Manuel technique

Moto Care C#

CFPT - TPI

Wohlers Luca

Table des matières

[Introduction 2](#_Toc42513568)

[Rappel de l’énoncé 2](#_Toc42513569)

[Organisation 2](#_Toc42513570)

[Livrables 2](#_Toc42513571)

[Matériel et logiciels à disposition 2](#_Toc42513572)

[Description de l’application 3](#_Toc42513573)

[Méthodologie 5](#_Toc42513574)

[Planification 7](#_Toc42513575)

[Interfaces 10](#_Toc42513576)

[Liste de fonctionnalités disponibles 10](#_Toc42513577)

[Présentation de l’interface 11](#_Toc42513578)

[La fiche frmMain 11](#_Toc42513579)

[Généralités concernant l’implémentation 17](#_Toc42513580)

[Base de données 17](#_Toc42513581)

[Classes 19](#_Toc42513582)

# Introduction

Ce document est un rapport présentant différents aspects de la conception du projet « Moto Care ». Ce projet a été réalisé dans le cadre du « Travail pratique individuel (TPI) » durant fin mai, début juin. Il a pour but de valider mes compétences acquises pendant la formation Informaticien CFC effectuée à l’école d’informatique du CFPT au Petit-Lancy.

Moto Care est une application C# qui permet à l’utilisateur de planifier des entretiens pour sa ou ses motos, d’indiquer les trajets effectués avec cette dernière et de placer sur une carte du monde les lieux favoris de l’utilisateur ou les endroits qu’il souhaite visiter.

# Rappel de l’énoncé

## Organisation

|  |
| --- |
| **Élève** |
| Luca Wohlers  <luca.whlrs@eduge.ch> |

|  |
| --- |
| **Maîtresse d’apprentissage** |
| Anne Terrier  <anne.terrier@edu.ge.ch> |

|  |  |
| --- | --- |
| **Experts** | |
| Borys Folomietow  <borys@folomietow.ch> | Alain Fontanini  <alain.fontanini@outlook.com> |

## Livrables

* Planning prévisionnel
* Rapport du projet
* Manuel utilisateur
* Journal de bord

## Matériel et logiciels à disposition

Étant donné que le TPI cette année s’effectue lors de la pandémie du Covid-19 le matériel est mon matériel personnel mais avec les outils utilisé à l’école.

* Un PC portable avec Windows 10, 2 écrans
* Visual Studio Community 2019
* La suite Office
* Git avec dépôt sur Github

## Description de l’application

Moto Care est une application C# destinée à un usage privé. Toute personne possédant l’application peut ajouter, modifier ou supprimer son véhicule. Il peut ajouter, modifier ou supprimer les trajets effectués. Ensuite, il peut ajouter, modifier ou supprimer des entretiens à effectuer sur le véhicule. Enfin, Il y a une carte du monde pour ajouter les points d’intérêts de l’utilisateur. Il n’y a pas de connexion nécessaire pour gérer les entretiens etc.

Les points suivants doivent être respectés :

* **La solution est fonctionnelle**

La solution finale fonctionne correctement.

* **La base de données est correctement implémentée. Il y a au moins 1 véhicule, 5 maintenances, 2 trajets et 1 point d’intérêt**

Dans la base de données toutes les tables doivent être remplies. Le nombre d’enregistrement correspond plus ou moins à l’importance de la table. Au moins, 1 enregistrement pour la table véhicule, 5 enregistrements pour celle des maintenances, 2 pour celle des trajets et 1 pour les points d’intérêts.

* **L’utilisateur peut consulter son carnet d’entretien**

Dans un des onglets l’utilisateur peut consulter tous les entretiens à venir ou ceux déjà effectués. Il a aussi la possibilité de cocher les entretiens qui viennent d’être fait.

* **L’utilisateur peut ajouter/supprimer/modifier les entretiens pour un véhicule**

L’utilisateur voit la liste des entretiens et il peut pour chaque entretien, le modifier ce qui va modifier les valeurs dans la base de données. Il peut le supprimer, ce qui enlèvera l’enregistrement de la base de données. Finalement il peut créer un nouvel entretien.

* **L’utilisateur peut ajouter/supprimer/modifier un véhicule**

L’utilisateur peut ajouter un nouveau véhicule en renseignant les champs nécessaires. Il peut le modifier, les champs sont préremplis avec les valeurs actuellement dans la base de données. Il peut également supprimer un véhicule de la base de données ce qui supprimera aussi tous les autres enregistrements liés à ce véhicule. Tous les trajets et entretiens appartenant au véhicule supprimé, seront également supprimés.

* **L’utilisateur peut ajouter/supprimer/modifier un trajet**

L’utilisateur peut ajouter un trajet qu’il a effectué avec un véhicule. Il a la possibilité de le modifier en ayant les champs préremplis avec les valeurs actuellement dans la base de données. Il peut aussi le supprimer définitivement. Tout ajout/suppression/modification de la distance du trajet engendre une modification du kilométrage du véhicule.

