Documentation pour l’application pour VerbiageVoiture

I. Introduction

A. Contexte

Le projet de l’application VerbiageVoiture est réalisé par cinq étudiants à Grenoble-INP - Ensimag : Coralie Allioux, Tiphaine Helmer, Valentin Marull, Christophe Néraud et Augustin Remont.

Cette application consiste à mettre en lien des conducteurs et des passagers pour faire du covoiturage : de la proposition de trajet jusqu’au paiement du trajet, en passant par la recherche et le suivi d’un parcours.

B. Objectifs du document

Ce document décrit l'application de VerbiageVoiture sur un aspect technique et utilisateur.

Il est destiné d’une part à l’utilisateur final, grâce au manuel utilisateur (Partie II). D’autre part, l’équipe qui reprendra, modifiera, maintiendra l’application pourra se référer aux aspects techniques décrits (analyse, conception et implémentation des fonctionnalités).

II. Analyse du projet

L’analyse de projet est basée sur la description de l’application dans le sujet fourni, ainsi que les négociations et éclaircissants avec les clients.

A. Analyse statique

1. Inventaire des données

La liste des propriétés suivante est déduit de la description de l’application dans le sujet fourni :

{ Email, First_Name, Last_Name, Password, Wallet, Gender, License_Plate, Brand, Model, Fiscal_Power, Seats_Number, Id_Path, Start_Date, Hour, Seats_Number_At_Start, Id_Section, Distance, Estimated_Time, Waiting_Time, Id_Course}.

Le genre est ajouté par rapport à la spécification. Cette donnée permet de choisir l’image de profil de l’utilisateur dans l’application.

Id_Path est l’identifiant d’un trajet proposé par un conducteur.

Id_Section est l'identifiant d'un tronçon composant un trajet.
Id_Course est l'identifiant d'un parcours réservé par un passager.

2. Dépendances fonctionnelles

Voici la liste des dépendances fonctionnelles de la base de données :

- Email → First_Name, Last_Name, Password, Wallet, Gender
- License_Plate → Brand, Model, Fiscal_Power, Seats_Number
- Id_Path → Start_Date, Hour, Seats_Number_At_Start
- Id_Path, Id_Section → Distance
- Id_Path → Email
- Id_Path → License_Plate
- Id_Course → Email

3. Contraintes

a) … de valeur

Voici la liste des contraintes de valeur sur les données :

- Le nombre de places dans un véhicule ne peut être négatif ou nul : Seats_Number > 0
- La puissance fiscale d'un véhicule ne peut être nulle : Fiscal_Power > 0
- Le nombre de place au départ ne peut être supérieur ou égal au nombre de places du véhicule utilisé : Seats_Number_At_Start < Seats_Number
- La distance d'un tronçon est positive : Distance > 0
- Le temps d'attente d'un tronçon est positive mais peut être nulle : Waiting_Time >=0.
- Le temps estimé d'un tronçon est strictement positif : Estimated_Time > 0
- L'énergie d'un véhicule peut prendre les valeurs suivantes : { Diesel, Electrique, Essence }
- Le solde d'un porte monnaie utilisateur ne peut être négatif : Wallet >= 0.

b) … de multiplicité

Voici la liste des contraintes de multiplicité entre les données :

- Un utilisateur peut conduire plusieurs véhicules :
  Email →> License_Plate
- Un véhicule peut être conduit par un ou plusieurs utilisateurs :
  License_Plate →> Email
- Un trajet est constitué d'un ou plusieurs tronçons
  Id_Path →> Id_Path, Id_Section
- Un utilisateur peut proposer plusieurs trajets :
  Email →> Id_Path
- Un tronçon peut posséder ou non un temps d'attente :
  Id_Path, Id_Section -|-> Waiting_Time
- Un parcours est constitué d'un ou plusieurs tronçons :
  Id_Course →> Id_Path, Id_Section

c) … de contexte

Voici la liste des contraintes de contexte pour les données :

- Un parcours ne peut avoir des tronçons appartenant à plus de deux trajets différents.

- Dans le cas ou un parcours est composé de deux trajets, le temps d'attente entre les deux ne peut excéder 1 heure et les points d'arrivée et de départ ne peuvent être éloignés de plus de 1 km.

