ÉCOLE CENTRALE LYON

Modélisation d'un atelier garage

SQL et PHP objet

Thibault CHATAIGNER
Thomas PERROT

Année 2015 - 2016

Table des matières

Introduction

Le SQL est un langage informatique qui permet de manipuler des bases de données structurées. Créé en 1970 par Donald Chamberlin et Raymond Foyce, il est aujourd'hui l'un des langages de bases de données les plus utilisés. Ainsi il semble nécessaire aux ingénieurs de demain d'acquérir les compétences nécessaire pour maîtriser ce langage, aussi pratique que répandu.

C'est dans ce but que nous avons réalisé le projet décrit dans ce rapport. Dans le cadre de ce projet, notre but est de créer une application pour organiser l'atelier d'un garage. Pour cela, nous avons dans un premier temps conçu une base de données permettant de stocker les données nécessaires, puis réalisé une couche applicative permettant d’interagir avec ces données depuis une interface graphique.

Ce document est constitué du cahier des charges qui correspond aux attentes d’un potentiel client ainsi qu’un manuel d’utilisation dont le but est de faciliter le maniement du site internet. La dernière partie présente l’ensemble des démarches à suivre pour tester le site internet lors de la procédure de recette.

I - Cahier des charges

1 – L'énoncé du problème

L’énoncé du problème est le suivant :

« Un garage peut comporter plusieurs ateliers. On ne s’occupe que de l’atelier de réparation dont on souhaite faire la gestion.

- Une voiture à réparer est identifiée par son numéro d’immatriculation et caractérisée par sa marque, son type, son année, son kilométrage et sa date d’arrivée au garage.
- Chaque client est identifié par un numéro et caractérisé par son nom, son prénom, son adresse (commune) et par le nom de la personne qui prend en charge le suivi administratif de la commande. Chaque commune est identifiée par son nom et caractérisée par le nombre de clients qui ont réparé leurs voitures dans le garage.
- Le garage offre aux clients deux types de réparations : soit forfaitaire, incluant les pièces détachées (vidange moteur, changement de filtre à huile, remplacement des plaquettes de frein, changement d’amortisseurs ou pneus, etc.), soit non forfaitaire dépendant de la main d’oeuvre et des pièces détachées.
- Pendant l’intervention, le technicien peut noter ses remarques.
- Le technicien est alors identifié par son numéro et caractérisé par son nom et prénom, ainsi que le nombre de voitures qu’il a réparées. »

2 – Les facteurs d'exploitation

- **Confidentialité :** le site doit protéger les informations sensibles des utilisateurs.
- **Maniabilité :** l’interface du site doit être intuitive de manière à ce que l’utilisateur puisse prendre le site en main rapidement.
- **Adaptabilité :** Le code source sera organisé de telle sorte que les fonctionnalités à ajouter puissent être implémentées de manière aisée et systématique.

II - Modélisation de la base de données

1 – Les entités à créer

D'après l'énoncé du problème évoqué ci-dessus, on en déduit immédiatement les entités à créer afin d'avoir une base de donnée normalisée. On aura :

- une table **voiture**, avec pour clé primaire son immatriculation, et pour autres attributs sa marque, son type, son année, son kilométrage, sa date d'arrivée au garage, et son propriétaire,
- une table **client**, avec pour clé primaire son numéro (de client), et pour autres attributs son nom, son prénom, son adresse et son référent,
- une table **technicien**, avec pour clé primaire son numéro (de technicien), et pour autres attributs son nom et son prénom,
- une table **commentaire**. Une commentaire correspond à la remarque d'un technicien sur une voiture à un instant donnée. On choisi donc comme clés primaires l'immatriculation de la

voiture concernée, le numéro du technicien, et le date du commentaire (enregistrée à la seconde près pour différencier deux commentaires). Cette table a un seul autre attribut : le texte du commentaire, qui correspond à la remarque du technicien.

- une table **intervention**, qui correspond à la liste des opérations possibles sur une voiture (changement des freins, gonflage des plneus, etc.), avec pour clé primaire une id en auto-incrément, et pour attribut le nom de la réparation et son prix.
- une table **facture**, avec pour clé primaire l'id de la facture, et pour attributs son prix. On peut créer une facture comme étant une somme d'intervention. Dans ce cas, le prix de la facture évoluera avec le nombre d'interventions concernées. On peut aussi créer une facture nommée « forfait » suivit du numéro du forfait en question, avec un prix fixe. Cela permet de gérer le système de forfaits.
- une table **utilisateurs**, qui gère les utilisateurs autorisés à utiliser l'application. Elle contient un admin, qui peut gérer la liste des utilisateurs, des techniciens, des référents, et des utilisateurs simples. Nous reviendrons sur leurs droits plus loin dans ce rapport. Elle a pour clé primaire l'id de l'utilisateur, et pour autres attributs le pseudo, le mot de passe et les privilèges associés au compte (admin ou non).

On peut formuler deux remarques. Premièrement, on peut récupérer le nom des villes avec leur nombre de clients y résidant en faisant une simple requête sur la table client. Il est donc inutile de créer une table contenant les villes.
Deuxièmement, la table facture est absolument nécessaire pour que l'application fonctionne. En effet, il aurait été possible de calculer le prix total de chaque réparation avec des jointures en fonction des interventions associées. Cependant, ceci n'est pas pertinent. En effet, le prix total de la facture aurait changé a chaque modification des prix dans la table intervention, ce qui n'est pas cohérent avec la réalité (le prix total d'une facture doit toujours reste le même, et ne doit pas changer avec le temps).

