# Récapitulatif de l'Application "ProjetCoursesVoitures"

# Campan Romain L3 NEC UPPA 2025

## Aperçu Général

L'application `ProjetCoursesVoitures` est une application iOS conçue pour permettre aux utilisateurs de gérer des trajets de courses de voitures. Elle offre des fonctionnalités pour créer, modifier, et visualiser des trajets, ainsi que pour gérer les paramètres utilisateur et les options de l'application. L'application utilise Core Data pour la gestion des données, Core Location pour la localisation, et MapKit pour l'affichage des cartes.

## Fonctionnalités Principales

1. **Gestion des Trajets**
    - **Création de Trajets** : Les utilisateurs peuvent créer de nouveaux trajets en ajoutant des checkpoints avec des coordonnées spécifiques.
    - **Modification de Trajets** : Les utilisateurs peuvent modifier les trajets existants en ajoutant, supprimant, ou modifiant des checkpoints.
    - **Visualisation des Trajets** : Les utilisateurs peuvent visualiser les trajets sur une carte, avec des annotations pour les checkpoints et des routes calculées entre les points.
2. **Gestion des Utilisateurs**
    - **Ajout d'Utilisateurs** : Les utilisateurs peuvent ajouter de nouveaux utilisateurs avec des informations personnelles et des informations sur leur voiture.
    - **Chargement d'Utilisateurs** : Les utilisateurs peuvent charger des utilisateurs existants en utilisant leur pseudo et mot de passe.
    - **Déconnexion** : Les utilisateurs peuvent se déconnecter de leur compte.
3. **Paramètres de l'Application**
    - **Thème** : Les utilisateurs peuvent basculer entre un thème clair et un thème sombre.
    - **Notifications** : Les utilisateurs peuvent activer ou désactiver les notifications.
    - **Services de Localisation** : Les utilisateurs peuvent activer ou désactiver les services de localisation.
    - **Mode d'Économie de Données** : Les utilisateurs peuvent activer ou désactiver le mode d'économie de données.
    - **Utilisation des Services Web** : Les utilisateurs peuvent activer ou désactiver l'utilisation des services web pour charger des trajets.
4. **Localisation et Simulation**
    - **Localisation en Temps Réel** : L'application peut suivre la position de l'utilisateur en temps réel et afficher cette position sur la carte.

- **Simulation de Trajets** : Les utilisateurs peuvent lancer une simulation de trajet, qui affiche les checkpoints et calcule les routes entre les points.

## Architecture et Technologies

5. **Core Data**
    - Utilisé pour la gestion persistante des données, y compris les utilisateurs, les trajets, et les checkpoints.
    - Les entités principales incluent `UtilisateurMO`, `TrajetMO`, et `CheckPointMO`.
6. **Core Location**
    - Utilisé pour obtenir la position actuelle de l'utilisateur et suivre les mouvements en temps réel.
7. **MapKit**
    - Utilisé pour afficher les cartes, ajouter des annotations pour les checkpoints, et dessiner les routes entre les points.
8. **UserNotifications**
    - Utilisé pour envoyer des notifications à l'utilisateur, par exemple pour indiquer la distance jusqu'au prochain checkpoint.

## Classes et Fichiers Principaux

9. **AppDelegate.swift**
    - Gère l'initialisation de l'application, la configuration des options, et la gestion des utilisateurs et des trajets.
10. **ajouterModifierSupprParcoursTVC.swift**
    - Gère l'affichage et la modification des trajets dans une table view.
11. **carteVC.swift**
    - Gère l'affichage de la carte, la localisation en temps réel, et les interactions avec la carte.
12. **choixTrajet.swift**
    - Permet aux utilisateurs de sélectionner un trajet parmi une liste.
13. **CreerTrajet.swift**
    - Permet aux utilisateurs de créer de nouveaux trajets en ajoutant des checkpoints.
14. **debugVC.swift**
    - Fournit une interface de débogage pour tester les fonctionnalités de la carte et de la localisation.
15. **ModifTrajet.swift**
    - Permet aux utilisateurs de modifier les trajets existants.