B. Analyse dynamique

1. Cas d'utilisations

a) Diagramme de cas d'utilisations

La liste détaillée des cas d'utilisations est donnée par le schéma ci-dessous. Certains cas d'utilisations sont commentés et se réfèrent à des diagrammes de séquence, présentés dans la section suivante.

CF DIAG CAS UTILISATIONS

b) Diagrammes de séquences

CF DIAGS DE SEQUENCE

2. Fonctionnement global de l'application

Le fonctionnement global de l'application est représenté par le schéma suivant :

CF DIAG A ETATS

Le diagramme à états illustre les différents processus de l'application :

- Ajouter un véhicule
- Ajouter du crédit au porte monnaie
- Proposer un trajet (de bout en bout)
- Réserver un parcours (de bout en bout)

III. Conception du projet

A. Conception de la base de données

1. Schéma Entités/Associations

a) Explication des choix

CF SCHEMA E/A

La ville et les coordonnées sont des entités extériorisés. Cela permet notamment de ne pas avoir des éléments en double dans la BDD et avoir une nomenclature cohérente.

L'identifiant d'un tronçon donne sa position dans la liste des tronçons composant le trajet.

L'annulation d'un tronçon, la confirmation d'arrivée et la confirmation de départ ne peuvent être faites que par le conducteur qui a proposé le trajet concerné.

b) Contraintes non représentées

- Les tronçons d'un parcours peuvent appartenir à 2 trajets différents maximum.

- Le solde de porte monnaie d'un utilisateur est positif : *Wallet >= 0*.
- Le nombre de place au départ d'un trajet doit strictement inférieur au nombre de places du véhicule utilisé : *Seats_Number_At_Start < Seats_Number*.
- La puissance fiscale d'un véhicule est strictement positive : *Fiscal_Power > 0*.
- L'énergie d'un véhicule peut prendre les valeurs : *{ Essence, Diesel, Electrique }*.
- La distance parcourue d'un tronçon est strictement positive : Distance > 0.
- Le temps estimé d'un tronçon est strictement positif : Estimated_Time > 0.
- Le temps d'attente entre deux tronçons est strictement positif : Waiting_Time > 0.
- Un parcours peut être composé de deux trajets seulement si l'attente <1h et que la distance gps est < 0,01.
- Un utilisateur ne peut pas rejoindre un trajet où il n'y a pas de place.
- Un utilisateur ne peut pas proposer un trajet (conducteur) ET réserver un parcours appartenant à ce trajet (passager).
- La validation des passagers pour chaque tronçon ne peut être fait qu'une fois la validation effectuée par le conducteur.
- La validation d'arrivée, de départ ou l'annulation d'un tronçon ne peut être faite que par le conducteur qui a proposé le trajet.

2. Traduction en relationnel

a) Entités simples

- UserVV(Email, First_Name, Last_Name, Password, Wallet, Gender)
- Vehicle(License_Plate, Brand, Model, Fiscal_Power, Seats_Number)
- Path(Id_Path, Start_Date, Hour, Seats_Number_At_Start)
- Course(Id_Course)
- City(City_Name, Postal_Code)
- Coordinates(Latitude, Longitude)
- Energy(Energy_Name)

b) Entités faibles

- Section(Id_Path, Id_Course, Distance, Estimated_Time, Waiting_Time)

c) Associations …

(1) … avec cardinalités 1..1

Les attributs en bleu sont des clés étrangères, ajoutés aux entités simples présentées plus haut.

- Path(Id_Path, Start_Date, Hour, Seats_Number_At_Start, License_Plate, Email)
- Vehicle(License_Plate, Brand, Model, Fiscal_Power, Seats_Number, Energy)
- UserVV(Email, First_Name, Last_Name, Password, Wallet, Gender, City_Name, Postal_Code)
- Course(Id_Course, Email)
- Coordinates(Latitude, Longitude, City_Name, Postal_Code)
- Section(Id_Path, Id_Course, Distance, Estimated_Time, Waiting_Time, Latitude_Begin, Longitude_Begin, Latitude_End, Longitude_End)

(2) … avec cardinalités 0..1

- Is_Cancelled(Id_Path, Id_Section, Email)
- Is_Arrived(Id_Path, Id_Section, Email)
- Is_Gone(Id_Path, Id_Section, Email)