2 – Les relations entre les tables

- **Client / voiture :** une voiture a un seul propriétaire, un client peut avoir plusieur voiture. On a donc une relation one-to-many. L'attribut numero de client est donc une foreign-key de voiture sur l'attribut propriétaire,
- **Commentaire / voiture / technicien :** un technicien peut faire des commentaires sur plusieurs voitures, et une voiture peut avoir des commentaires de la part de plusieurs techniciens. On a donc une relation many-to-many entre les voitures et les techniciens concernant les commentaires. Ainsi, le champ voiture du commentaire a pour foreign-key l'immatriculation de la voiture associée, et le champ technicien a pour foreign-key le numéro du technicien associé.
- **Voiture / technicien :** un technicien répare plusieurs voitures, une voiture est réparée par plusieurs techniciens. On a donc une relation many-to-many, qui amène à créer une table de mixage « repare ». Celle-ci a pour foreign key le numéro du technicien, l'immatriculation de la voiture, et la date de début associée (de manière à ce qu'une voiture puisse être réparée plusieurs fois par le même technicien). Elle prend aussi pour attribut la date de fin (null si la voiture est encore en réparation) et l'id de la facture, qui a pour foreign-key l'id de la facture associée dans la table facture.
- **Repare / facture :** une réparation e'st associée à une facture, d'où la foreign-key dans la table facture sur l'attribut idFacture de la table repare.
- **Facture / intervention :** une facture est composée de plusieurs interventions, une intervention est mentionnée dans plusieurs factures. On a donc une relation many-to-many, qui nous amène à créer une table de mixage facture_intervention, contenant deux clés

primaires : l'id de la facture et l'id de l'intervention, ayant pour foreign-key respectives idFacture de la table facture et idIntervention de la table intervention.

- **Client / utilisateurs :** le référent d'un client est forcément l'un des utilisateurs de l'application. Le champ id de la table utilisateurs est donc la foreign-key du champ référent dans la table client.
- **Technicien / utilisateurs :** un technicien est un utilisateur, son id a donc pour foreign key l'id d'un des utilisateurs de la table utilisateurs.

Le diagramme entité association du problème est le suivant. Pour des raisons de lisibilité, les attributs des tables ne sont pour la plupart pas montré sur ce diagramme, mais dans le diagramme suivant représentant les tables.

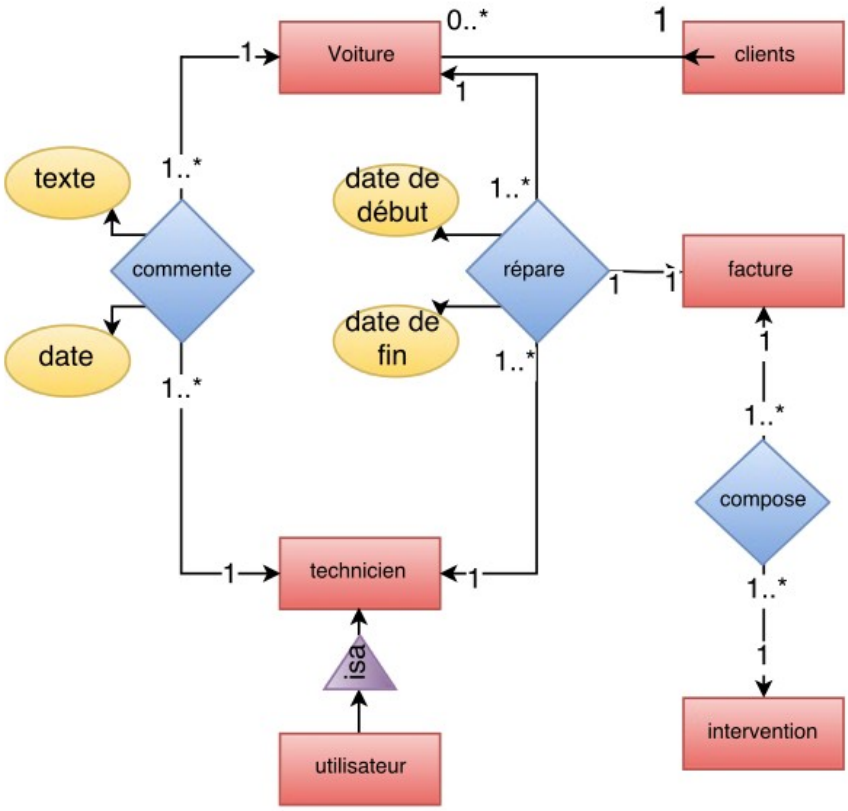


Diagramme entités-associations

Au vu des relation évoquées entre les tables, on a donc la structure de base résumée dans le schéma ci-dessous.

