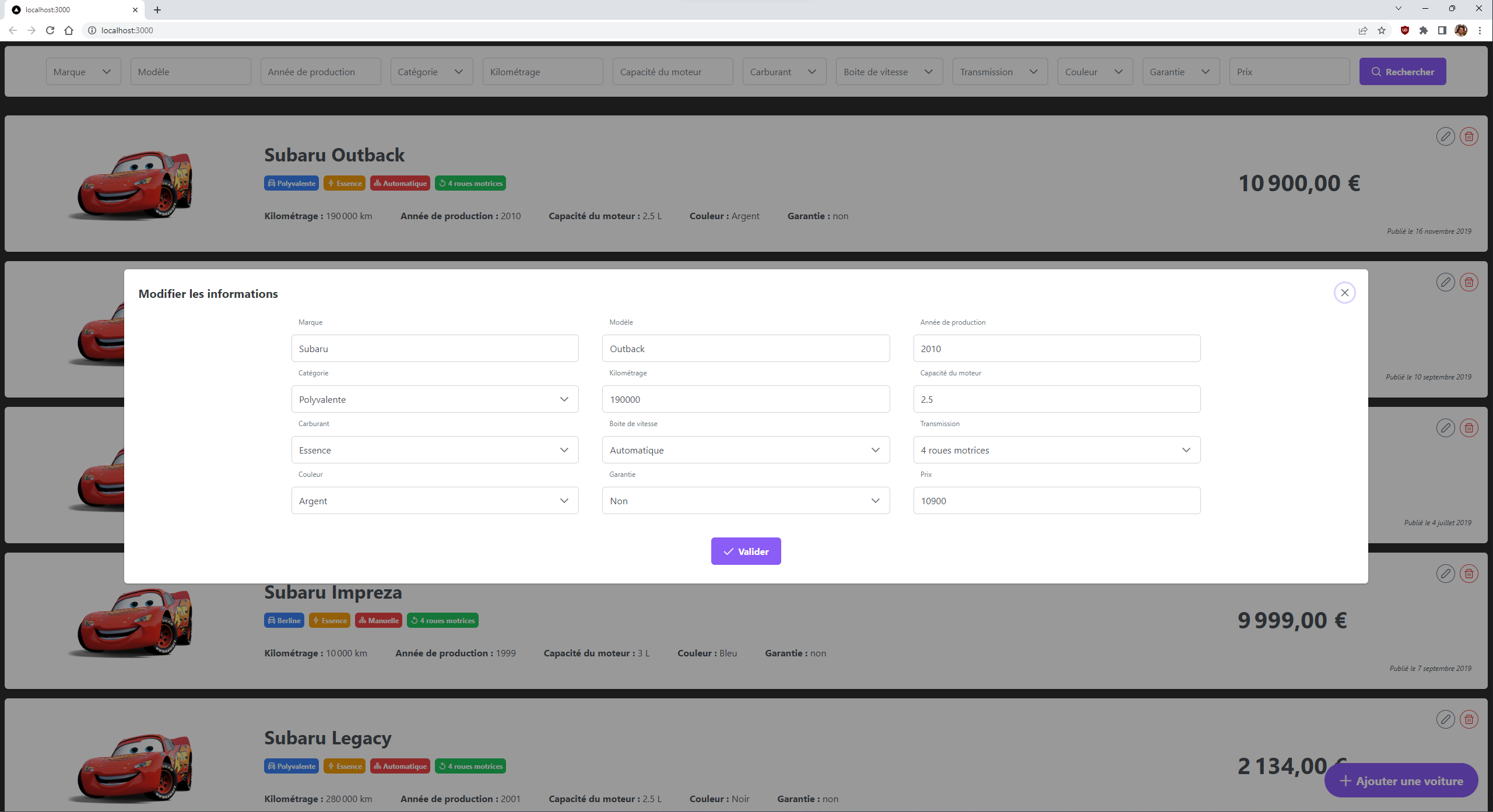


On peut également constater que l’interface est responsive.