- Is_Payed(Id_Course, Email)

(3) … avec cardinalités ?..*

- Can_Drive(Email, License_Plate)
- Is_Reserved(Id_Course, Id_Path, Id_Section)
- Is_In_Car(Id_Path, Id_Section, Id_Course) : Seulement une montée par trajet
- Is_Out_Car(Id_Path, Id_Section, Id_Course) : Seulement une descente par trajet

B. Conception de l'application

1. Architecture MVC

L'application est sous forme d'une architecture MVC (Modèle-Vue-Contrôleur). Les parties suivantes la présente de façon générale. Le diagramme de classe complet de l'application sont données en annexe.

CF SCHEMA ARCHI LOGICIELLE (lien entre les différentes paquetages)

2. Paquetage Vue

Le format de l'interface utilisé est JavaFX, avec un fichier FXML. Le principe est de décrire une vue (soit une page) à l'aide d'un fichier .fxml. Il est construit pour cette application, à l'aide du logiciel Scene Builder.

Chaque vue est représentée par un fichier .fxml.

CF SCHEMA ZOOM VUE

L'équivalent de ce package se situe dans l'arborescence suivante : *src/main/resources/fr/ensimag/equipe3/controller/*. Le package se nomme *controller* pour que les contrôleurs associés puissent y accéder.

3. Paquetage Contrôleur

Chaque vue (soit chaque fichier fxml) est associée à un contrôleur qui permet de demander une mise à jour des données dans l'interface. Il demande au modèle de lui fournir les données à jour contenues dans la base de données.

Une classe ViewControleur permet de choisir la vue à afficher dans l'interface : elle détruit l'ancienne vue et crée la nouvelle.

CF SCHEMA ZOOM CONTROLEUR

4. Paquetage Modèle

Le modèle contient une architecture en couches.

La première contient les classes métiers de l'application.

La deuxième couche est un DAO (Data Access Object) permettant d'abstraire les requêtes à la base de données. Un objet d'une classe métier pourrait interagir avec la BDD uniquement via son DAO associé.

La troisième est JDBC, déjà fourni par Java. C'est un paquetage extérieur à l'application. Elle permet d'accéder à la dernière couche, la base de données Oracle qui, elle aussi, est extérieure à l'application.

CF SCHEMA ZOOM MODELE

IV. Implémentation de l'application

A. Fonctionnalités implémentées

Chaque fonctionnalité implémentée est explicitée dans les sections ci-dessous. Elle est accompagnée des fichiers qui permettent de la définir (vue, contrôleur et classes métiers utilisées). Dans le cas d'un formulaire, les valeurs attendues sont décritent. Si le données fournies ne correspondent pas aux contraintes de valeur, un message d'erreur s'affiche. Les requêtes SQL nécessaires sont également listées.

1. Inscription d'un nouvel utilisateur

a) Ressources concernés

La vue concernée par l'inscription est le fichier **signupScreen.fxml**, contrôlé par **SignupController**. Le contrôleur utilise les classes métiers **User**, **City** et **Gender**.

b) Données attendues dans le formulaire

L'inscription d'un nouvel utilisateur demandes les informations suivantes :

Intitulé du champs	**Valeur attendu**	**Commentaires**
Adresse e-mail	Format d'une adresse email	L'existence de l'adresse n'est pas vérifiée
Nom	Chaîne de caractères	Ne peut contenir des chiffres
Prénom	Chaîne de caractères	Ne peut contenir des chiffres
Genre	Homme, Femme ou non spécifié	Choix avec des boutons radios
Code Postal	5 chiffres consécutifs	
Ville de résidence	Chaîne de caractères	A choisir parmis une liste déroulante, associée au code postal précédemment renseigné
Mot de passe	Chaîne de caractères	Minimum 8 caractères, avec au moins une minuscule,

		une majuscule et un caractère spécial
Confirmation du mot de passe	Chaîne de caractères	Doit être la même que le champs Mot de passe

c) Requêtes SQL

- Dans UserDAO, permet de vérifier si l'email indiqué existe déjà ou non dans la base de donnée
  - SELECT * FROM UserVV WHERE email = *valeur*
  - Si l'email est déjà utilisé, un message d'erreur apparaît pour indiquer que l'utilisateur existe déjà.
- Dans UserDAO, permet d'ajouter un nouvel utilisateur dans le BDD
  - INSERT INTO UserVV VALUES(?, ?, ?, ?, ?, ?, ?, ?)
- Dans CityDAO, permet de vérifier si la ville existe ou non dans la BDD
  - SELECT * FROM City WHERE City_Name = ? AND Postal_Code = ?
  - Si la ville n'existe pas, elle est ajoutée dans la BDD
- Dans CityDAO, permet d'ajouter une nouvelle ville dans le BDD
  - INSERT INTO City VALUES(?, ?)