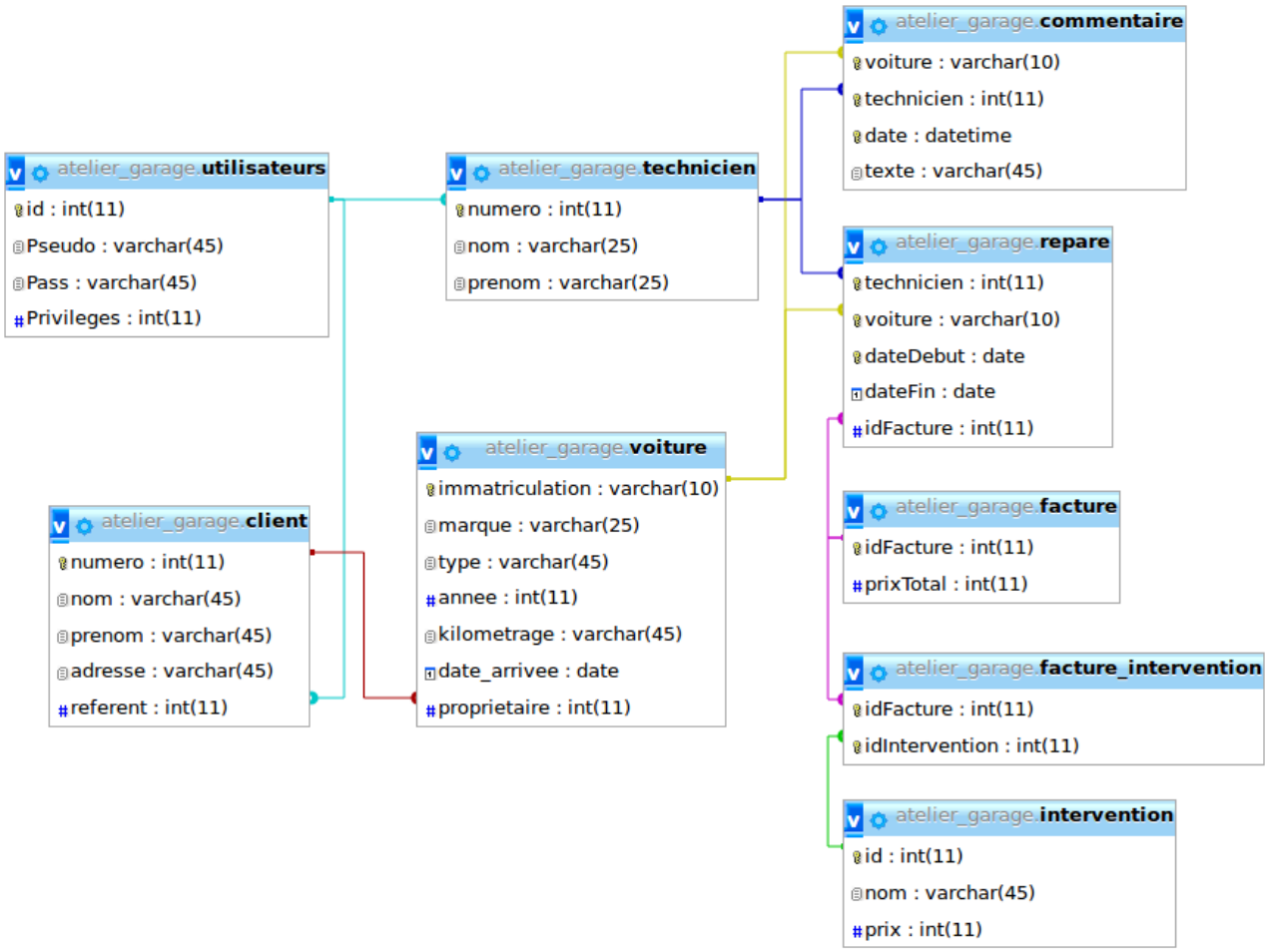


Schéma des tables et des relations

3 – Les vues et les procédures

Nous avons choisi de ne pas stocker de requêtes dans la base de données, et de les implémenter dans l'application. En déplaçant la complexité vers la couche applicative, notre objectif est d'alléger la base de données pour faciliter sa maintenabilité et son évolutivité.

4 – Les triggers

Comme évoqué dans les remarques sur les entités à créer au début de cette partie, des triggers sont nécessaires pour maintenir la table facture a jour. En effet, pour une facture donnée, on veut garder un prix total égal à la somme des prix des interventions liées à cette facture. Ainsi, si on ajoute une intervention à la facture (ajout d'un champ dans la table facture_intervention), le prix total est augmenté du prix de la facture. Si on supprime une intervention de la facture (suppression d'un champ dans la table facture_intervention), le prix total est également modifié, idem si on modifie les interventions liées à une facture. Nous avons donc créé trois triggers dans la classe facture_intervention pour répondre à ce besoin : après l'ajout, après la modification.

On peut remarquer qu'il ne faut pas créer de triggers modifiant le prix total des factures si on modifie les prix des interventions associée. En effet, les factures ayant déjà été réalisées et payées,

leur prix total ne change plus !

5 – La gestion des utilisateurs

pour garantir la pérennité de l'application, toute personne souhaitant l'utiliser doit d'identifier par un login et s'authentifier par un mot de passe. En fonction du profile de la personne (admin, référent, technicien, ou autre), celle-ci aura différents droits. Le tableau ci-dessous détaille la liste des champs pouvant être modifiés avec les privilèges accordé pour les différents utilisateurs. La notation suivante est adoptée : w signifie que l'utilisateur peut ajouter, supprimer ou modifier les champs concernés (write), r signifie que l'utilisateur peut voir les champs concernés.

Champs	Admin (3)	Référent (2)	Technicien (1)	Autre (0)
facture	rw	rw	r	r
client	rw	rw	r	r
voiture	rw	rw	r	r
technicien	rw	rw	r	r
repare	rw	rw	r	r
intervention	rw	rw	r	r
commentaire	rw	r	rw	r
utilisateurs	rw	-	-	-

Tableau récapitulatif des droits des utilisateurs

L'admin est autorisé à tout voir et tout modifier. Les référents (qui organisent et gèrent le garage) peuvent voir et modifier toutes les tables, à l'exceptions des commentaires, qui seules les techniciens peuvent modifier. Les techniciens peuvent tout voir, mais ne peuvent modifier que la liste des commentaires. Seul l'admin peut voir ou modifier la liste des utilisateurs.