* **La solution respecte une implémentation MV**

La façon dont le code a été écrit est en Modèle-Vue, c’est-à-dire qu’on sépare le modèle de la vue. Tout ce qui concerne l’affichage s’effectue à un endroit et ce qui concerne le programme se fait de l’autre côté.

# Méthodologie

Pour planifier mon projet de TPI, je me suis basé sur la méthode en 6 étapes.

Une image contenant texte, carte, signe

Description générée automatiquement

1. **S’informer**

La toute première étape de mon projet était la lecture en profondeur de mon énoncé pour comprendre toutes les fonctionnalités qu’il était nécessaire d’implémenter. J’ai également demandé à ma formatrice quelques clarifications sur certains détails lorsque c’était nécessaire.

1. **Planifier**

Dès le début du projet, j’ai préparé un planning de travail pour savoir ce qu’il fallait faire et quand le faire. J’ai donc séparé mon travail en sous-tâches importantes. Une fois les tâches importantes isolées j’ai créé un diagramme de Gantt pour visualiser au quotidien ma progression dans le travail ainsi que les différences entre ma planification et mon avancement effectif.

1. **Décider**

Au cours de l’avancement de mon travail, j’ai dû prendre de nombreuses décisions sur la manière de le réaliser. Lorsque je faisais des choix que je jugeais importants pour le projet, j’en parlais dans mon journal de bord en expliquant les raisons qui m’ont poussé à faire ce choix en question.

1. **Réaliser**

Une fois les bonnes décisions prises, je les implémentais dans le code.

1. **Contrôler**

À chaque fois que je terminais une fonctionnalité, je la testais dans différents cas d’usage pour être sûr qu’elle fonctionnait comme prévu.

1. **Évaluer**

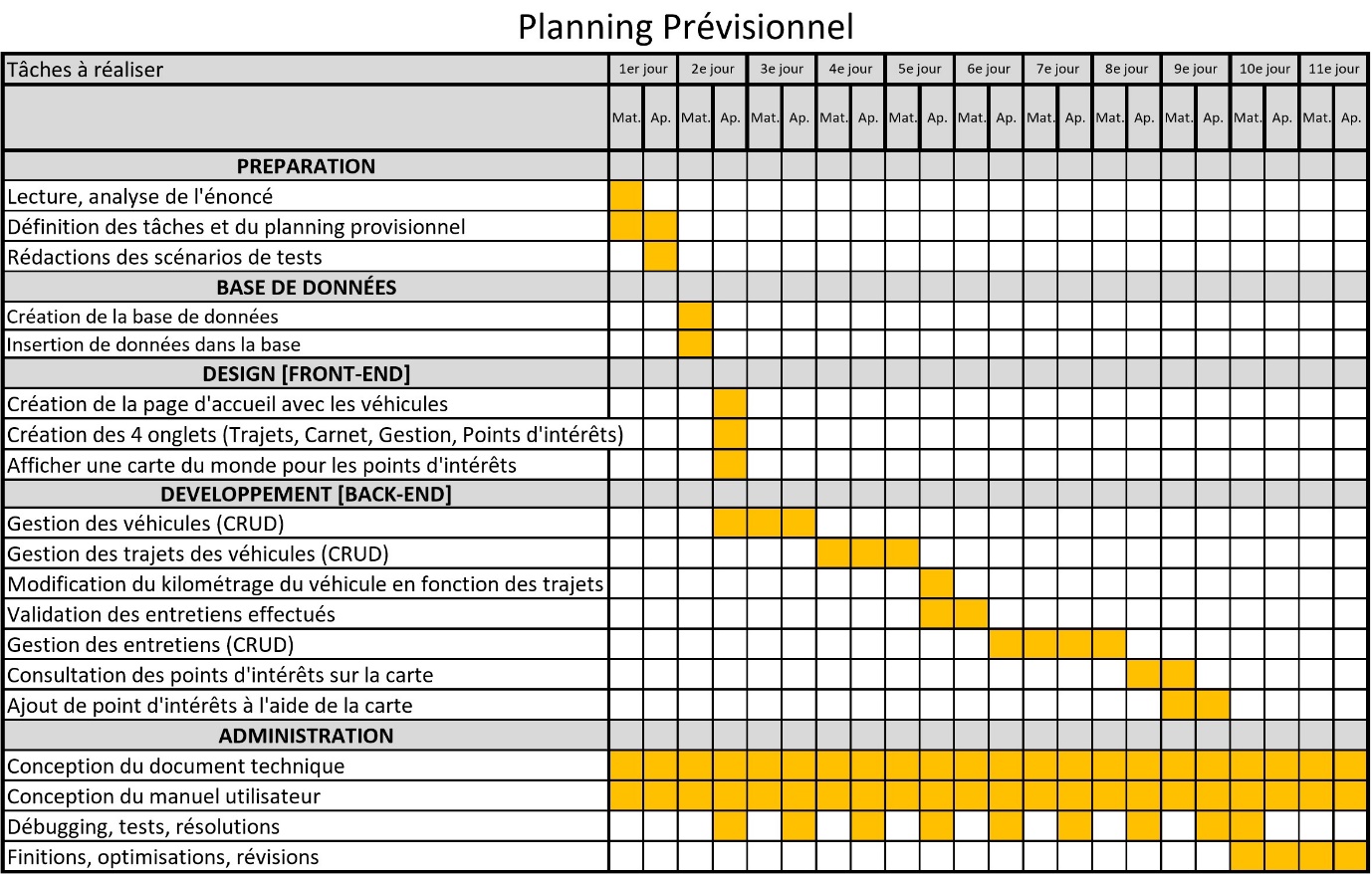
La dernière étape de la méthodologie en 6 étapes est l’évaluation, pour faire une rétrospective de ce que j’ai fait et avoir un regard critique sur ce qui pourrait être amélioré. Pour ce faire, j’ai écrit des bilans journaliers à chaque fin de journée de travail dans mon journal de bord.