16. **Outils.swift**
    - Fournit des méthodes utilitaires pour interagir avec Core Data, trier des données, et gérer des opérations courantes.
17. **principaleVC.swift**
    - Gère la vue principale de l'application, la localisation en temps réel, et la simulation des trajets.
18. **settingsVC.swift**
    - Permet aux utilisateurs de configurer les paramètres de l'application.
19. **userSettingsVC.swift**
    - Gère les paramètres utilisateur, y compris l'ajout, le chargement, et la déconnexion des utilisateurs.

# AppDelegate.swift

## Aperçu

Le fichier `AppDelegate.swift` est responsable de la gestion globale de l'application, y compris l'initialisation, la gestion des utilisateurs, la localisation, les notifications, et la configuration de Core Data. La classe principale, `AppDelegate` implémente les protocoles `UIApplicationDelegate`, `CLLocationManagerDelegate`, et `UNUserNotificationCenterDelegate`.

## Propriétés Statiques

Plusieurs propriétés statiques sont définies pour gérer les états globaux de l'application :

- `currentUser` : L'utilisateur actuellement connecté.
- `userListeTrajets` : La liste des trajets de l'utilisateur actuel.
- `allUsersTrajets` : La liste de tous les trajets de tous les utilisateurs.
- `currentTrajet` : Le trajet actuel.
- `allUsers` : La liste de tous les utilisateurs.
- `nextCheckpoint` : Le prochain point de contrôle.
- `lesOptions` : Les options de l'application.
- `couleurVoiture` : La couleur de la voiture de l'utilisateur actuel.

## Méthodes Principales

1. **`application(_:didFinishLaunchingWithOptions:)`**
    - Initialise l'application.
    - Charge les options système et les applique.
    - Configure les services de localisation si activés.

    - Charge la liste des utilisateurs et définit un utilisateur par défaut.
    - Configure les notifications si activées.
2. `setCurrentUser()`
    - Définit l'utilisateur actuel et charge ses trajets.
3. `logoutCurrentUser()`
    - Déconnecte l'utilisateur actuel et réinitialise sa liste de trajets.
4. `application(_:configurationForConnecting:options:)`
    - Configure la scène de connexion.
5. `application(_:didDiscardSceneSessions:)`
    - Appelée lorsque l'utilisateur rejette une session de scène.
6. `persistentContainer`
    - Configure le conteneur persistant pour Core Data.
7. `saveContext()`
    - Sauvegarde le contexte Core Data si des modifications ont été apportées.
8. `changeTheme(to:)`
    - Change le thème de l'application entre clair et sombre.
9. `loadAndApplyOptions(_:)`
    - Charge et applique les options de l'application, y compris le thème.

## ajouterModifierSupprParcoursTVC.swift

### Aperçu

Le fichier `ajouterModifierSupprParcoursTVC.swift` est responsable de la gestion des trajets utilisateur. La classe principale, `ajouterModifierSupprParcoursTVC`, hérite de `UITableViewController` et implémente le protocole `CLLocationManagerDelegate`. Elle gère l'affichage, l'ajout, la modification et la suppression des trajets utilisateur.

### Protocoles

Deux protocoles sont définis pour assurer le bon fonctionnement du rechargement de la table après l'ajout ou la modification d'un trajet :

- `TrajetCreationDelegate` : Contient la méthode `ajouteTrajet()`.
- `TrajetModificationDelegate` : Contient la méthode `modifTrajet()`.

### Méthodes Principales

1. `viewDidLoad()`

- Initialise la vue.

2. **Configuration de la Table**
   - `numberOfSections(in:)` : Retourne 1, indiquant qu'il y a une seule section dans la table.
   - `tableView(_:numberOfRowsInSection:)` : Retourne le nombre de trajets dans `AppDelegate.userListeTrajets`.
   - `tableView(_:cellForRowAt:)` : Configure les cellules de la table avec les informations des trajets.
3. **Suppression de Ligne**
   - `tableView(_:commit:forRowAt:)` : Permet la suppression d'une ligne en swipant vers la gauche. La ligne est supprimée du contexte et de la table view.
4. **Préparation des Segues**
   - `prepare(for:sender:)` : Prépare les segues pour la sélection et la création de trajets. Les délégués sont assignés pour les vues de modification et de création de trajets.
5. **Action pour Ajouter un Trajet**
   - `tapSurAjouterTrajet(_:)` : Effectue une segue pour ajouter un nouveau trajet.