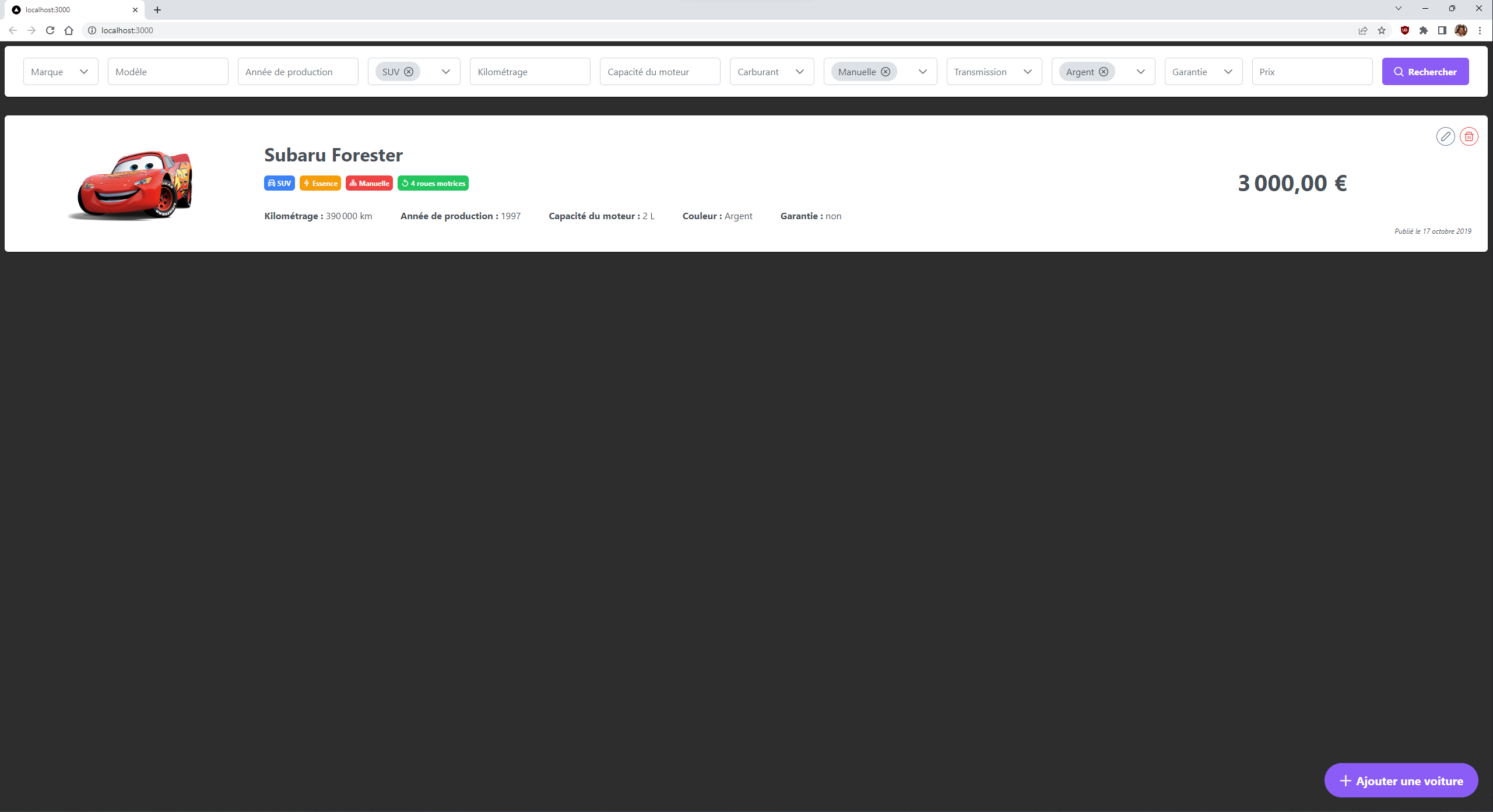
On peut également ajouter des véhicules :



On peut également modifier les propriétés d’un véhicule en cliquant sur l’icône en forme de crayon en haut à droite de chaque fiche. Les informations seront déjà remplies :



On peut également supprimer un véhicule en cliquant sur l’icône en forme de corbeille en haut à droite de chaque fiche.



On peut également filtrer les données à l’aide des champs en haut de la page.

# Test de montée en charge

**Informations préalables** :

* Tous les tests de montée en charges ont été réalisés en insérant l’ensemble de nos données une fois supplémentaire à chaque itération jusqu’à atteindre le million de véhicules.
* L’unité de temps utilisé durant tous les tests de montée en charge est la seconde.
* L’opération exécuté pour toutes les opérations de mises à jour est l’incrémentation du prix de chaque véhicule de 1.

Temps d'insertion : Le temps d'insertion semble augmenter progressivement à mesure que le nombre de voitures testées augmente. Cependant, la croissance n'est pas linéaire, ce qui indique peut-être une complexité d'insertion légèrement supérieure à l'échelle. Cette observation peut être due à plusieurs facteurs comme les performances du système à un moment donné.

Temps de mise à jour : Le temps de mise à jour augmente également avec le nombre de voitures testées. En effet, plus le nombre de documents à mettre à jour augmente, plus il peut être nécessaire de parcourir un plus grand volume de données, ce qui entraîne des temps de mise à jour plus longs.