# Planification

Pour choisir dans quel ordre effectuer mon projet j’ai tout d’abord relevé toutes les tâches importantes :

* Tout d’abord, ce qui concerne la base de données
  + Création de la base de données
  + Insertion de données
* Ensuite, ce qui est en lien avec le design
  + Création de la page principale
  + Création des 4 onglets
    - Trajet
    - Carnet
    - Gestion
    - Points d’intérêts
  + Affichage de la carte du monde
* Puis, le développement
  + Gestion des véhicules
  + Gestion des trajets
  + Modification du kilométrage du véhicule en fonction des trajets
  + Validation des entretiens effectués
  + Gestion des entretiens
  + Consultation des points d’intérêts sur la carte
  + Ajout de points d’intérêts sur la carte

Après avoir analysé les tâches importantes j’ai créé un diagramme de Gantt en les positionnant dans un ordre logique afin de voir plus grossièrement ce qu’il fallait que je fasse. Ensuite, j’ai estimé la durée que chaque tâche devrait me prendre pour ne pas prendre de retard. Le résultat est le suivant :



Le diagramme si dessus est le planning prévisionnel, c’est-à-dire qu’il s’agit du diagramme que j’ai créé avant même de commencer mon travail. Mais au fur et à mesure où j’avançais dans mon travail certaines tâches n’ont pas été effectuées au moment où je l’avais prévu. Par exemple j’ai fini le CRUD des trajets une demi-journée en avance. J’ai du coup commencé la modification du kilométrage du véhicule pas le 5ème jour dans l’après-midi mais le matin. Cela m’a aussi pris plus de temps que prévu car non seulement il m’a fallu me familiariser avec les checkBox et le dataGridView mais aussi réfléchir à une interface optimum pour le carnet d’entretien.

Ensuite, je me suis rendu compte que pour faire la validation des entretiens il fallait tout d’abords que je commence le CRUD des entretiens, au moins le « Read », autrement dit la lecture des entretiens à partir de la base de données.

Mais encore, la consultation des points d’intérêts et leur ajout à été nettement plus rapide que ce que j’avais défini. J’ai terminé les deux durant l’après-midi de la 8ème journée et non pas au cours de la 9ème.

Finalement, en ce qui concerne la documentation, je n’ai pas travaillé sur le manuel technique et utilisateur chaque jour, j’ai cependant évidemment mis chaque jour mon journal de bord à jour.

Voici donc le planning effectif de mon travail :

Une image contenant capture d’écran

Description générée automatiquement

# Interfaces

Cette section de la documentation traite la partie visible de l’application, vue par l’utilisateur.

Tout d’abord, sont énumérées les fonctionnalités disponibles dans l’application. Ensuite viennent les présentations des fenêtres de l’interface utilisateur.

## Liste de fonctionnalités disponibles

L’utilisateur peut :

