CovoitUniv documentation

Introduction

Bienvenue dans la documentation technique de **CoVoitUniv**, une application de covoiturage développée avec Flask et SQLAlchemy. Ce document présente l'architecture, les modules, ainsi que les étapes d'installation et d'utilisation de l'application.

Présentation du Projet

CovoitUniv permet aux utilisateurs de : - S'inscrire et se connecter. - Gérer leur calendrier de trajets. - Créer et mettre à jour des demandes de covoiturage. - Rechercher et offrir des trajets.

L'application repose sur plusieurs composants : - **Flask** pour le backend. - **SQLAlchemy** pour la gestion de la base de données. - Des API externes pour le géocodage et le calcul d'itinéraires. - Un système de cache pour accélérer le géocodage d'adresses.

Usage

Pour démarrer l'application, exécutez :

```
python manageReact.py
```

L'interface web sera accessible à l'adresse :
http://localhost:5000

Architecture

L'architecture du projet se compose principalement de :

- **manageReact.py** : Le point d'entrée de l'API Flask.
- **models.py** : Définit les modèles de données (User, RideRequest, DriverOffer, CalendarEntry, etc.) avec SQLAlchemy.
- **utils.py** : Contient des fonctions utilitaires (géocodage, calcul d'itinéraires, etc.).
- **pingMail.py** : Gère l'envoi d'e-mails pour notifier les utilisateurs.

Sommaire:

Modules

La documentation détaillée des modules est générée automatiquement à partir des docstrings en utilisant l'extension autodoc de Sphinx.

manageReact.**deploy_week**()

Anciennement on dupliquait user.calendar[source_week] -> toutes les semaines. Maintenant qu'on n'a plus user.calendar, on lit la table CalendarEntry. On garde la même signature JSON, mais on change la logique.

{ « user_id »: « … », « source_week »: 6 } On copie toutes les entrées de la semaine 6 vers les semaines 1..52, en évitant les doublons (check de non-duplication). Endpoint: /propagateCalendar Méthode: POST

Description:

Duplication de la semaine source (spécifiée par « source_week ») dans la table CalendarEntry pour toutes les semaines (1 à 52). Avant création d'une nouvelle entrée, on vérifie qu'il n'existe pas déjà une entrée pour la même combinaison (user_id, year, week_number, day_of_week) pour éviter la duplication.

Entrée (JSON):

- user_id (str)
- source_week (int)

Sortie (JSON):

Un message indiquant le nombre d'entrées créées.

manageReact.**find_passengers**()

Endpoint: /find_passengers Méthode: POST

Description:

Recherche des demandes de trajet (RideRequest) potentielles correspondant aux critères d'un conducteur. Compare l'heure et les adresses (départ/destination) dans le calendrier des conducteurs et passagers.

Entrée (JSON):

- user_id (str): l'ID du conducteur
- day (str): date (« YYYY-MM-DD »)
- time_slot (str): « morning » ou « evening » (optionnel, défaut « morning »)

Sortie (JSON):

Une liste de demandes potentielles avec des informations sur le passager et les itinéraires.

manageReact.**get_calendar**(*user_id*)

Endpoint: /getCal/<user_id> Méthode: GET

Description:

Récupère les entrées de calendrier pour un utilisateur donné et une semaine spécifiée (via le paramètre “indexWeek”) et reconstruit un objet JSON au format : {

« weekNumber »: <indexWeek>, « days »: [

{

« date »: « <ISO8601> », # reconstitué à partir de year, week_number, day_of_week
« startHour »: <int>, « endHour »: <int>, « departAller »: <str>, « destinationAller »: <str>, « departRetour »: <str>, « destinationRetour »: <str>, « disabled »: <bool>, « roleAller »: <str>, « roleRetour »: <str>, « validatedAller »: <bool>, « validatedRetour »: <bool>

]

}

Entrée (query parameter):

- indexWeek (int)

Sortie (JSON):

L’objet calendrier pour la semaine spécifiée, ou null si aucune donnée.

`manageReact.log_call(func)`

Décorateur pour enregistrer les appels et retours de fonctions dans le fichier trace.log.

Paramètres:
func – La fonction à instrumenter.
Renvoie:
La fonction décorée qui logue l’appel et le retour.

`manageReact.login()`

Endpoint: /login Méthode: POST

Description:

Authentifie un utilisateur en vérifiant son email et son mot de passe.

Entrée (JSON):

- email (str): Adresse e-mail.
- password (str): Mot de passe.

Sortie (JSON):

- Un token (ici, l’ID de l’utilisateur) en cas de succès, ou un message d’erreur.