2. Connexion d'un utilisateur

a) Ressources concernés

La vue associée est **loginScreen.fxml**, contrôlée par **LoginController**. Il utilise la classe métier **User**.

b) Données attendues dans le formulaire

Les données attendues permettent d'authentifier un utilisateur sur l'application :

Intitulé du champs	**Valeur attendu**	**Commentaires**
Adresse e-mail	Format d'une adresse email	
Mot de passe	Chaîne de caractères	

L'adresse email et le mot de passe doivent être les mêmes que dans la base de données. Si c'est un couple qui n'existe pas, alors un message d'erreur apparaît pour informer l'utilisateur que les identifiants sont incorrects.

c) Requêtes SQL

- Dans UserDAO, permet de vérifier si l'email indiqué existe la base de donnée
  - SELECT * FROM UserVV WHERE email = *valeur*
  - Si l'email n'existe pas, un message d'erreur apparaît pour indiquer que les identifiants entrés sont incorrects.

3. Gestion des véhicules d'un utilisateur

a) Liste des véhicules

(1) Ressources concernées

La vue correspondante est **vehiclesScreen.fxml**, contrôlée par **VehiclesController**. Il utilise la classe métier **User**.

(2) Requêtes SQL

- Dans UserDAO, permet d'avoir la liste des véhicules que peut conduire l'utilisateur courant :
  - SELECT * FROM Can_Drive WHERE email = ?

b) Ajout d'un véhicule

(1) Ressources concernées

La vue pour ajouter un véhicule est **newVehicleScreen**.fxml, contrôlé par **NewVehicleController**. Il utilise les classes métiers **Energy**, **User** et **Vehicle**.

(2) Données attendues dans le formulaire

Intitulé du champs	**Valeur attendu**	**Commentaires**
Immatriculation	Chaîne de caractères	Au maximum 10 caractères
Marque	Chaîne de caractères	
Modèle	Chaîne de caractères	
Energie	Essence, Diesel ou Électrique	Choix parmis une liste déroulante
Puissance fiscale	Entier	Positif non nul
Nombre de places	Entier	Positif non nul, strictement inférieur au nombre de place du véhicule sélectionné

L'immatriculation n'a pas de vérification stricte car les plaques d'immatriculation étrangère n'ont pas le même format. En conséquence, l'application accepte tout type de plaque d'immatriculation.

(3) Requêtes SQL

- Dans EnergyDAO, permet de récupérer la liste des énergies possible pour un véhicule :
  - SELECT * FROM Energy
  - S'il n'y a pas d'énergie dans la BDD, un message d'erreur s'affiche.
- Dans VehicleDAO, permet de vérifier si la plaque d'immatriculation est déjà utilisée ou non :
  - SELECT * FROM Vehicle WHERE License_Plate = ?

- ➢ Si la plaque d'immatriculation existe déjà, c'est le véhicule dans la BDD qui est conservé. L'application ajoute ce véhicule dans la liste de l'utilisateur courant, même si toutes les informations ne sont pas rigoureusement identiques. Cela évite qu'un utilisateur puisse modifier le véhicule d'un autre utilisateur.

❖ Dans VehicleDAO, permet d'ajouter dans la BDD le nouveau véhicule s'il n'existe pas :
- ➢ INSERT INTO Vehicle VALUES(?, ?, ?, ?, ?, ?)

❖ Dans UserDAO, permet d'ajouter la nouvelle voiture à l'utilisateur dans la BDD :
- ➢ INSERT INTO Candrive VALUES(?, ?)

c) Suppression d'un véhicule

(1) Ressources concernées

La suppression peut se faire via la vue qui liste les véhicules d'un utilisateur. Un clic droit sur le véhicule fait apparaître un bouton pour supprimer le véhicule.

