



INF3710 –Fichiers et Bases de données

Hiver 2023

TP No. 4

Groupe 4

2147174 – Bakashov, Marsel

2154453 – Hamadache, Wassim

Soumis à : Charles de Lafontaine

14 avril 2023

Introduction résumant le projet et contributions

Le projet consiste à modéliser une base de données d'une coopérative de co-voiturage et à développer une application web permettant d'interroger cette base de données. Le projet comporte plusieurs étapes telles que la création d'un modèle entité-association, la conversion de ce modèle en un modèle relationnel avec la création d'une base de données SQL, la saisie de données et la création de requêtes SQL pour répondre à des besoins spécifiques, la création d'un déclencheur pour créer une table AVENDRE lorsque la valeur d'odomètre d'un véhicule spécifique est atteinte, l'analyse des dépendances fonctionnelles et la forme normale de la base de données, et enfin la création d'une application web pour interroger la base de données. La base de données a été modélisée avec PostgreSQL. L'objectif est de permettre aux membres de la coopérative de réserver et d'utiliser les véhicules appartenant à la coopérative. Les types de véhicules comprennent des voitures hybrides, des berlines et des minicamionnettes. Les véhicules doivent avoir une plaque d'immatriculation et une assurance automobile valide, et leur kilométrage doit être enregistré. Les emplacements de stationnement doivent être gérés pour chaque véhicule, ainsi que les informations sur les membres, y compris leur adhésion à la coopérative et leur permis de conduire valide. L'application web doit permettre la recherche d'information d'un membre, l'affichage et l'ajout d'une réservation.

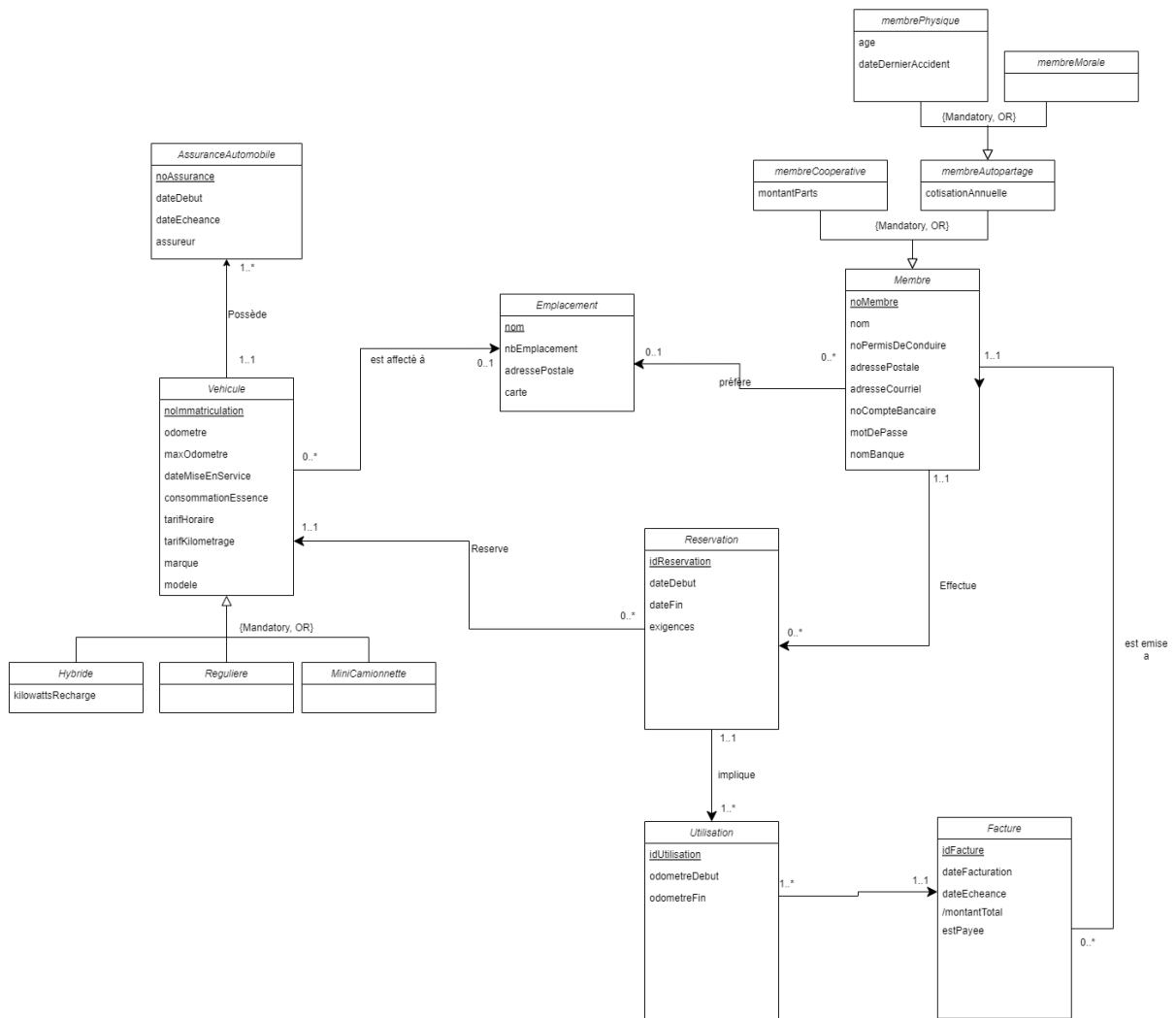
En ce qui concerne les contributions, Marsel s'est occupé de la modélisation en notation UML, de la conversion en modèle relationnel et de l'application Web. Wassim, quant à lui, s'est occupé de créer les requêtes demandées, d'implémenter le déclencheur, de lister les tables, les dépendances fonctionnelles et la forme normale de la base de données relationnelles obtenue. Finalement, l'entrée de données et le rapport a été complété ensemble.