* Véhicules
  + Ajouter un nouveau véhicule
    - Nom
    - Description
    - Nb. de kilomètre lors de l’achat
    - Photo
  + Modifier un véhicule existant
    - Nom
    - Description
    - Nb. de kilomètre lors de l’achat
    - Photo
  + Supprimer un véhicule existant
  + Choisir quel véhicule afficher
    - Son nom s’affiche
    - Sa description s’affiche
    - Son nombre de kilomètre lors de l’achat s’affiche
    - Son nombre de kilomètre actuelle s’affiche
    - Sa photo s’affiche
* Trajets
  + Ajouter un nouveau trajet
    - Départ
    - Arrivée
    - Distance parcourue (en km)
    - Date lors du trajet
  + Modifier un trajet existant
    - Départ
    - Arrivée
    - Distance
    - Date du trajet
  + Supprimer un trajet existant
* Entretiens/Maintenances
  + Cocher ses entretiens pour indiquer qu’ils ont été faits
  + Ajouter un nouvel entretien
    - Description
    - Le kilométrage auquel faudra effectuer le premier entretien en qu’on ajoute
    - La fréquence en kilomètre à laquelle faudra répéter l’entretien
  + Modifier un entretien, seulement ceux pas encore effectués
    - Description
    - Fréquence
  + Supprimer un entretien
* Points d’intérêts
  + Ajouter un point d’intérêt
    - Nom
    - Lieux déjà visité
    - Description
  + Visualiser sur la carte tous les points d’intérêts

## Présentation de l’interface

L’interface utilisateur comprend un total de sept formulaires. Une seule d’entre elle est la principale et c’est sur celle-ci qu’on voit toutes les informations. Les six autres sont pour modifier et ajouter de nouveaux éléments, comme les véhicules, les trajets et les entretiens.

### La fiche frmMain

Une image contenant capture d’écran

Description générée automatiquement

Cette fiche est la fiche principale de l’application. L’utilisateur a accès à tout à partir de cette fenêtre. Il peut sélectionner à l’aide du composant comboBox le véhicule sur lequel il compte faire des modifications. Il peut voir dans la section Véhicules se trouvant dans le premier tiers de la fenêtre, toutes les informations sur son véhicule. Les informations visibles du véhicule sont :

* Son nom, à l’aide d’un ComboBox qui permet la sélection d’un véhicule parmi ceux qu’on possède.
* Sa « Description » à l’aide d’un TextBox en lecture seule.
* Son « Nombre de kilomètres lors de l’achat » grâce à un TextBox en lecture seule.
* Puis finalement son « Nombre de kilomètres actuel » grâce à un TextBox en lecture seule. Cette valeur est calculé avec tous les trajets du véhicule et son kilométrage lors de l’achat.

La section véhicule est séparée du reste grâce à un composant GroupBox. L’utilisateur peut aussi dans cette section, voir les différentes actions qu’il peut effectuer sur son véhicule grâce aux différents boutons, comme par exemple le modifier, le supprimer ou encore ajouter un nouveau véhicule.

Dans la partie inférieure de cette fenêtre, l’utilisateur peut gérer plusieurs choses. Tel que les trajets qu’il a effectué avec le véhicule actuellement sélectionné, les entretiens qu’il a effectué sur son véhicule ou ceux qui sont à venir et les points d’intérêts de l’utilisateur, indépendant du véhicule sélectionné. Tout ceci est séparé à l’aide d’un TabControl qui est composé de différents onglets.

* **Onglet Saisie des trajets**

Une image contenant capture d’écran

Description générée automatiquement

Dans cet onglet apparaît la liste des trajets du véhicule sélectionné. Chaque trajet est affiché dans une ligne du composant DataGridView. La première colonne est le départ du trajet, ensuite son lieu d’arrivé, la distance parcourue entre le départ et l’arrivée, la date lorsque le trajet a été effectué, finalement un bouton bleu pour modifier le trajet et un rouge pour le supprimer. La liste des trajets est triée par leur date, le plus récent en dernier dans le tableau. Les colonnes n’ont pas toutes le même type de donnée :

* La colonne Départ
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial.
* La colonne Arrivée
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial.
* La colonne Distance
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial.
* La colonne Date
  + Est de type DataGridViewTextBoxColumn.
  + Avec un format date pour afficher la date d’une telle manière. Exemple : samedi, 23 mai 2020.
* La colonne Modifier
  + Est de type DataGridViewButtonColumn.
  + Sans format spécial.
* La colonne Supprimer
  + Est de type DataGridViewButtonColumn.
  + Sans format spécial.

Un composant de type Button se trouve en haut de l’onglet pour ajouter un nouveau trajet.

L’utilisateur peut à partir de cet onglet :

* Ajouter un trajet
* Modifier un trajet
* Supprimer un trajet
* **Onglet Carnet d’Entretiens**

Une image contenant capture d’écran

Description générée automatiquement

Dans cet onglet, l’utilisateur peut voir tous les entretiens effectués et ceux à venir. Les entretiens effectués sont affichés en vert, ceux qui doivent être effectués dans les 100 prochains kilomètres sont en jaune comme affiché sur l’image ci-dessus. Puis ceux en retard ou qui doivent être effectués absolument sont en rouge comme sur l’image si dessous.