## Extensions

Deux extensions sont utilisées pour implémenter les protocoles de délégation :

6. `TrajetCreationDelegate`
   - `ajouteTrajet()` : Recharge les données de la table après l'ajout d'un trajet.
7. `TrajetModificationDelegate`
   - `modifTrajet()` : Recharge uniquement la ligne modifiée après la modification d'un trajet.

# carteVC.swift

## Aperçu

Le fichier `carteVC.swift` est responsable de la gestion de la carte en plein écran. La classe principale, `carteVC`, hérite de `UIViewController` et implémente les protocoles `CLLocationManagerDelegate` et `MKMapViewDelegate`. Elle gère l'affichage des checkpoints, la localisation en temps réel, et le calcul des routes sur une carte.

## Propriétés

- `carte` : Une vue de carte de type `MKMapView`.
- `tabCheckpoints` : Un tableau de checkpoints de type `CheckPointMO`.
- `userAnnotation` : Une annotation pour la position de l'utilisateur.

- `locationManager` : Un gestionnaire de localisation de type `CLLocationManager`.

## Méthodes Principales

1. **`viewDidLoad()`**
    - Initialise la vue.
    - Configure la carte et ajoute les annotations pour les checkpoints.
    - Configure le gestionnaire de localisation et demande les autorisations nécessaires.
    - Ajoute un gestionnaire de gestes pour les appuis longs.
2. **`handleLongPress(_:)`**
    - Gère les appuis longs sur la carte pour ajouter une nouvelle annotation à l'endroit de l'appui.
3. **`calculerRoute(from:to:)`**
    - Calcule une route entre deux points de coordonnées et l'affiche sur la carte.
4. **`locationManager(_:didUpdateLocations:)`**
    - Met à jour la position de l'utilisateur en temps réel et affiche les checkpoints et les routes associées au trajet actuel.

## Extensions

5. **`MKMapViewDelegate`**
    - **`mapView(_:rendererFor:)`** : Dessine les routes sur la carte.
    - **`mapView(_:viewFor:)`** : Configure les vues pour les annotations, y compris l'annotation de la position de l'utilisateur et les checkpoints.

# choixTrajet.swift

## Aperçu

Le fichier `choixTrajet.swift` est responsable de la gestion de la sélection des trajets par l'utilisateur. La classe principale, `choixTrajet`, hérite de `UIViewController` et implémente les protocoles `UITableViewDataSource` et `UITableViewDelegate`. Elle gère l'affichage des trajets disponibles dans une table view et permet à l'utilisateur de sélectionner un trajet.

## Propriétés

- `maTable` : Une table view de type `UITableView`.
- `delegate` : Un délégué de type `SelectionTrajet` pour notifier la sélection d'un trajet.

## Méthodes Principales

1. **`viewDidLoad()`**
    - Initialise la vue.

- Configure la table view en définissant ses délégués (`dataSource` et `delegate`).

2. **Configuration de la Table View**
   - `numberOfSections(in:)` : Retourne 1, indiquant qu'il y a une seule section dans la table.
   - `tableView(_:numberOfRowsInSection:)` : Retourne le nombre de trajets dans `AppDelegate.userListeTrajets`.
   - `tableView(_:cellForRowAt:)` : Configure les cellules de la table avec les informations des trajets.
3. **Sélection d'un Trajet**
   - `tableView(_:didSelectRowAt:)` : Met à jour le trajet en cours (`AppDelegate.currentTrajet`) avec le trajet sélectionné, notifie le délégué, et ferme la vue.

## CreerTrajet.swift

### Aperçu

Le fichier `CreerTrajet.swift` est responsable de la création et de la gestion des trajets par l'utilisateur. La classe principale, `CreerTrajet`, hérite de `UIViewController` et implémente les protocoles `UITableViewDataSource` et `UITableViewDelegate`. Elle permet à l'utilisateur de créer de nouveaux trajets, d'ajouter des checkpoints, et de charger des trajets depuis un web service.