Ces fonctions n'ont pas été implémenté dans la base de données afin de faciliter sa maintenabilité et son évolutivité, mais dans la couche applicative. Cela permet par ailleurs de n'afficher les options de modifications des tables qu'aux personnes concernées, ce qui facilite grandement l'expérience utilisateur, en affichant des interfaces personnalisés aux besoins de chacun.

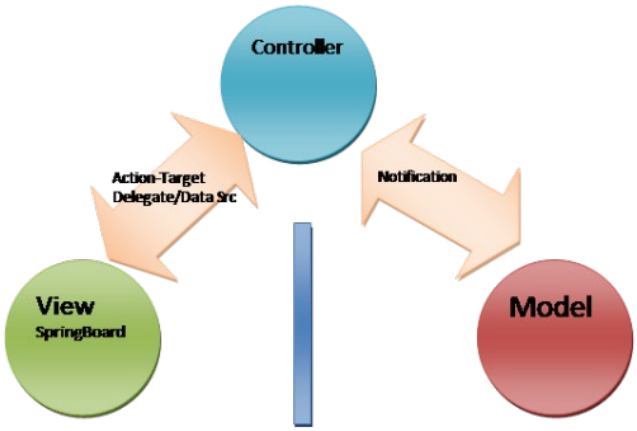
II - Architecture logicielle

1 – Le choix du langage et d'un design pattern

Le premier choix à faire fut celui du langage. Le projet étant la réalisation d'une application métier, il est judicieux de penser à héberger celle-ci dans un cloud et de la louer comme un SaaS aux personnes intéressées. On gagne ainsi en souplesse, en évolutivité, et cela permet d'atteindre le plus vite possible un maximum de clients potentiels. Aussi il est intéressant d'utiliser un langage serveur. Ayant tous les deux eu plusieurs expériences réussies en PHP, nous nous sommes tout naturellement tourné vers ce langage.

Afin de répondre au mieux à la taille et à la complexité du projet, il était nécessaire d'adopter une architecture dans notre code qui nous permettrait de travailler de manière synchronisée avec un maximum de cloisonnement entre les fonctionnalités. Nous nous sommes donc tourné vers de la programmation objet.

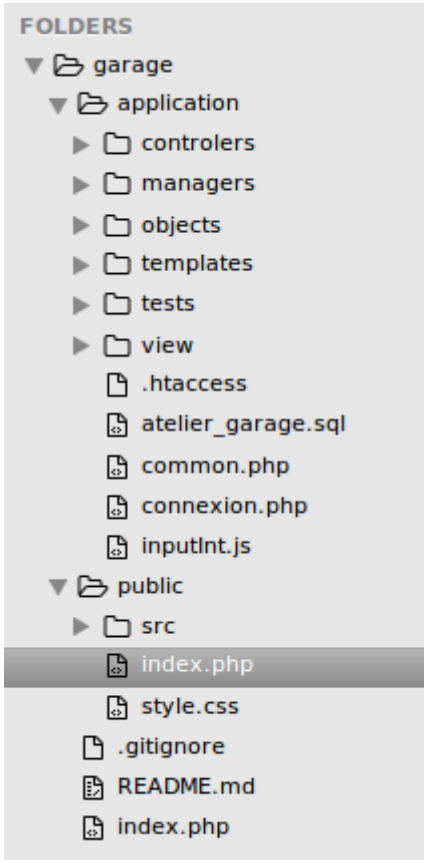
Enfin, pour assurer la cohérence du projet tout au long de sa réalisation, nous avons décidé d'utiliser une architecture MVC (modèle, vue, contrôleur). Cela nous permet de suivre un schéma sont l'efficacité est reconnu.



L'architecture MVC

2 – L'organisation du code source

Le code source se trouve sur un dépôt Githu à l'adresse https://github.com/e-Daho/atelier_garage. La racine du répertoire ne contient que deux dossier, un dossier public et un dossier privé. Pour des raisons de sécurité, seuls les fichiers contenus dans le répertoire « public » sont accessibles à l'utilisateur. Ce répertoire ne contient qu'un fichier appelant les codes contenus dans le répertoire « application » et des images. Le dossier application contient des fichiers fonctions, les fichiers template et le fichier common.php. L'accès au dossier « application » n'est pas possible pour l'utilisateur, même en manipulant l'url, grâce au fichier .htaccess qui bloque les requêtes d'accès. Ainsi on restreint l'utilisateur à utiliser la partie de l'application qui lui est destiné.



L'organisation du code source

Capture de l'interface utilisateur lors d'une requête dans le dossier « application »

3 – Les variables objets (dossier objects)

Toutes les variables objets créées sont placée dans le dossier objects. Ces objets transitent de la vue au modèle en passant par le contrôler, et sont utilisés par chaque autre objets du programme. On retrouve par exemple les objets technicien, voiture, client, etc. Chaque objets possède des attributs propres à la table associée, ainsi que des accesseurs et mutateurs (méthodes get et set).

4 – Le modèle (dossier managers)

Le modèle est constitué de managers (un par table), présents dans le dossiers applications/managers.

Ceux-ci permettent de faire l'interface entre le programme et la base de données en « convertissant » les objets utilisés dans le programme en champ dans les tables, et vice-versa. Ils possèdent tous des méthodes pour ajouter, lire, modifier ou supprimer les objets en base de données, plus quelques fonctions annexes (tester l'existence d'un objet en base de donnée, obtenir une liste d'objets en fonction d'entrées spécifiées, etc.).

5 – La vue (dossier view)

La vue est composée d'un ensemble d'objets vues, qu'on « hydrate » avec les objets associés. Chaque objet vue correspond à une interface différente dans l'application. Ce découpage en différents objets, chacun associés à une fonction précise permet encore une fois de bien séparer les fonctionnalités au sein du code.

