Documentation de l'Application Gestionnaire de Voitures

Table des matières

Introduction

L'application "Gestionnaire de Voitures" est une interface graphique développée en Python permettant aux utilisateurs de gérer une collection de voitures et leurs modifications. Elle utilise la bibliothèque CustomTkinter pour l'interface graphique et permet de stocker les données dans un fichier JSON local.

L'application permet de:

- Ajouter des voitures à une base de données
- Visualiser la liste des voitures enregistrées
- Consulter les détails d'une voiture sélectionnée
- Ajouter des modifications à une voiture
- Calculer le prix total d'une voiture avec ses modifications
- Supprimer des voitures de la base de données

Architecture de l'application

L'application suit une architecture à plusieurs couches:

Couche de données

- **Modèles** (`Voiture`, `Modification`): Définissent la structure des données
- **Gestionnaire** (`GestionnaireVoitures`): Gère la persistance et les opérations sur les données

Couche interface

- **Application principale** (`ApplicationVoiture`): Fenêtre principale et gestion de navigation
- **Frames** (`AccueilFrame`, `AjouterVoitureFrame`, `AfficherVoituresFrame`): Écrans de l'application

Persistance

Les données sont sauvegardées dans un fichier JSON (`voitures.json`).

Modèles de données

Classe `Modification`

Représente une modification appliquée à une voiture.

Attributs:

- `description` (str): Description de la modification
- `cout` (float): Coût de la modification en euros

Méthodes:

- `to_dict()`: Convertit l'objet en dictionnaire pour la sérialisation
- `from_dict(data)`: Crée un objet Modification à partir d'un dictionnaire (méthode statique)

Classe `Voiture`

Représente un véhicule avec ses caractéristiques et modifications.

Attributs:

- `id_voiture` (str): Identifiant unique (UUID)
- `nom` (str): Nom du véhicule
- `marque` (str): Marque du véhicule
- `prix_achat` (float): Prix d'achat en euros
- `modifications` (list): Liste des objets Modification

Méthodes:

- `ajouter_modification(description, cout)`: Ajoute une modification à la voiture
- `calculer_prix_total()`: Calcule le prix total (achat + modifications)
- `to_dict()`: Convertit l'objet en dictionnaire pour la sérialisation
- `from_dict(data)`: Crée un objet Voiture à partir d'un dictionnaire (méthode statique)

Classe `GestionnaireVoitures`

Gère la collection de voitures et la persistance des données.

Attributs:

- `voitures` (list): Liste des objets Voiture
- `fichier_donnees` (str): Chemin du fichier de sauvegarde JSON

Méthodes:

- `ajouter_voiture(voiture)`: Ajoute une voiture à la collection
- `sauvegarder_donnees()`: Sauvegarde les données dans le fichier JSON
- `charger_donnees()`: Charge les données depuis le fichier JSON
- `supprimer_voiture_par_id(id_voiture)`: Supprime une voiture par son identifiant

Interface utilisateur

L'application utilise CustomTkinter pour créer une interface moderne avec un thème sombre. Les éléments d'interface suivent une charte graphique cohérente définie par les constantes globales:

- `THEME_COLOR`: Couleur de fond (#1b263b)
- `BUTTON_COLOR`: Couleur des boutons (#0d1b2a)
- `TEXT_COLOR`: Couleur du texte (#ffffff)
- `ACCENT_COLOR`: Couleur d'accent (#81A1C1)
- `FONT`: Police par défaut (Arial, 12)

Des icônes Unicode sont utilisées pour améliorer l'interface:

- `ICON_HOME`: (Accueil)
- `ICON_ADD`: (Ajouter)
- `ICON_LIST`: (Liste)
- `ICON_REFRESH`: (Actualiser)
- `ICON_EDIT`: ✏ (Modifier)
- `ICON_DELETE`: (Supprimer)
- `ICON_SAVE`: (Sauvegarder)

Architecture des écrans

L'application est organisée en trois écrans principaux:

1. **AccueilFrame**: Écran d'accueil affichant le titre de l'application
2. **AjouterVoitureFrame**: Formulaire pour ajouter une nouvelle voiture
3. **AfficherVoituresFrame**: Affiche la liste des voitures et leurs détails

La navigation entre les écrans est gérée par une barre de navigation en haut de l'application.

Fonctionnalités

Ajout de voiture

Permet d'ajouter une nouvelle voiture avec:

- Nom
- Marque
- Prix d'achat

Consultation des voitures

Affiche une liste des voitures enregistrées. En cliquant sur une voiture, ses détails sont affichés:

- Informations de base (nom, marque, prix d'achat)
- Prix total avec modifications
- Liste des modifications avec leurs coûts

Ajout de modification

Pour une voiture sélectionnée, permet d'ajouter une modification:

- Description de la modification
- Coût de la modification

Suppression de voiture

Permet de supprimer une voiture sélectionnée après confirmation.

Persistance des données

Toutes les données sont automatiquement sauvegardées dans un fichier JSON et rechargées au démarrage de l'application.

Guide technique

Dépendances

- Python 3.x
- customtkinter
- tkinter (inclus dans Python)
- json (module standard)

- uuid (module standard)

Structure des fichiers

- `voitures.json`: Fichier de stockage des données
- Script principal contenant tout le code de l'application

Format des données JSON

Le fichier `voitures.json` contient un tableau d'objets voiture:

```
[
  {
    "id_voiture": "uuid-string",
    "nom": "Nom de la voiture",
    "marque": "Marque",
    "prix_achat": 15000.0,
    "modifications": [
      {
        "description": "Description de la modification",
        "cout": 500.0
      }
    ]
  }
]
```

Flux de travail typiques

Ajout d'une nouvelle voiture

1. Cliquer sur "□ Ajouter Voiture" dans la barre de navigation
2. Remplir les champs du formulaire (nom, marque, prix d'achat)
3. Cliquer sur "□ Enregistrer la Voiture"

Ajout d'une modification à une voiture

1. Cliquer sur "□ Liste Voitures" dans la barre de navigation
2. Sélectionner une voiture dans la liste en cliquant dessus
3. Cliquer sur "✏ Ajouter Modification"
4. Remplir les champs de la fenêtre popup (description, coût)
5. Cliquer sur "□ Valider"

Suppression d'une voiture

1. Cliquer sur "□ Liste Voitures" dans la barre de navigation
2. Sélectionner une voiture dans la liste en cliquant dessus
3. Cliquer sur "□ Supprimer"
4. Confirmer la suppression dans la boîte de dialogue