Une image contenant capture d’écran

Description générée automatiquement

Cet onglet est composé d’un seul composant, un DataGridView. Il contient six différentes colonnes visibles, la première est une pour savoir si l’entretien a été effectué ou non, la suivante décris de quel entretien il s’agit, celle d’après indique la date de la dernière même maintenance s’il y en a une, la suivante indique le nombre de kilomètre de la moto lors de la même dernière maintenance, puis la prochaine indique la fréquence de la maintenance, finalement la dernière colonne montre dans combien de kilomètre il faudra effectuer la maintenance en question. Dans ce dataGridView il y a aussi une colonne cachée en première position pour enregistrer l’id de l’entretien pour savoir facilement quel entretien on coche comme étant effectué. Cette liste d’entretien est trié par le km de la prochaine maintenance, ceux étant fait se retrouvent plus bas dans la liste et le reste est trié en décroissant. Voici les différentes données en fonction des colonnes.

* La colonne Id
  + Est de type DataGridViewTextBoxColumn
  + Sans format spécial mais caché
* La colonne Effectué
  + Est de type DataGridViewCheckBoxColumn
  + Sans format spécial
* La colonne Description
  + Est de type DataGridViewTextBoxColumn
  + Sans format spécial
* La colonne Date Dernière Maintenance
  + Est de type DataGridViewTextBoxColumn
  + Avec un format date
* La colonne Km Dernière Maintenance
  + Est de type DataGridViewTextBoxColumn
  + Sans format spécial
* La colonne Fréquence
  + Est de type DataGridViewTextBoxColumn
  + Sans format spécial
* La colonne Prochaine Maintenance dans (km)
  + Est de type DataGridViewTextBoxColumn
  + Avec un format numérique

L’utilisateur peut à partir de cet onglet :

* Coché un trajet comme étant effectué (qui créera automatiquement le prochain entretien)
* **Onglet Gestion des Entretiens**

Une image contenant capture d’écran

Description générée automatiquement

Dans cet onglet, l’utilisateur peut ajouter, modifier ou supprimer un entretien. L’onglet contient deux composants, un de type Button pour ajouter un entretien et un DataGridView pour afficher la liste des entretiens et en fonction de la colonne il est possible d’en modifier ou supprimer un. Il faut savoir qu’il est possible de modifier seulement un entretien qui n’a pas encore été effectué.

Le DataGridView est trié dans l’ordre croissant du kilométrage du véhicule lors de l’entretien. Il est composé de six différentes colonnes visibles, la première est celle intitulé Effectué pour savoir si l’entretiens a déjà été effectué, la deuxième contient la description de l’entretien, la troisième indique le kilométrage lorsque l’entretien doit ou a été effectué, la quatrième montre la fréquence de l’entretien, la suivante contient un bouton bleu pour modifier l’entretien et la dernière un bouton rouge pour le supprimer. Voici les différentes données en fonction des colonnes :

* La colonne Id
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial mais caché.
* La colonne Effectué
  + Est de type DataGridViewCheckBoxColumn.
  + Sans format spécial.
* La colonne Description
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial.
* La colonne Km lors de l’entretien
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial.
* La colonne Fréquence
  + Est de type DataGridViewTextBoxColumn.
  + Sans format spécial.
* La colonne Modifier
  + Est de type DataGridViewButtonColumn.
  + Sans format spécial.
* La colonne Supprimer
  + Est de type DataGridViewButtonColumn.
  + Sans format spécial.

L’utilisateur peut à partir de cet onglet :

* Ajouter un entretien
* Modifier un entretien
* Supprimer un entretien
* **Onglet Points d’intérêts**

Une image contenant texte, carte

Description générée automatiquement

Dans cet onglet, l’utilisateur peut voir sur la carte tous les points d’intérêts qu’il a ajouté. Ces points d’intérêts peuvent être des lieux où l’utilisateur a aimé se rendre ou encore des lieux où il aimerait aller. Il peut toujours ajouter et voir ses points d’intérêts même s’il ne possède pas de véhicule car ils sont dépendant de l’utilisateur et non pas des véhicules.

Dans cet onglet il y a plusieurs contrôles, sur la gauche se trouve un composant GMapControl inséré à l’aide d’une librairie GMap.Net.Windows. Sur la droite se trouve les composant pour l’ajout d’un marqueur. Un TextBox pour le nom que l’utilisateur souhaite donner au marqueur, un ComboBox pour indiquer si oui on a déjà visité le lieu ou non, puis un autre TextBox multiligne pour laisser à l’utilisateur de la place pour écrire une description du lieu. Un composant de type Button pour que l’utilisateur puisse enregistrer ce qu’il a donné comme caractéristique au marqueur.

### La fiche frmAjoutVehicule

Une image contenant capture d’écran

Description générée automatiquement

Cette fiche s’ouvre lorsqu’on souhaite ajouter un véhicule dans la section supérieur de l’interface principale.