6 – Le contrôleur (dossier controlers)

Le contrôleur est composé d'un ensemble d'objets contrôleurs qui reçoivent les requêtes de l'utilisateur depuis la vue, et les envoie ensuite au modèle après en avoir contrôlé la validité. Ils sont les liens indispensables qui permettent à la vue et au modèle de communiquer, et permettent de renforcer la sécurité en contrôlant toutes les requêtes qui arrivent sur le modèle.

7 – La base de données

La base de données a été exportée au format .sql, et se trouve dans le dossier application. Il suffit de l'importer pour retrouver la base de données décrite ci-dessus.

III – Application

1 – L'accueil

Au début de la visite, un écran d'accueil nous invite à nous connecter. On se connecte en utilisant un couple pseudo/password stocké dans la base de données. Pour des raisons de sécurité, les mots de passe ne sont pas stockés en clair dans la base de donnée, mais sous forme de hash, c'est à dire cryptés. Une fonction de hashage va donc transformer le mot de passe entré par l'utilisateur en un hash, et comparer avec les hash stockés dans la base de donnée.

Capture d'écran de la page de connexion

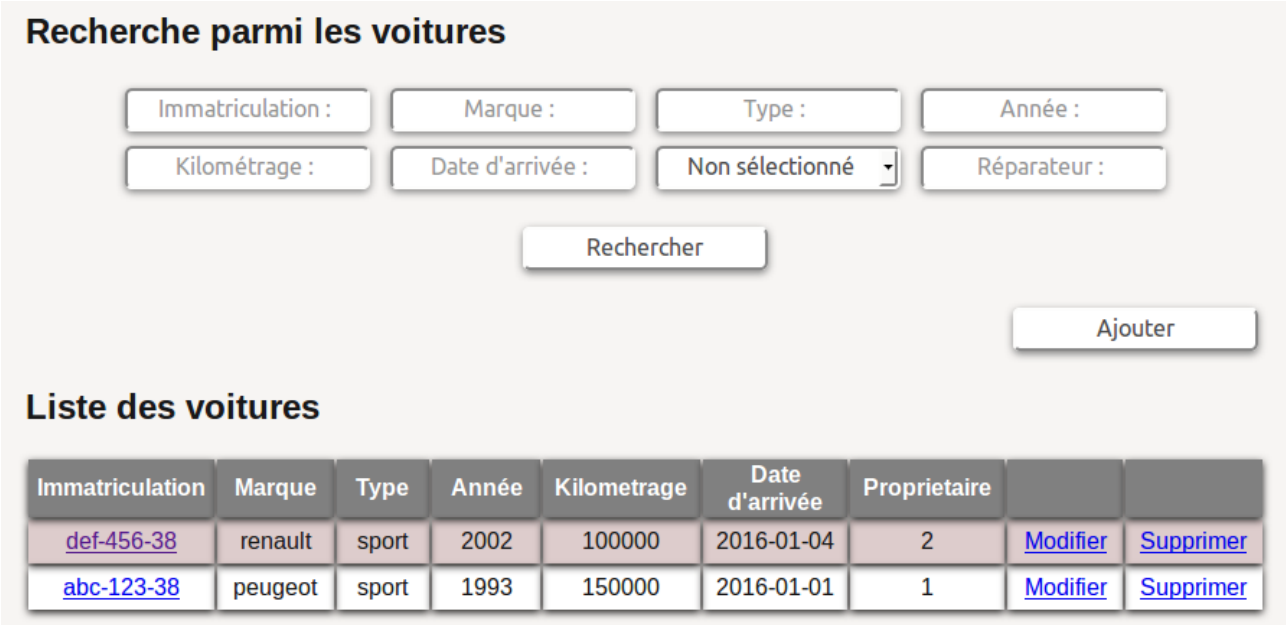
On arrive ensuite sur la page d'accueil, présentant les différents menus. Ceux-ci sont accessibles depuis le menu, on directement en cliquant sur les vignettes associées. En fonction des droits associés à l'utilisateur, celui-ci peut voir 6 ou 7 menus.