Modèle conceptuel UML

- Les véhicules sont principalement achetés neufs et vendus dès qu'une valeur d'odomètre spécifique au type de véhicule est atteinte

Supposition

1. On peut savoir si le permis est valide avec le Numero de permis



Dépendances fonctionnelles et explication de la forme normale de la BD

Voici les dépendances fonctionnelles et la forme normale de chaque table :

Table **Emplacement** :

Dépendances fonctionnelles : nom → adresse, carte, nbEmplacement

Forme normale : 3NF

Table **Vehicule** :

Dépendances fonctionnelles : noImmatriculation → odometre, dateMiseEnService, consommationEssence, tarifHoraire, tarifKilometrage, marque, modele, Emplacement

Forme normale : 3NF

Table **Hybride** :

Dépendances fonctionnelles : noImmatriculation → kilowattsRecharge

Forme normale : 3NF

Table **Reguliere** :

Dépendances fonctionnelles : noImmatriculation → noImmatriculation (clé primaire)

Forme normale : 1NF

Table **MiniCamionnette** :

Dépendances fonctionnelles : noImmatriculation → noImmatriculation (clé primaire)

Forme normale : 1NF

Table **AssuranceAutomobile** :

Dépendances fonctionnelles : noAssurance → dateDebut, dateFin, assureur, noImmatriculation

Forme normale : 3NF

Table **Membre** :

Dépendances fonctionnelles : noMembre → nom, noPermisDeConduire, adressePostale, adresseCourriel, noCompteBancaire, nomBanque, motDePasse, emplacementFavori

Forme normale : 3NF

Table **MembreCooperative**

Dépendances fonctionnelles : noMembre → montantParts

Forme normale : 3NF

Table **MembreAutopartage** :

Dépendances fonctionnelles : noMembre → cotisationAnnuelle

Forme normale : 3NF

Table **MembrePhysique** :