(2) Requêtes SQL

❖ Dans UserDAO, permet de supprimer l'association entre l'utilisateur courant et le véhicule supprimé :
- ➢ DELETE FROM Candrive WHERE License_Plate = ? AND Email = ?

❖ Dans le cas où le véhicule ne peut être conduit par personne, il est supprimé de la BDD depuis VehicleDAO :
- ➢ DELETE FROM Vehicle WHERE License_Plate = ?

4. Gestion du porte monnaie

L'utilisateur peut créditer et débiter son porte monnaie à l'aide d'un numéro de carte. Cette fonctionnalité est simulée. La numéro de carte n'est pas stocké dans l'application ou dans la BDD. Le crédit et le débit sont instantanés.

a) Ressources concernées

La vue est décrite par **wallet.fxml**, contrôlée par **WalletController**. Le contrôle utilise uniquement la classe métier **User**.

b) Données attendues dans le formulaire

Intitulé du champs	**Valeur attendu**	**Commentaires**
Montant	Nombre avec ou sans décimal	- Limité à 2 décimales dans la BDD - Strictement positif - Dans le cas d'un débit, ne peut être supérieur au solde
Numéro de carte	Nombre entier	Exactement 16 chiffres

c) Requêtes SQL

- Dans UserDAO, permet de mettre à jour dans la base de données le solde du porte monnaie de l'utilisateur :
  - UPDATE UserVV SET Wallet = ? WHERE Email = ?

5. Proposition d'un trajet

a) Ressources concernées

La vue utilisée est **newPath.fxml**, contrôlée par **NewPathController**. Il utilise les classes métiers **Path**, **Section**, **Coordinates** et **User**.

b) Données attendues dans le formulaire

Les champs suivants sont nécessaires pour la proposition d'un trajet :

Intitulé du champs	**Valeur attendu**	**Commentaires**
Véhicule	Immatriculation d'un véhicule que l'utilisateur courant peut conduire	Choix parmi une liste déroulante
Nombre de places libres	Nombre entier	- Strictement positif - Ne peut dépasser la capacité de la voiture sélectionnée (sans compter la place du conducteur)
Date	Jour, mois et année	- Choix parmi un sélectionneur de date - Ne peut être antérieur à la date du jour
Heure	2 chiffres pour les heures, deux points, 2 chiffres pour les minutes HH:MM	- Heures comprises entre 00 et 23 - Minutes comprises entre 00 et 59
Latitude (de départ et prochaine étape)	Entier ou flottant	
Longitude (de départ et prochaine étape)	Entier ou flottant	
Temps estimé	Nombre entier	En minutes
Distance	Nombre entier	En kilomètres

Les coordonnées sont automatiquement associée à une ville.

Pour ajouter des tronçons au trajet, le formulaire avec les données suivantes est à remplir :

Intitulé du champs	**Valeur attendu**	**Commentaires**
Latitude prochaine étape	Entier ou flottant	
Longitude prochaine étape	Entier ou flottant	
Temps estimé	Nombre entier	En minutes
Distance	Nombre entier	En kilomètres
Temps d'attente	Nombre entier	- En minutes - Optionnel

c) Requêtes SQL

❖

6. Recherche d'un parcours

a) Ressources concernées

La vue concernée est **searchPath.fxml**, contrôlée par **SearchPathController**. Il utilise les classes métier **City**.

b) Données attendues dans le formulaire

Intitulé du champs	**Valeur attendu**	**Commentaires**
Code postal (départ et arrivée)	Nombre à 5 chiffres	
Ville (départ et arrivée)		Liste déduite du code postal : utilisation d'une API

c) Requêtes SQL

❖ Dans CityDAO, permet de vérifier si des trajets existes avec la ou les villes données :
- ➢ SELECT * FROM City WHERE City_Name = ? AND Postal_Code = ?
- ➢ Si la ville n'est pas spécifiée dans la BDD, alors un message d'erreur s'affiche pour informer l'utilisateur qu'aucun parcours ne correspond à sa recherche.