Capture d'écran de l'écran d'accueil en mode admin

Capture d'écran de l'écran d'accueil en mode référent

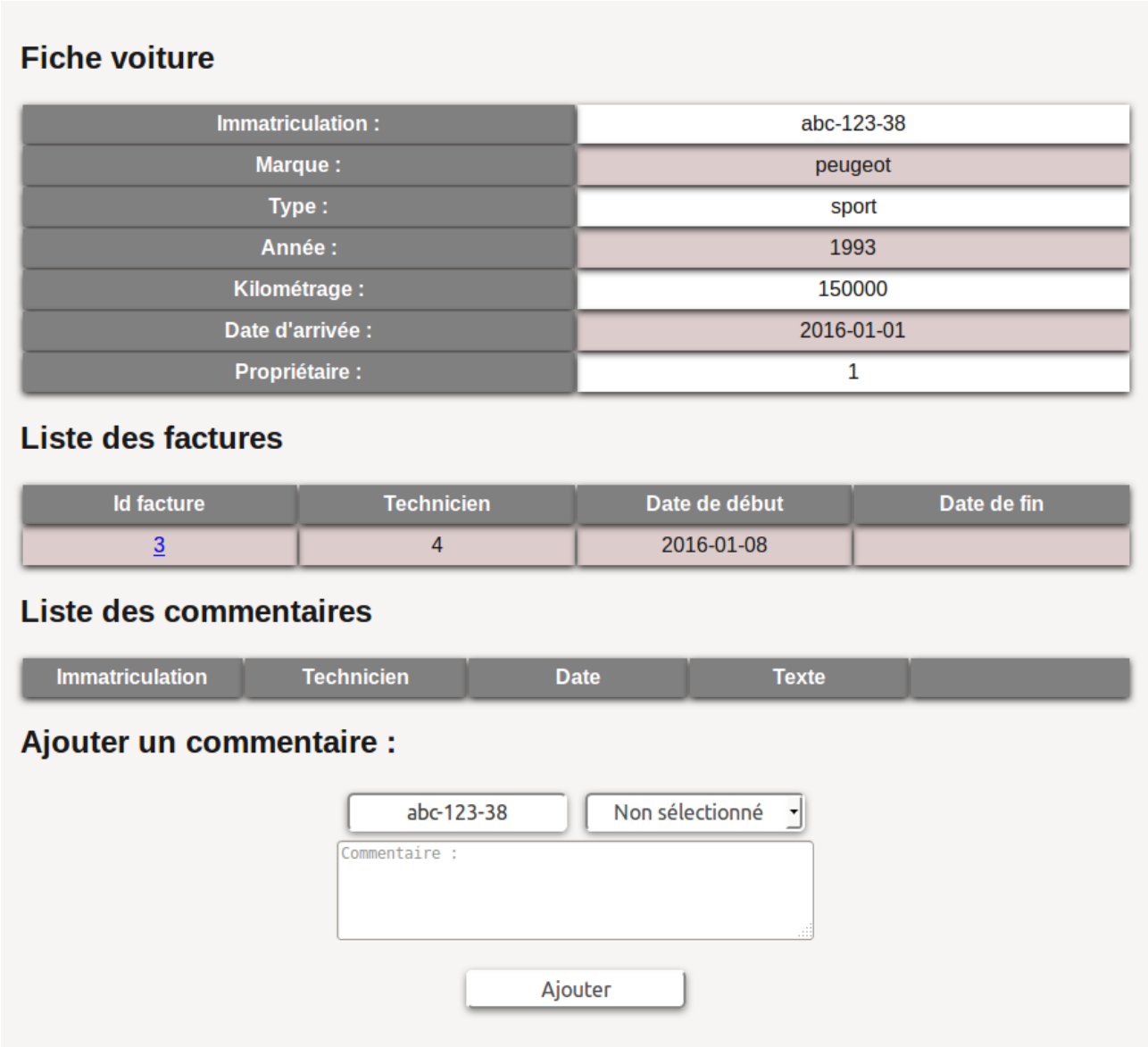
2 – La gestion des voitures

On peut rechercher les voitures selon tous les critères que celles-ci possèdent. On aura alors une liste des voitures avec leurs paramètres. On peut, pour chacune, la modifier ou la supprimer si on est identifié en tant que référent ou admin.



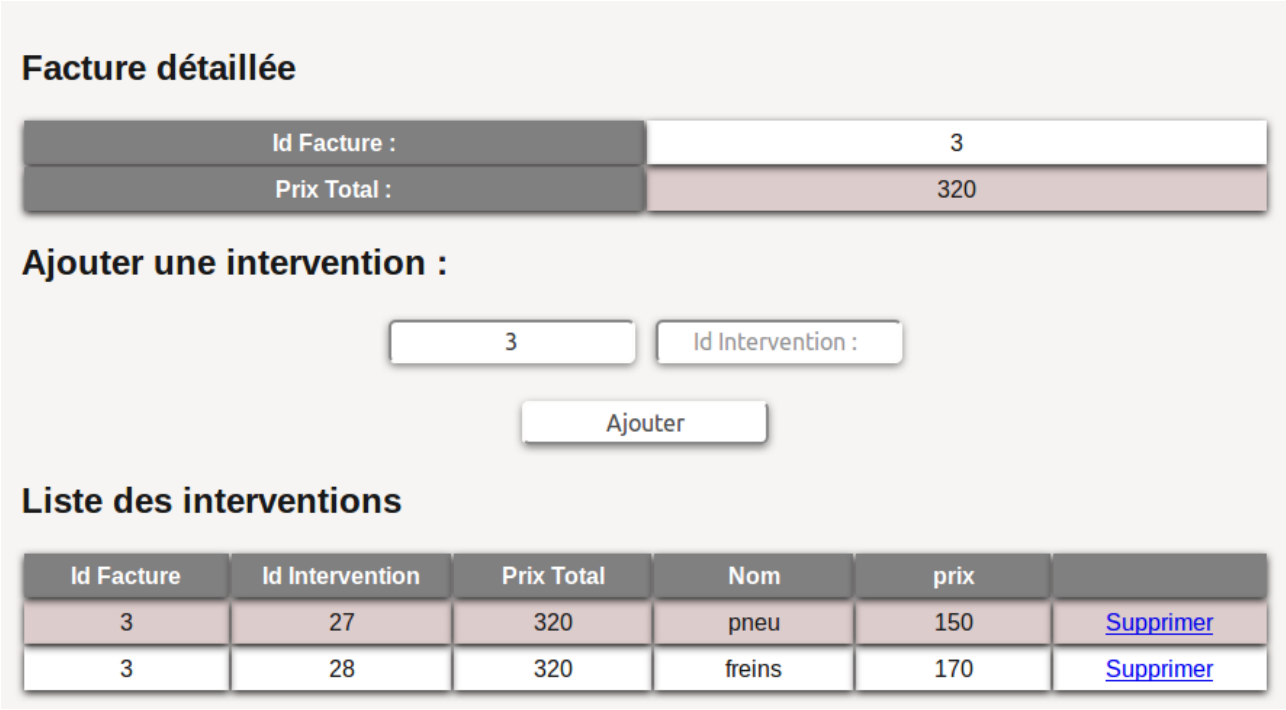
Capture d'écran de l'onglet voitures

Si on clic sur l'immatriculation de la voiture, on accède à la fiche de la voiture. Celle-ci se compose de la liste des attributs de la voiture, mais aussi de la liste de factures associées, et de la liste des commentaires. On peut ajouter un commentaire si on est identifié en temps que technicien ou admin.