Dépendances fonctionnelles : noMembre → age, dateDernierAccident

Forme normale : 3NF

Table **MembreMorale** :

Dépendances fonctionnelles : noMembre → noMembre (clé primaire)

Forme normale : 1NF

Table **Facture** :

Dépendances fonctionnelles : idFacture → dateFacturation, dateEcheance, noMembre, montantTotal, estPayee

Forme normale : 3NF

Table **Reservation** :

Dépendances fonctionnelles : idReservation → dateDebut, dateFin, exigences, noMembre, noImmatriculation

Forme normale : 3NF

Table **Utilisation** :

Dépendances fonctionnelles : idUtilisation → odometreDebut, odometreFin, idReservation, idFacture

Forme normale : 3NF

Pour chaque table, nous avons identifié les dépendances fonctionnelles entre les attributs.

Ensuite, nous avons vérifié si chaque table satisfait la 1ère forme normale (1NF). Une table est en 1NF si tous ses attributs sont atomiques, c'est-à-dire qu'ils ne peuvent pas être divisés en parties plus petites.

Nous avons vérifié si chaque table satisfait la 2ème forme normale (2NF). Une table est en 2NF si elle est en 1NF et que tous ses attributs qui ne font pas partie de la clé primaire dépendent fonctionnellement complètement de la clé primaire.

Nous avons vérifié si chaque table satisfait la 3ème forme normale (3NF). Une table est en 3NF si elle est en 2NF et que aucun attribut non primaire ne dépend transitivement de la clé primaire.

Si une table ne satisfait pas la 3NF, nous avons vérifié si elle satisfait la forme normale de Boyce-Codd (BCNF). Une table est en BCNF si pour chaque dépendance fonctionnelle non triviale (c'est-à-dire une dépendance où le déterminant n'est pas une clé candidate), le déterminant est une clé candidate.

En utilisant cette démarche, nous avons pu déterminer la forme normale de chaque table dans le schéma de base de données relationnelle fourni.

En résumé :

Tables Emplacement, Vehicule, Hybride, AssuranceAutomobile, Membre, MembreCooperative, MembreAutopartage, MembrePhysique, Facture, Reservation et Utilisation sont en 3NF.

Tables Reguliere, MiniCamionnette et MembreMorale ne sont qu'en 1NF.

Présentation de l'application développée avec copies d'écran permettant de démontrer toutes les fonctionnalités.

L'application Web utilisant la stack PEAN (PostgreSQL, Express, Angular et Node.js) doit permettre d'insérer et d'interroger les données de votre base de données. L'application doit permettre de rechercher l'information d'un membre et d'afficher les réservations ainsi que de permettre l'ajout d'un membre.

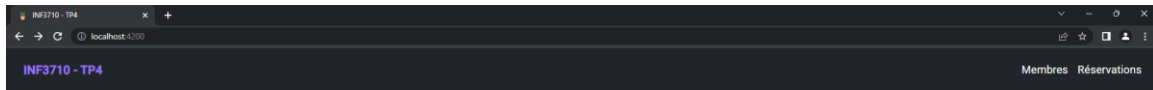
Recherche d'information d'un membre

En ce qui concerne la fonctionnalité de recherche d'informations de membre, tous les membres sont affichés sur la page "Membres" de l'application Web. Pour réaliser cette fonctionnalité, nous utilisons un service appelé "communication.service" via sa méthode "getMembres()", qui envoie une requête GET au serveur pour récupérer la liste de tous les membres.

Du côté du serveur, la requête est interceptée par le "database.controller", qui à son tour appelle la méthode "getAllMembres" du "databaseService". Cette méthode exécute une requête SQL "SELECT * FROM Coovoiturage_schema.Membre;" pour récupérer la liste de tous les membres stockés dans la base de données. Cette liste est ensuite renvoyée au client qui l'affiche sur la page "Membres".