Temps de suppression : Les temps de suppression augmentent eux-aussi. Il est intéressant de noter que les temps de suppression sont généralement inférieurs à ceux de l'insertion et de la mise à jour, ce qui peut être dû au fait que la suppression de données nécessite moins d'opérations internes.

Conclusion : En conclusion, vos observations montrent que l'opération qui nécessite le plus de temps est l'insertion, suivie de la mise à jour, puis de la suppression. Ces résultats sont cohérents avec les caractéristiques et les performances attendues de MongoDB. Les temps d'insertion peuvent être plus élevés en raison de la complexité de l'insertion à grande échelle, tandis que les temps de mise à jour et de suppression peuvent augmenter en fonction du nombre de documents à traiter. La conception orientée document de MongoDB lui permet d'optimiser les temps de suppression, ce qui explique pourquoi ils sont généralement inférieurs aux temps d'insertion et de mise à jour. Il convient également de noter que d'autres facteurs tels que la configuration du système et les ressources disponibles peuvent également avoir une incidence sur les performances observées.

# Comparaison avec Mongoose

Temps d'insertion : Les temps d'insertion sont toujours plus élevés lors de l'utilisation de Mongoose par rapport à MongoDB seul. Plus le nombre de véhicules insérer est grand, plus l’écart devient important.

Temps de mise à jour : Les temps de mise à jour semblent être équivalents entre l'utilisation de MongoDB et de Mongoose. Dans les deux cas, les opérations de mise à jour s'appuient sur les fonctionnalités intégrées de MongoDB pour modifier les documents existants. Mongoose n'ajoute pas de surcoût significatif dans ce cas, car il se base sur les fonctionnalités de mise à jour de MongoDB.

Temps de suppression : De manière similaire au test de modification, les temps de suppression sont du même ordre de grandeur, que ce soit en utilisant une bibliothèque comme Mongoose ou en interagissant directement avec une base de données MongoDB. Les opérations de suppression sont généralement rapides, car elles ne nécessitent pas de vérifications supplémentaires ou de manipulations complexes des données.

Conclusion : Dans l'ensemble, lorsqu'il s'agit de performances brutes, l'utilisation de MongoDB seul peut être plus rapide que l'utilisation de Mongoose. Cependant, Mongoose offre des fonctionnalités supplémentaires et une couche d'abstraction qui peut faciliter le développement et la gestion des données, surtout lorsque des contraintes de schéma strictes et des relations complexes sont impliquées.

# Comparaison avec MySQL

Temps d'insertion : Les temps d'insertion pour MongoDB augmentent progressivement avec l'augmentation du nombre de voitures testées. Cela peut suggérer que MongoDB nécessite plus de temps pour insérer de nouvelles données à mesure que la taille de la base de données augmente. En revanche, les temps d'insertion pour MySQL sont généralement plus bas et moins affectés par l'augmentation du nombre de voitures testées. Cela indique que MySQL peut gérer efficacement les opérations d'insertion, même avec une base de données plus importante.

Temps de mise à jour : Les temps de mise à jour pour MongoDB augmentent également progressivement avec l'augmentation du nombre de voitures testées. Cela peut signifier que les opérations de mise à jour dans MongoDB prennent plus de temps à mesure que la quantité de données à mettre à jour augmente. D'autre part, le même phénomène est observé en utilisant une base MySQL. Les temps de mises à jour entre ces deux bases données sont équivalents.

Temps de suppression : Les temps de suppression pour les deux bases de données augmentent globalement avec le nombre de voitures testées. Cependant, les temps de suppression pour MongoDB sont généralement moins élevés que ceux de MySQL. MongoDB est conçu pour une suppression efficace des documents, grâce à son modèle de stockage orienté document et à son système d'indexation. Il peut localiser rapidement les documents à supprimer et les retirer de manière optimisée. En revanche, MySQL peut nécessiter plus de temps pour gérer les opérations de suppression en raison de son modèle orienté. Néanmoins, les performances de suppression dans les deux bases de données restent acceptables, avec des temps généralement plus rapides dans le cas de MongoDB.

Conclusion : Les résultats de ces tests de montée en charge ont révélé des différences de performance entre MongoDB et MySQL. MySQL a montré des performances supérieures en termes de temps d'insertion, tandis que les temps de mise à jour étaient similaires entre les deux systèmes de gestion de base de données. En ce qui concerne la suppression, MongoDB a affiché des temps plus courts que MySQL, grâce à sa conception orientée document. Ces résultats contredisent nos attentes initiales, où nous pensions que MongoDB serait systématiquement plus rapide pour insérer des données massives dans une seule et unique table. Ces différences soulignent l'importance de prendre en compte les caractéristiques et les contraintes spécifiques de chaque base de données lors du choix de la solution adaptée à un cas d'utilisation donné.