Les champs :

* Dans « Nom », le composant correspondant est une TextBox, l’utilisateur peut entrer n’importe quel nom tant qu’il sait reconnaitre le véhicule avec le nom entré.
* Dans « Description », le composant correspondant est une TextBox, l’utilisateur peut entrer ce qui lui plaît comme description de son véhicule. Comme par exemple le style, la puissance, la maniabilité, etc.
* Dans « Nb. de Km lors de l’achat », le composant correspondant est un NumericUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur doit simplement à combien de kilomètre était la moto lorsqu’il l’a achetée.
* Pour « Photo », l’utilisateur doit cliquer sur le bouton « Ajouter une photo », ce qui lui ouvrira l’explorateur de fichier en créant un nouvel objet OpenFileDialog, ce qui permet d’obtenir un fichier se trouvant sur l’ordinateur, dans ce cas-là une image. Il peut alors sélectionner une seule photo de son choix. Une fois l’image sélectionné, elle s’affiche dans le PictureBox en dessous du bouton « Ajouter une photo » ici non visible car il n’y a pas d’image.

Une fois tous les champs remplis seulement, l’utilisateur peut cliquer sur le bouton « Valider » pour ajouter son véhicule ou sur celui « Annuler » s’il souhaite annuler l’opération.

### La fiche frmModifierVehicule

Une image contenant capture d’écran

Description générée automatiquement

Cette fiche s’ouvre lorsqu’on souhaite modifier un véhicule dans la section supérieur de l’interface principale. Il faut pour cela avoir sélectionné un véhicule à partir du ComboBox se trouvant dans la même section.

Les champs sont les même que pour l’ajout d’un véhicule sauf que les champs sont préremplis avec les valeurs du véhicule que l’utilisateur est en train de modifier :

* Dans « Nom », le composant correspondant est une TextBox, l’utilisateur peut entrer n’importe quel nom tant qu’il sait reconnaitre le véhicule avec le nom entré.
* Dans « Description », le composant correspondant est une TextBox, l’utilisateur peut entrer ce qui lui plaît comme description de son véhicule. Comme par exemple le style, la puissance, la maniabilité, etc.
* Dans « Nb. de Km lors de l’achat », le composant correspondant est un NumericUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur doit simplement à combien de kilomètre était la moto lorsqu’il l’a achetée.
* Pour « Photo », l’utilisateur doit cliquer sur le bouton « Ajouter une photo », ce qui lui ouvrira l’explorateur de fichier en créant un nouvel objet OpenFileDialog, ce qui permet d’obtenir un fichier se trouvant sur l’ordinateur, dans ce cas-là une image. Il peut alors sélectionner une seule photo de son choix. Une fois l’image sélectionné, elle s’affiche dans le PictureBox en dessous du bouton « Ajouter une photo » ici non visible car il n’y a pas d’image.

Une fois tous les champs remplis seulement, l’utilisateur peut cliquer sur le bouton « Valider » pour ajouter son véhicule ou sur celui « Annuler » s’il souhaite annuler l’opération.

### La fiche frmAjouterTrajet

Pour ajouter un trajet il faut cliquer sur le bouton « Ajouter un nouveau trajet » se trouvant dans l’onglet « Saisie des trajets ».

Une fois cliqué sur le bouton cette fenêtre apparaît :

Une image contenant capture d’écran

Description générée automatiquement

Les champs :

* Dans « Départ », le composant correspondant est une TextBox, l’utilisateur peut entrer n’importe quel nom de lieu d’où il est parti.
* Dans « Arrivée », le composant correspondant est une TextBox, l’utilisateur peut entrer n’importe quel nom de lieu d’où il est arrivé.
* Dans « Distance parcourue », le composant correspondant est un NumericUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur doit entrer le nombre de kilomètre qu’il a parcouru lors de ce trajet.
* Pour « Date lors du trajet », le composant correspondant est un DateTimePicker qui sert à sélectionner une date, l’utilisateur doit entrer la date à laquelle le trajet à été effectué.

Une fois tous les champs remplis seulement, l’utilisateur peut cliquer sur le bouton « Valider » pour ajouter son trajet ou sur celui nommé « Annuler » s’il souhaite annuler l’opération. La distance du trajet sera alors ajoutée au kilométrage réel de la moto.

### La fiche frmModifierTrajet

Pour modifier un trajet il faut cliquer sur le bouton bleu se trouvant sur la même ligne que le trajet que l’on souhaite modifier. Une fois cliqué sur le bouton cette fenêtre doit apparaître :

Une image contenant capture d’écran

Description générée automatiquement

Les champs sont les même que pour l’ajout d’un trajet sauf qu’ils sont déjà préremplis avec les valeurs actuelles du trajet que l’utilisateur est en train de modifier :

* Dans « Départ », le composant correspondant est une TextBox, l’utilisateur peut entrer n’importe quel nom de lieu d’où il est parti.
* Dans « Arrivée », le composant correspondant est une TextBox, l’utilisateur peut entrer n’importe quel nom de lieu d’où il est arrivé.
* Dans « Distance parcourue », le composant correspondant est un NumericUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur doit entrer le nombre de kilomètre qu’il a parcouru lors de ce trajet.
* Pour « Date lors du trajet », le composant correspondant est un DateTimePicker qui sert à sélectionner une date, l’utilisateur doit entrer la date à laquelle le trajet à été effectué.