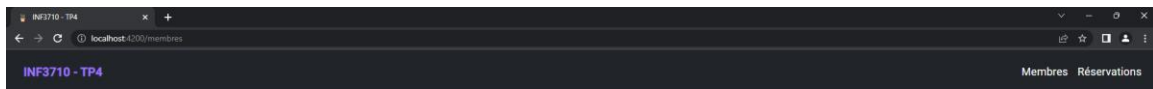
Pour la fonctionnalité de recherche, nous avons choisi de l'implémenter côté client plutôt que côté serveur, car la liste des membres était déjà affichée et il suffisait de la filtrer selon les entrées de l'utilisateur. Ainsi, lorsque l'utilisateur saisit un nom ou une partie de nom dans la barre de recherche, le client trie la liste des membres en utilisant les informations fournies par l'utilisateur et n'affiche que les résultats correspondants.

Cette approche offre une meilleure expérience utilisateur car les résultats de la recherche s'affichent instantanément et il n'y a pas de temps d'attente lié à une requête supplémentaire au serveur. Cependant, il est important de noter que cette approche peut devenir moins efficace si la liste des membres est très longue, auquel cas il pourrait être préférable de réaliser la recherche côté serveur.



Bienvenue
à
Autopartage

[Voir les réservations →](#)



Membres

Search Tapez le nom d'un membre ici et appuyez sur ENTER

No Membre	Nom	No Permis de Conduire	Adresse Postale	Adresse Courriel	No Compte Bancaire	Nom Banque	Mot de Passe	Emplacement Favori
987143	Muller Guerre	JF1038721K	93 rue Sevrin, Laval, QC, H9K1Y5	smallenergy@gmail.com	742824642	65473296	grangou431	Parc Ophile
452643	Jean-Claude Todibin	GHJU4932JM	667 rue Mangemeau, Laval, QC, K9S5S5	legrandj@gmail.com	674413234	98503845	motdepassecomplique	Place de la Chapelle
631571	Lucas Kapoera	NU183FN328	5532 rue Parlezom, Sorel-Tracy, QC, J3F4G6	magicmonster@gmail.com	315465553	48310481	mangezdupoulet	Plage Idéal

INF3710 - TP4

localhost:4200/membres

MembresRéervations

Membres

Search

lu

No Membre	Nom	No Permis de Conduire	Adresse Postale	Adresse Courriel	No Compte Bancaire	Nom Banque	Mot de Passe	Emplacement Favori
631571	Lucas Kapoera	NU183FN328	5532 rue Parlezcom, Sorel-Tracy, QC, J3F4G6	magicmonster@gmail.com	315465533	48310481	mangedupoulet	Plage idéal

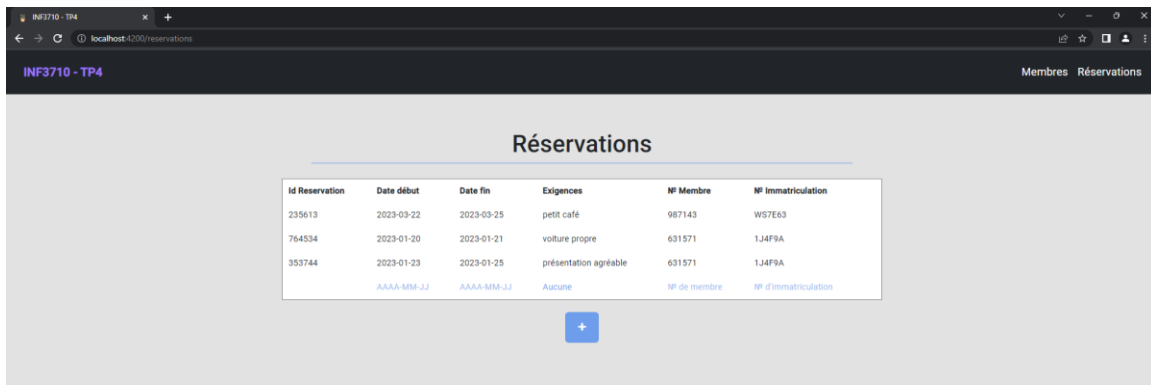
L'affichage et l'ajout d'une réservation