### Structures de Données

- `Trajet` : Une structure codable représentant un trajet avec un nom, un label, et une liste de checkpoints.
- `Checkpoint` : Une structure codable représentant un checkpoint avec un identifiant, un label, et des coordonnées x et y.

### Propriétés

- `urlTextField` : Un champ de texte pour entrer l'URL du web service.
- `checkpoints` : Un tableau de checkpoints.
- `delegate` : Un délégué de type `TrajetCreationDelegate` pour notifier la création d'un trajet.
- `nomTF`, `labelTF`, `maTable`, `currentCPLab`, `currentCPX`, `currentCPY` : Des champs de texte et une table view pour l'interface utilisateur.

### Méthodes Principales

1. `viewDidLoad()`
   - Initialise la vue.
   - Configure la table view en définissant ses délégués (`dataSource` et `delegate`).
2. **Configuration de la Table View**

- `numberOfSections(in:)` : Retourne 1, indiquant qu'il y a une seule section dans la table.
- `tableView(_:numberOfRowsInSection:)` : Retourne le nombre de checkpoints dans `checkpoints`.
- `tableView(_:cellForRowAt:)` : Configure les cellules de la table avec les informations des checkpoints.

3. **Suppression de Ligne**
   - `tableView(_:commit:forRowAt:)` : Permet la suppression d'une ligne en swipant vers la gauche. La ligne est supprimée du tableau `checkpoints` et de la table view.
4. **Ajout de Checkpoint**
   - `addRow(_:)` : Ajoute un nouveau checkpoint à la liste `checkpoints` et met à jour la table view.
5. **Sauvegarde du Trajet**
   - `tapSurSauver(_:)` : Crée un nouveau trajet avec les checkpoints ajoutés, le sauvegarde dans Core Data, et notifie le délégué.
6. **Chargement depuis le Web Service**
   - `chargerTrajetDepuisWebService(_:)` : Charge un trajet depuis une URL spécifiée, le sauvegarde dans Core Data, et notifie le délégué.

## debugVC.swift

### Aperçu

Le fichier `debugVC.swift` est responsable de la gestion de la carte pour le débogage. La classe principale, `debugVC`, hérite de `UIViewController` et implémente les protocoles `CLLocationManagerDelegate` et `MKMapViewDelegate`. Elle permet à l'utilisateur de visualiser sa position sur une carte, d'ajouter des annotations, de calculer des routes, et de déplacer sa position de manière simulée.

### Propriétés

- `carte` : Une vue de carte de type `MKMapView`.
- `locationManager` : Un gestionnaire de localisation de type `CLLocationManager`.
- `userLocation` : Les coordonnées de la position de l'utilisateur.
- `userAnnotation` : Une annotation pour la position de l'utilisateur.

### Méthodes Principales

1. `viewDidLoad()`
   - Initialise la vue.
   - Configure la carte et le gestionnaire de localisation.

- Ajoute un gestionnaire de gestes pour les appuis longs.

2. **Ajout de l'Annotation de l'Utilisateur**
   - `addUserAnnotation()` : Ajoute ou met à jour l'annotation de la position de l'utilisateur sur la carte.
3. **Déplacement de la Position de l'Utilisateur**
   - `moveUserLocation(latitudeDelta:longitudeDelta:)` : Déplace la position de l'utilisateur de manière simulée en ajustant les coordonnées.
   - `deplacerLocalisationHaut(_:)`, `deplacerLocalisationBas(_:)`, `deplacerLocalisationGauche(_:)`, `deplacerLocalisationDroite(_:)` : Méthodes pour déplacer la position de l'utilisateur dans différentes directions.
4. **Gestion des Appuis Longs**
   - `handleLongPress(_:)` : Ajoute une annotation à l'endroit de l'appui long sur la carte.
5. **Calcul de Route**
   - `calculerRoute(from:to:)` : Calcule une route entre deux points de coordonnées et l'affiche sur la carte.
6. **Mise à Jour de la Localisation**
   - `locationManager(_:didUpdateLocations:)` : Met à jour la position de l'utilisateur en temps réel et ajoute l'annotation correspondante sur la carte.