Une fois tous les champs remplis seulement, l’utilisateur peut cliquer sur le bouton « Valider » pour ajouter son trajet ou sur celui nommé « Annuler » s’il souhaite annuler l’opération. La distance du trajet sera alors ajoutée au kilométrage réel de la moto.

### La fiche frmAjoutEntretien

Pour ajouter un entretien il faut tout d’abord cliquer sur le bouton « Ajouter un nouvel Entretien » se trouvant dans l’onglet « Gestion des Entretiens », la fenêtre suivante doit ensuite s’afficher :

Une image contenant capture d’écran

Description générée automatiquement

Les champs :

* Dans « Description », le composant correspondant est une TextBox, l’utilisateur doit entrer quel est l’entretien qui va devoir être effectuer.
* Dans « Le premier entretien se fera à », le composant correspondant est un NumeriqueUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur doit indiquer à quel kilométrage de la moto le premier entretien aura lieu.
* Dans « La fréquence à laquelle effectuer l’entretien », le composant correspondant est un NumeriqueUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur doit entrer la fréquence en kilomètre à laquelle l’entretien devra être effectué à nouveau.

Une fois tous les champs remplis seulement, l’utilisateur peut cliquer sur le bouton « Valider » pour ajouter son entretien ou sur celui nommé « Annuler » s’il souhaite annuler l’opération.

### La fiche frmModifierEntretien

Pour modifier un entretien il faut cliquer sur le bouton bleu sur la même ligne que l’entretien que l’on souhaite modifier. Il est seulement possible de modifier un entretien non effectué. Une fois cliqué sur le bouton bleu d’un entretien pas encore effectué la fenêtre frmModifierEntretien apparaît :

Une image contenant capture d’écran

Description générée automatiquement

Les champs :

* Dans « Description », le composant correspondant est un TextBox, l’utilisateur peut modifier la description de l’entretien.
* Dans « La fréquence en km à laquelle effectuer l’entretien », le composant correspondant est un NumericUpDown qui permet d’entrer seulement des valeurs numériques, l’utilisateur peut modifier la fréquence en kilomètre à laquelle l’entretien devra être effectué à nouveau.

Une fois le ou les champs modifiés, l’utilisateur peut cliquer sur le bouton « Valider » pour valider les changements ou sur celui nommé « Annuler » pour annuler l’opération.

### Les labels informatifs

Une image contenant capture d’écran

Description générée automatiquement

Pour chaque formulaire de modification ou d’ajout. Que ce soit pour les véhicules, les trajets, les entretiens ou les points d’intérêts un message rouge apparaît dans un label si les champs ne sont pas renseignés.

### Les messages d’informations

#### Suppression d’un véhicule

Une image contenant capture d’écran

Description générée automatiquement

Ce message de confirmation de suppression de véhicule apparaît lorsque l’utilisateur clique sur le bouton pour supprimer un véhicule. Il a pour but d’éviter de faire une fausse manipulation.

#### Suppression d’un trajet

Une image contenant capture d’écran

Description générée automatiquement

Ce message de confirmation de suppression d’un trajet apparaît lorsque l’utilisateur clique sur un bouton rouge, servant à supprimer un trajet, se trouvant dans l’onglet des trajets. Ce message a pour but d’éviter de faire une fausse manipulation.

#### Suppression d’un entretien

Une image contenant capture d’écran

Description générée automatiquement

Ce message de confirmation de suppression d’un trajet apparaît lorsque l’utilisateur clique sur un bouton rouge, servant à supprimer un entretien, se trouvant dans l’onglet de gestion des trajets. Il sert à éviter les fausses manipulations.

#### Impossible de modifier entretien

Une image contenant capture d’écran

Description générée automatiquement

Ce message informatif apparaît lorsque l’utilisateur clique sur un bouton bleu pour modifier un entretien mais l’entretien a déjà été effectué, il est donc impossible de modifier cet entretien. Seul les entretiens pas encore effectués peuvent être modifiés.

#### Mauvais format de fichier

Une image contenant capture d’écran

Description générée automatiquement

Ce message apparaît lorsque l’utilisateur essaie d’ajouter un fichier autre que en .png ou .jpg lors de l’ajout ou modification d’un véhicule.