En ce qui concerne la fonctionnalité d'affichage et d'ajout de réservations, elle se déroule sur la page "Réservations" de l'application Web. Pour afficher toutes les réservations existantes, nous utilisons le service "communication.service" via la méthode "getReservations()", qui envoie une requête GET au serveur pour récupérer la liste de toutes les réservations. Pour ajouter une nouvelle réservation, nous utilisons la méthode "insertReservation(reservation: Reservation)" qui envoie une requête POST au serveur pour ajouter la nouvelle réservation à la base de données.

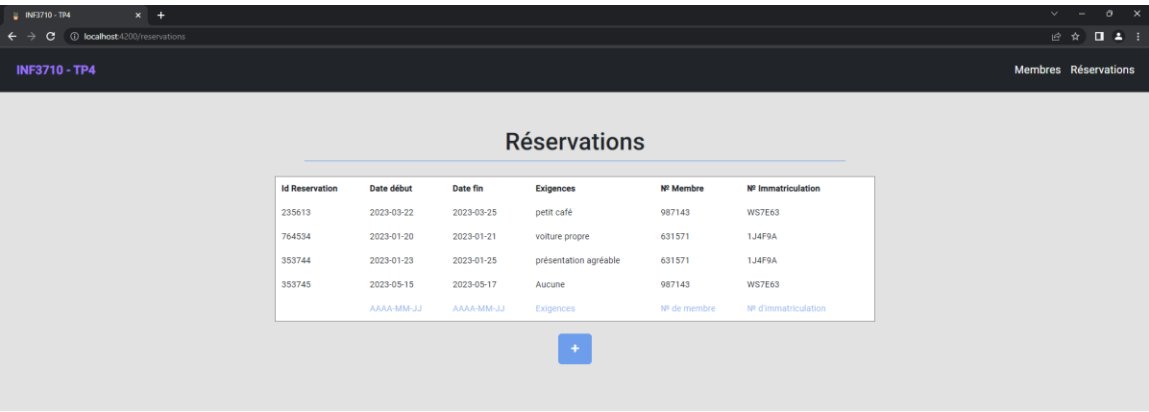
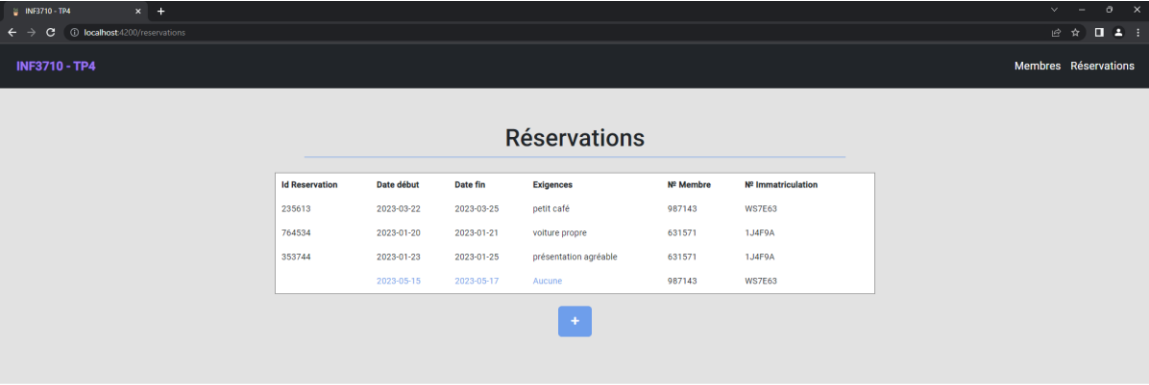
Côté serveur, la requête GET est interceptée par le "database.controller", qui appelle la méthode "getAllReservations()" du "databaseService". Cette méthode exécute une requête SQL "SELECT * FROM Coovoiturage_schema.Reservation;" pour récupérer la liste de toutes les réservations stockées dans la base de données. Cette liste est ensuite renvoyée au client qui l'affiche sur la page "Réservations".