## Extensions

7. `MKMapViewDelegate`
   - `mapView(_:rendererFor:)` : Dessine les routes sur la carte.
   - `mapView(_:didSelect:)` : Calcule une route entre la position de l'utilisateur et l'annotation sélectionnée.

# ModifTrajet.swift

## Aperçu

Le fichier `ModifTrajet.swift` est responsable de la modification des trajets par l'utilisateur. La classe principale, `ModifTrajet`, hérite de `UIViewController` et implémente les protocoles `UITableViewDataSource` et `UITableViewDelegate`. Elle permet à l'utilisateur de modifier les informations d'un trajet existant, d'ajouter, de modifier et de supprimer des checkpoints, et de sauvegarder les modifications.

## Propriétés

- `delegate` : Un délégué de type `TrajetModificationDelegate` pour notifier la modification d'un trajet.
- `trajet` : Le trajet à modifier de type `TrajetMO`.

- `checkpoints` : Un tableau de checkpoints.
- `maTable`, `nomTF`, `labelTF`, `currentCPLab`, `currentCPX`, `currentCPY` : Des champs de texte et une table view pour l'interface utilisateur.

## Méthodes Principales

1. `viewDidLoad()`
    - Initialise la vue.
    - Configure la table view en définissant ses délégués (`dataSource` et `delegate`).
    - Charge le trajet importé par la segue et initialise les champs de texte et les checkpoints.
2. **Configuration de la Table View**
    - `numberOfSections(in:)` : Retourne 1, indiquant qu'il y a une seule section dans la table.
    - `tableView(_:numberOfRowsInSection:)` : Retourne le nombre de checkpoints dans `checkpoints`.
    - `tableView(_:cellForRowAt:)` : Configure les cellules de la table avec les informations des checkpoints.
3. **Sélection d'un Checkpoint**
    - `tableView(_:didSelectRowAt:)` : Affiche les informations du checkpoint sélectionné dans les champs de texte.
4. **Suppression de Ligne**
    - `tableView(_:commit:forRowAt:)` : Permet la suppression d'une ligne en swipant vers la gauche. La ligne est supprimée du tableau `checkpoints` et de la table view.
5. **Ajout de Checkpoint**
    - `addRow(_:)` : Ajoute un nouveau checkpoint à la liste `checkpoints` et met à jour la table view.
6. **Modification d'une Ligne**
    - `modifierUnLigne(_:)` : Modifie les informations d'un checkpoint sélectionné et met à jour la table view.
7. **Sauvegarde du Trajet**
    - `tapSurSauver(_:)` : Met à jour les informations du trajet et des checkpoints, les sauvegarde dans Core Data, et notifie le délégué.

# Outils.swift

## Aperçu

Le fichier `Outils.swift` contient une classe utilitaire `Outils` qui fournit diverses méthodes pour interagir avec Core Data, trier des données, et gérer des opérations courantes dans l'application. Cette classe est essentielle pour centraliser les fonctions réutilisables et maintenir une architecture propre.

## Méthodes Principales

8. `donneContexte()`
    - **Description** : Retourne le contexte du magasin persistant de Core Data à partir de l'objet `AppDelegate`.
    - **Utilisation** : Utilisée pour obtenir le contexte nécessaire pour effectuer des opérations de Core Data.
9. `quickSort(_:)`
    - **Description** : Implémente l'algorithme de tri rapide (QuickSort) pour trier un tableau de checkpoints par leur identifiant.
    - **Utilisation** : Utilisée pour trier les checkpoints de manière efficace.
10. `donneCouleurVoiture(_:)`
    - **Description** : Retourne une couleur `UIColor` en fonction d'une chaîne de caractères représentant la couleur.
    - **Utilisation** : Utilisée pour définir la couleur de la voiture sur la carte.
11. `donneTousLesUtilisateurs()`
    - **Description** : Récupère tous les utilisateurs enregistrés dans Core Data.
    - **Utilisation** : Utilisée pour obtenir la liste complète des utilisateurs.
12. `donneLUtilisateur(_:)`
    - **Description** : Récupère un utilisateur spécifique en fonction de son pseudo.
    - **Utilisation** : Utilisée pour obtenir un utilisateur particulier à partir de son pseudo.
13. `donneCheckpointsTriesParId(pourTrajet:)`
    - **Description** : Récupère les checkpoints d'un trajet donné et les trie par leur identifiant.
    - **Utilisation** : Utilisée pour obtenir une liste triée de checkpoints pour un trajet spécifique.
14. `donneTousLesTrajetsDUnUtilisateur(_:)`
    - **Description** : Récupère tous les trajets d'un utilisateur donné et trie les checkpoints de chaque trajet par leur identifiant.
    - **Utilisation** : Utilisée pour obtenir tous les trajets d'un utilisateur et s'assurer que les checkpoints sont triés.