### Mesures de sécurité mises en place

Voici une liste des contrôles mis en place afin d’éviter les mauvaises manipulations des utilisateurs et ainsi empêcher un maximum de soulèvement d’exceptions dans le code :

* Les fenêtres ne sont pas redimensionnables afin d’éviter les problèmes d’affichage
* On ne peut pas maximiser l’affichage de la fenêtre principale pour de pas avoir de problème d’affichage
* Uniquement des chiffres entre 0 et 999999 sont possible d’entrer dans les NumericUpDown
* Si l’utilisateur n’entre aucune valeur dans un TextBox et que c’est nécessaire un message rouge dans un label apparaît et lui dit de bien remplir tous les champs.

# Généralités concernant l’implémentation

## Base de données

Pour stocker tous les véhicules, trajets, entretiens et points d’intérêts j’ai dû utiliser une base de données SQLite, comme mentionné dans le cahier des charges.

Voici le modèle conceptuel fourni dans le cahier des charges :

Une image contenant capture d’écran

Description générée automatiquement

J’ai donc créé cette base de données à l’aide de l’outil « DB Browser ». Pour la table pointsInterets il m’a fallu définir les différents champs. J’ai choisi d’ajouter les champs suivant :

* idPointInteret 🡪 pour pouvoir identifier chaque points d’intérêts
* lat 🡪 pour enregistrer la latitude du marqueur
* lng 🡪 pour enregistrer la longitude du marqueur
* nom 🡪 pour pouvoir donner un nom au marqueur. Par exemple un nom de lieu
* visite 🡪 pour indiquer lors de l’ajout si le lieu a déjà été visité ou non
* description 🡪 pour donner une description au marqueur

Voici la table :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **pointInteret** | | | | |
| **Nom** | **Type** | **Clef primaire** | **Auto-Incrément** | **Non Null** |
| idPointInteret | INTEGER | TRUE | TRUE | FALSE |
| lat | REAL | FALSE | FALSE | TRUE |
| lng | REAL | FALSE | FALSE | TRUE |
| nom | TEXT | FALSE | FALSE | FALSE |
| visite | TEXT | FALSE | FALSE | FALSE |
| description | TEXT | FALSE | FALSE | FALSE |

Pour la table maintenance j’ai décidé d’ajouter le champ « kmPremiereMaintenance » pour être capable de différencier les tout premiers entretiens, lorsqu’on les crée, de ceux qui se répètent simplement. Cela permet lors de l’ajout d’un entretien, de définir à combien de kilomètre veut-on effectuer la première maintenance et avec quelle fréquence en kilomètre. Aussi pour le champ « fait », dans DB Browser il n’y a pas de type booléen donc j’ai fais avec du INTEGER où j’entre 0 ou 1.

Voici ce que cela donne :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **maintenance** | | | | |
| **Nom** | **Type** | **Clef primaire** | **Auto-Incrément** | **Non Null** |
| idMaintenance | INTEGER | TRUE | TRUE | FALSE |
| description | TEXT | FALSE | FALSE | TRUE |
| freqKm | INTEGER | TRUE | FALSE | FALSE |
| kmPremiereMaintenance | INTEGER | FALSE | FALSE | FALSE |
| kmDerniereMaintenance | INTEGER | FALSE | FALSE | FALSE |
| dateDerniereMaintenance | TEXT | FALSE | FALSE | FALSE |
| fait | INTEGER | FALSE | FALSE | FALSE |
| idVehicule | INTEGER | FALSE | FALSE | FALSE |

Pour le reste je n’ai rien changé.

Les photos sont stocké en type BLOB, il s’agit de stocker des données binaires. Ca a l’avantage de pouvoir stocker n’importe quoi comme donnée mais ça prend du temps à les récupérer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **vehicule** | | | | |
| **Nom** | **Type** | **Clef primaire** | **Auto-Incrément** | **Non Null** |
| idVehicule | INTEGER | TRUE | TRUE | FALSE |
| nom | TEXT | FALSE | FALSE | TRUE |
| description | TEXT | FALSE | FALSE | FALSE |
| kmInitial | INTEGER | FALSE | FALSE | FALSE |
| kmReel | INTEGER | FALSE | FALSE | FALSE |
| photo | BLOB | FALSE | FALSE | FALSE |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **trajet** | | | | |
| **Nom** | **Type** | **Clef primaire** | **Auto-Incrément** | **Non Null** |
| idTrajet | INTEGER | TRUE | TRUE | FALSE |
| depart | TEXT | FALSE | FALSE | FALSE |
| arrivee | TEXT | FALSE | FALSE | FALSE |
| distance | INTEGER | FALSE | FALSE | TRUE |
| date | TEXT | FALSE | FALSE | FALSE |
| idVehicule | INTEGER | FALSE | FALSE | FALSE |

## Classes

Pour pouvoir réaliser le projet correctement j’ai divisé le projet en plusieurs classes.

* Vehicule
* Trajet
* Entretien
* PointInteret