7. Réservation d'un parcours

B. Eléments non implémentés

C. Explication de choix d'implémentation

1. Gestion des clés primaires composées de plusieurs attributs

L'interface **DAO** représente la clé primaire de la table comme un unique objet. Or, il y a des tables dans la base de données qui possède une clé primaire composée de plusieurs attributs. Tels que **Coordinates** (*Longitude*, *Latitude*), **City** (*City_Name*, *Postal_Code*) et **Section** (*Id_Path*, *Id_Section*).

Pour respecter la signature imposée par l'interface, il faut avoir alors un seul objet pouvant représenter un couple de clé. L'objet **Pair** est utilisé à cet effet.

Par exemple, la clé de **Coordinates** est : **Pair**<*Double*, *Double*> id, avec les deux types associés respectivement à *Latitude* et *Longitude*.

2. GeoApi : cohérence de données entre les coordonnées, la ville et le code postal

a) Correspondance Code postal / Ville
b) Correspondance Coordonnées / ville

https://geo.api.gouv.fr/adresse

3. Validation de formulaire
   a) Validator
   b) ValidateForm

V. Conclusion

A. Bilan critique

1. Objectifs atteints
2. Objectifs non atteints

B. Perspectives et améliorations

VI. Bilan de gestion

Cette section décrit la gestion de projet au sein de l'équipe.

A. Organisation

Chaque semaine, 1 à 2 réunions sont mises en place, avec pour objectifs :

- Faire un point sur l'état d'avancement du projet : qui a fait quoi
- Remonter les problèmes (difficulté sur la tâche attribuée par exemple)
- Essayer de trouver des solutions à ces problèmes
- Redéfinition des objectifs et répartition les prochaines tâches à faire

Ces réunions permettent de rythmer le groupe et de ne perdre personne en chemin.

B. Outils de communication

Lors de ce projet, l'équipe a mis en place un ensemble d'outils afin de communiquer : avec les membres ainsi qu'avec les enseignants. La distanciation due au contexte de la crise sanitaire a renforcé la prise de conscience sur l'importance de la communication et de l'utilisation des outils présentés ci-après.

1. Discord

Discord est un outil permettant de communiquer en vocal ou par texte. Il permet également de faire des appels avec des partages d'écran, ce qui est très pratique en cas de problème. Par exemple, pour des erreurs de compilation sur le projet ou un problème d'installation et de dépendances de l'IDE.

2. Trello

Trello est un gestionnaire de tâche agile. L'équipe l'utilise à partir de l'implémentation de l'application. Il permet de répartir entre les membres les cas d'utilisations à implémenter.

3. Riot

Riot est l'outil de communication entre les membres de l'équipe et les enseignants/clients. Il permet de poser des questions mais aussi de valider certaines parties du projet.

C. Points positifs

Le début du projet a été très positif : le sujet a plu à l'ensemble de l'équipe et le travail a commencé dès la première séance encadrée. L'équipe a effectué les étapes d'analyse rapidement, en équipe de deux.

L'organisation s'est bien installée : des réunions régulières avec des reviews du travail effectué pour que chaque membre puisse prendre conscience et amélioration chaque partie du projet. Le discord est un bon moyen de communiquer à distance, autant textuel que vocal. Il a été accepté rapidement par tous les membres.

Plusieurs membres sont restés actifs et motivés jusqu'au bout, avec un expert en conception et un autre en implémentation. Les autres membres ont pu s'accrocher en se faisant aider de ces deux membres.

D. Difficultées rencontrées et analyse

La phase d'analyse et de conception de la base de données a été laborieuse, nécessitant plusieurs essais. Les équipes ont été inversées pour résoudre le problème plus rapidement et éviter de décourager l'équipe en charge. Cette phase a mis en retard l'équipe et a cassé le rythme en place, ceci créant de la frustration.

Ce que nous pouvons retenir est de toujours avoir des méthodes rigoureuses pour ce genre de tâche. Cette phase est le pilier pour la suite : elle ne peut se permettre d'avoir des failles. Le fait d'avoir recommencer plusieurs fois la démarche est dû au fait d'avoir oublié des détails qui ont cassé la compréhension de la suite...

Il y a eu une baisse d'efficacité, de présence et de motivation au fil du projet. La dernière ligne droite pour le rendu du projet a été confrontée à plusieurs membres du groupe inactifs. Malgré les réunions régulières, les objectifs ont petit à petit été repoussées, faute de temps ou de motivation...