# principaleVC.swift

## Aperçu

Le fichier `principaleVC.swift` est responsable de la gestion de la vue principale de l'application. La classe principale, `principaleVC`, hérite de `UIViewController` et implémente les protocoles `CLLocationManagerDelegate` et `SelectionTrajet`. Elle gère l'affichage de la carte, la localisation en temps réel, la simulation des trajets, et les notifications.

## Propriétés

- `message`, `carte`, `nbCheckpointsLab`, `nbTotalCheckpoints`, `distanceProchainCheckpointLab`, `vitesseUtilisateurLab` : Des labels et une carte pour l'interface utilisateur.
- `locationManager` : Un gestionnaire de localisation de type `CLLocationManager`.
- `checkpoints` : Un tableau de checkpoints de type `CheckPointMO`.
- `currentCheckpointIndex` : L'index du checkpoint actuel.
- `userAnnotation` : Une annotation pour la position de l'utilisateur.

## Méthodes Principales

1. **`viewDidLoad()`**
    - Initialise la vue.
    - Configure la carte et le gestionnaire de localisation.
    - Ajoute un gestionnaire de gestes pour les appuis longs.
    - Gère le thème au démarrage.
2. **Gestion des Appuis Longs**
    - **`handleLongPress(_:)`** : Affiche les coordonnées de l'endroit où l'utilisateur appuie longtemps sur la carte.
3. **Lancer la Simulation**
    - **`lancerSimulation(_:)`** : Lance la simulation du trajet sélectionné, affiche les checkpoints et calcule les routes.
4. **Calcul de Route**
    - **`calculerRoute(from:to:)`** : Calcule une route entre deux points de coordonnées et l'affiche sur la carte.
5. **Localisation en Temps Réel**
    - **`locationManager(_:didUpdateLocations:)`** : Met à jour la position de l'utilisateur en temps réel, affiche les informations sur la vitesse et la distance jusqu'au prochain checkpoint, et met à jour les notifications.
6. **Gestion des Segues**

- `prepare(for:sender:)` : Prépare les segues pour la sélection des trajets.

7. **Notifications**
    - `updateDistanceNotification()` : Met à jour les notifications de distance jusqu'au prochain checkpoint.
    - `calculateDistance(from:to:)` : Calcule la distance entre la position de l'utilisateur et un checkpoint.
    - `notifieUtilisateur(distance:)` : Crée et envoie une notification à l'utilisateur.
    - `userNotificationCenter(_:didReceive:withCompletionHandler:)` : Gère les actions de l'utilisateur sur les notifications.

## Extensions

8. `SelectionTrajet`
    - `pickTrajet()` : Met à jour les informations affichées lorsqu'un trajet est sélectionné.
9. `MKMapViewDelegate`
    - `mapView(_:rendererFor:)` : Dessine les routes sur la carte.
    - `mapView(_:viewFor:)` : Configure les vues pour les annotations, y compris l'annotation de la position de l'utilisateur et les checkpoints.

# settingsVC.swift

## Aperçu

Le fichier `settingsVC.swift` est responsable de la gestion des paramètres de l'application. La classe principale, `settingsVC`, hérite de `UIViewController` et permet à l'utilisateur de configurer diverses options telles que le thème, les notifications, les services de localisation, le mode d'économie de données, et l'utilisation des services web.