Capture d'écran de l'écran d'une fiche technique d'une voiture

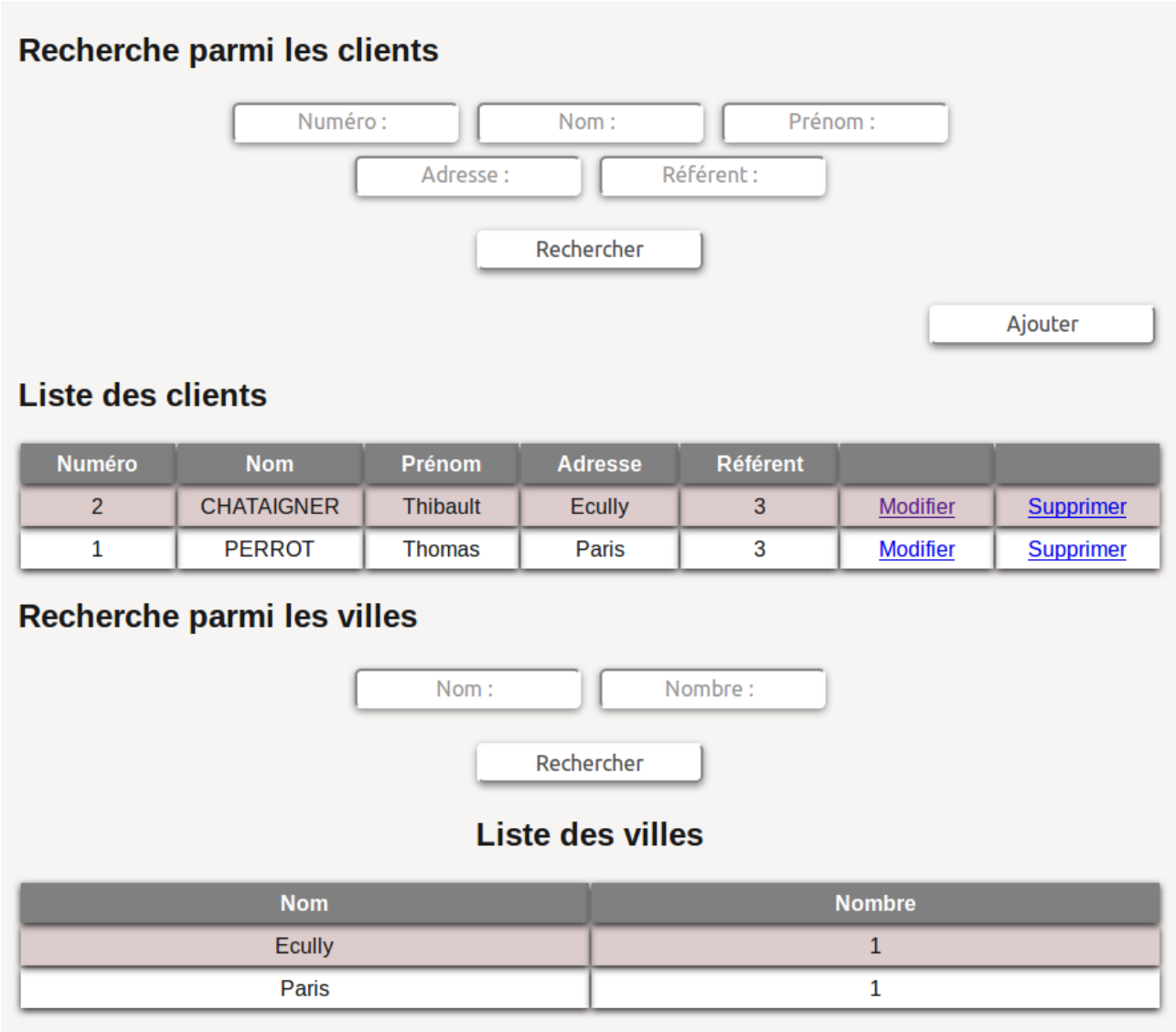
En cliquant sur l'id de la facture, on accède au détail de celle-ci. On trouve alors son identifiant, son prix total, et la liste des interventions qui ont été faites, avec les prix associés. Si on est connecté en tant qu'admin ou référent, on peut ajouter ou supprimer les interventions.