La requête POST est interceptée par le "database.controller", qui à son tour appelle la méthode "createReservation" du "databaseService". Cette méthode exécute une requête SQL "INSERT INTO Coovoiturage_schema.Reservation VALUES(\$1,\$2,\$3,\$4,\$5,\$6);" pour ajouter la nouvelle réservation à la base de données avec les entrées de l'utilisateur reçues côté client. Toutefois, avant d'ajouter la réservation, nous effectuons quelques vérifications pour nous assurer que les données sont valides. Nous vérifions que toutes les entrées ont été reçues, que la date de fin n'est pas antérieure à la date de début et que la voiture n'est pas réservée pendant la période demandée par l'utilisateur.

Pour permettre à l'utilisateur d'ajouter une réservation, nous avons créé une section "Ajout" dans la page de Réservations. Cette section permet à l'utilisateur de remplir les champs nécessaires pour la création d'une nouvelle réservation. Une fois que l'utilisateur a rempli les champs et envoyé la demande, les données sont envoyées au serveur via la méthode "insertReservation" du "communication.service" et sont traitées comme expliqué précédemment. L'ID de la réservation n'est pas considéré comme une entrée, car il est incrémenté automatiquement. Nous avons également inclus des valeurs par défaut dans les champs du formulaire pour faciliter les tests du correcteur. Enfin, la page d'affichage de toutes les réservations permet à l'utilisateur de visualiser toutes les réservations existantes dans la base de données.



ID Reservation	Date début	Date fin	Exigences	N° Membre	N° Immatriculation
235613	2023-03-22	2023-03-25	petit café	987143	W57E63
764534	2023-01-20	2023-01-21	voiture propre	631571	1J4F9A
353744	2023-01-23	2023-01-25	présentation agréable	631571	1J4F9A
AAAA-MM-JJ	AAAA-MM-JJ	Aucune		N° de membre	N° d'immatriculation



Guide d'installation et de configuration qui permette d'installer et exécuter l'application

Pour utiliser l'application...

Assurez-vous d'avoir installé PostgreSQL.

Assurez-vous d'avoir installé Node.

- Allez dans /client et lancez la commande npm ci dans un terminal.
- Allez dans /server et lancez la commande npm ci dans un terminal.
- Allez dans /server/app/services/database.service.ts et modifiez connectionConfig avec les bons paramètres de votre BD.
- Sur PGAdmin, cliquez sur Query Tool et ouvrez le fichier bdschema.sql fourni et lancez-le.
- Insérez les données en répétant l'étape précédente avec le fichier data.sql fourni.
- Allez dans /client et lancez la commande npm start dans un terminal. (Si une question vous est posée dans le terminal, entrez 'n')
- Allez dans /server et lancez la commande npm start dans un terminal.

Exemple de configuration :

- Créez une base de données (le nom de votre choix) et postgres comme user
- Définissez le mot de passe de postgres à root (Login/Group Roles -> Click droit sur postgres -> Properties -> Definition)
- Respectez cette configuration pour la db dans VS Code dans le fichier database.service.ts
- user: "postgres",
- database: NOM DE LA DATABASE,
- password: "root",
- port: 5432,
- host: "127.0.0.1",
- keepAlive: true,
- Allez dans /server et faites la commande npm start dans un terminal. Le serveur est lancé au localhost:4000 par défaut.
- Allez dans /client et faites la commande npm start dans un terminal. Le client est lancé au localhost:4200 par défaut.