## Structures et Classes

10. `AppOptions`
    - **Description** : Une structure codable représentant les options de l'application.
    - **Propriétés** :
        - `theme` : Le thème de l'application (clair ou sombre).
        - `notificationsEnabled` : Indique si les notifications sont activées.
        - `locationServicesEnabled` : Indique si les services de localisation sont activés.
        - `dataSavingModeEnabled` : Indique si le mode d'économie de données est activé.
        - `useWebService` : Indique si les services web sont utilisés.
11. `OptionsManager`

- **Description** : Une classe singleton pour gérer la lecture et l'écriture des options de l'application dans un fichier JSON.
- **Méthodes** :
  - `lireLesOptions()` : Lit les options à partir d'un fichier JSON.
  - `ecrireLesOptions(_:)` : Écrit les options dans un fichier JSON.
  - `getDocumentsDirectory()` : Retourne l'URL du répertoire des documents de l'application.

12. `settingsVC`
    - **Description** : Un contrôleur de vue pour gérer les paramètres de l'application.
    - **Propriétés** :
      - `themeSwitch`, `notificationsSwitch`, `locationServicesSwitch`, `dataSavingModeSwitch`, `useWebServiceSwitch` : Des interrupteurs pour activer ou désactiver les options.
      - `boutonAppliquer`, `boutonReinitialiser` : Des boutons pour appliquer et réinitialiser les options.
      - `options` : Les options actuelles de l'application.
    - **Méthodes** :
      - `viewDidLoad()` : Charge et applique les options sauvegardées lors du chargement de la vue.
      - `themeSwitchChanged(_:)`, `notificationsSwitchChanged(_:)`, `locationServicesSwitchChanged(_:)`, `dataSavingModeSwitchChanged(_:)`, `useWebServiceSwitchChanged(_:)` : Méthodes d'action pour les interrupteurs.
      - `boutonAppliquerPressed(_:)` : Applique les options sauvegardées.
      - `boutonReinitialiserPressed(_:)` : Réinitialise les options aux valeurs par défaut.
      - `chargerLesOptions()` : Charge les options dans l'interface utilisateur.
      - `appliquerLesOptions()` : Applique les options sauvegardées.
      - `reinitialiserLesOptions()` : Réinitialise les options aux valeurs par défaut.

## userSettingsVC.swift

### Aperçu

Le fichier `userSettingsVC.swift` est responsable de la gestion des paramètres utilisateur. La classe principale, `userSettingsVC`, hérite de `UIViewController` et permet à l'utilisateur de charger, ajouter, et déconnecter des utilisateurs, ainsi que de gérer les informations de la voiture associée à l'utilisateur.

## Propriétés

- `UNomTF`, `UPrenomTF`, `UPseudoTF`, `UMdpTF`, `VNomTF`, `VMarqueTF`, `VCouleurTF`, `chargerUser`, `chargerMDP`, `labelInfo`, `deconnexionBouton` : Des champs de texte et des boutons pour l'interface utilisateur.

## Méthodes Principales

1. `viewDidLoad()`
    - Initialise la vue.
    - Configure les champs de texte pour le mot de passe et les placeholders pour les champs de la voiture.
2. **Gestion du Mot de Passe**
    - `tapSurVoirOuCacher(_:)` : Permet de basculer entre l'affichage et le masquage du mot de passe dans le champ de texte.
3. **Chargement d'un Utilisateur**
    - `tapSurCharger(_:)` : Charge un utilisateur en fonction du pseudo et du mot de passe fournis. Met à jour les trajets de l'utilisateur et affiche un message de succès ou d'erreur.
4. **Ajout d'un Utilisateur**
    - `tapSurAjouterUser(_:)` : Ajoute un nouvel utilisateur avec les informations fournies, y compris les informations de la voiture associée. Sauvegarde les données dans Core Data.
5. **Déconnexion de l'Utilisateur**
    - `deconnecterUtilisateur(_:)` : Déconnecte l'utilisateur actuel et met à jour l'interface utilisateur pour refléter la déconnexion.

## Points à Améliorer

6. **Validation des Entrées**
7. **Optimisation des Performances**
8. **Interfaces utilisateur**
9. **Statistiques des trajets manquantes**
10. **Gestion des erreurs manquantes**