Capture d'écran d'une facture détaillée

3 – La gestion des clients

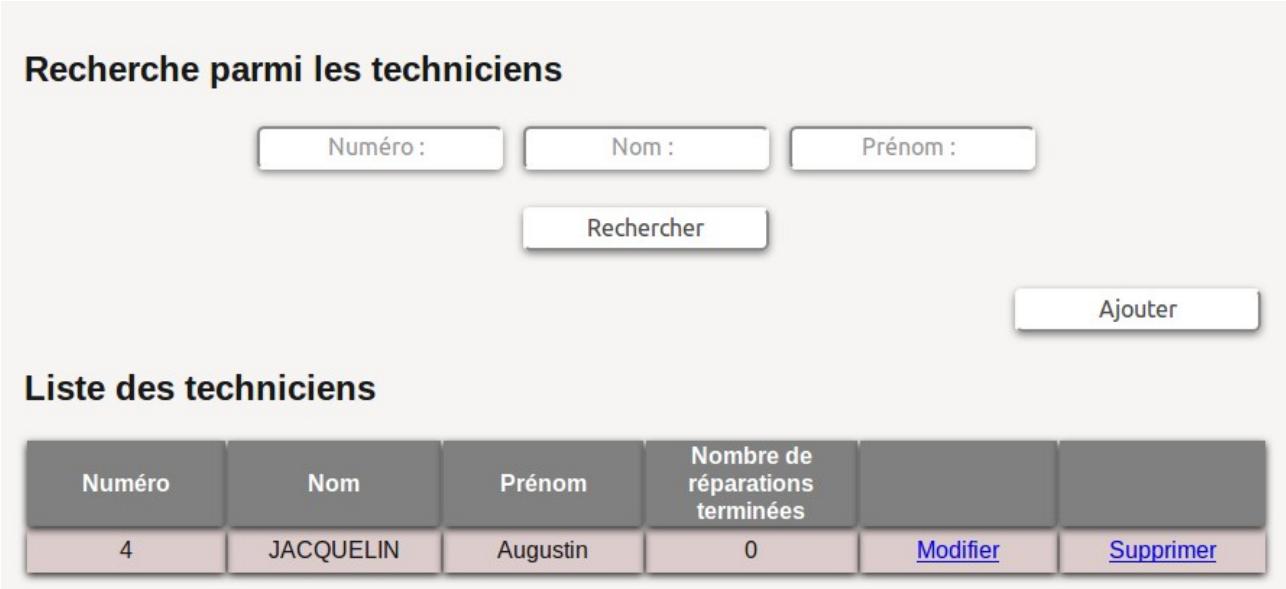
Sur la page de gestion des clients, on voit la liste des clients. On peut les modifier ou les supprimer si on a les droits nécessaires. Si on clic sur ajouter, un formulaire d'ajout d'un nouveau client s'ouvre. On peut également faire une recherche sur les villes dans lesquelles les clients sont présents. On affiche alors la ville et le nombre de clients qui y résident.



Capture d'écran de la gestion des clients

4 – La gestion des techniciens

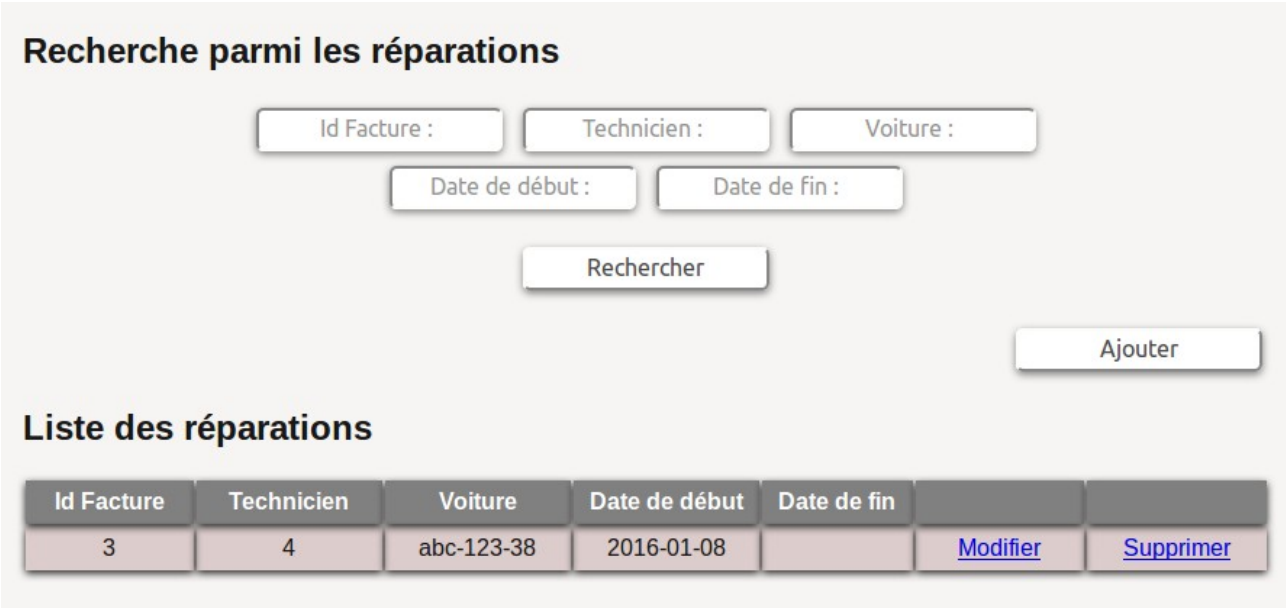
On peut rechercher les techniciens dans la base de donnée selon différents critères, et ajouter, modifier, ou supprimer des techniciens si on a les droits nécessaires.



Capture d'écran de la gestion des techniciens

5 – La gestion des réparations

On peut rechercher les réparations dans la base de donnée selon différents critères, et ajouter, modifier, ou supprimer des réparations si on a les droits nécessaires.



Capture d'écran de la liste des réparation

Conclusion

Au cours de projet, nous avons mobilisé nos connaissances afin de répondre à un problème complexe en étant acteurs de notre formation. Grâce à la modélisation de la base de donnée, puis à l'implémentation du site web associé, nous avons développé nos compétences dans des domaines essentiels pour les ingénieurs de demain. Ce site web, disponible sur un dépôt sur Github, est maintenant mis au service d'une large communauté, qui pourra exploiter et réécrire les codes créés afin de poursuivre et transcender ce magnifique projet.