`manageReact.offer_passenger()`

Endpoint: /offerPassenger Méthode: POST

Description:

Permet à un conducteur d’offrir un trajet à une demande de trajet (RideRequest) existante. Vérifie que le conducteur n’offre pas un trajet à lui-même, envoie un email de notification au passager, puis crée une entrée dans DriverOffer.

Entrée (JSON):

- driver_id (str)
- ride_request_id (str)
- departure_hour (int)

Sortie (JSON):

Un message de succès avec l’ID de l’offre créée.

`manageReact.passengers(id)`

Endpoint: /passengers/<id> Méthode: GET

Description:

Récupère la liste de tous les utilisateurs qui ne sont pas conducteurs, à l'exclusion de l'utilisateur spécifié.

Paramètre d'URL:

- id (str): Identifiant de l'utilisateur à exclure.

Sortie (JSON):

Liste des passagers (prénom, nom, adresse).

`manageReact.request_ride()`

Endpoint: /request_ride Méthode: POST

Description:

Crée ou met à jour une demande de trajet (RideRequest) pour un utilisateur pour un jour donné. Si une demande existe déjà pour ce jour, elle est mise à jour. Les informations (adresse de départ, destination, horaires) sont obtenues depuis CalendarEntry et transformées via replace_placeholders.

Entrée (JSON):

- user_id (str)
- day (str): date au format « YYYY-MM-DD »
- timeSlot (str): « morning » ou « evening »

Sortie (JSON):

Informations sur la demande de trajet créée ou mise à jour.

`manageReact.ride_offers()`

Endpoint: /rideOffers Méthode: POST

Description:

Récupère toutes les offres de trajet (DriverOffer) associées à une demande de trajet (RideRequest) donnée.

Entrée (JSON):

- ride_request_id (str)

Sortie (JSON):

Une liste d'offres comprenant l'ID de l'offre, le statut, le nom et l'adresse du conducteur.

`manageReact.save_calendar_iso()`

Endpoint: /saveCal Méthode: POST

Description:

Sauvegarde (ou met à jour) les entrées de calendrier dans la table CalendarEntry. Pour chaque jour fourni dans « calendar_changes », le système vérifie s'il existe déjà une entrée pour (user_id, year, week_number, day_of_week). Si c'est le cas, il met à jour; sinon, il crée une nouvelle entrée.

Entrée (JSON):

- user_id (str): Identifiant de l'utilisateur.
- calendar_changes (dict): Contient « weekNumber » et une liste « days » d'objets avec :
  - date (str, ISO8601)
  - startHour (int)
  - endHour (int)
  - departAller (str)
  - destinationAller (str)

- departRetour (str)
- destinationRetour (str)
- disabled (bool)
- roleAller (str)
- roleRetour (str)
- validatedAller (bool)
- validatedRetour (bool)

Sortie (JSON):

Message de succès.

manageReact.**signup**()

Endpoint: /signup Méthode: POST

Description:

Inscription d'un nouvel utilisateur. Valide l'unicité de l'email, effectue le géocodage de l'adresse, puis crée un nouvel utilisateur.

Entrée (JSON):

- email (str): Adresse e-mail de l'utilisateur.
- first_name (str): Prénom.
- last_name (str): Nom.
- address (str): Adresse postale.
- password (str): Mot de passe (en clair ou haché en production).
- is_driver (bool): Indique si l'utilisateur est conducteur.

Sortie (JSON):

Un message de succès et l'ID de l'utilisateur nouvellement créé.

manageReact.**user_info**(*id*)

Endpoint: /user/<id> Méthode: GET

Description:

Récupère et retourne les informations de l'utilisateur spécifié.

Paramètre d'URL:

- id (str): Identifiant de l'utilisateur.

Sortie (JSON):

Détails de l'utilisateur (prénom, nom, adresse, statut de conducteur).

Module de définition des modèles de base de données et des fonctions utilitaires pour la gestion du calendrier et des requêtes de covoiturage.

class models.**AddressCache**(***kwargs*)

Bases : **Model**

Modèle représentant le cache d'adresses géocodées pour éviter de répéter des requêtes de géocodage.

Attributs :

- id (int) : Clé primaire, autoincrémentée.
- address (str) : Adresse unique.
- lat (float) : Latitude.
- lon (float) : Longitude.

address

id

lat

lon

query: *t.ClassVar[Query]*

A SQLAlchemy query for a model. Equivalent to `db.session.query(Model)`. Can be customized per-model by overriding `query_class`.

> **Avertissement:**
>
> The query interface is considered legacy in SQLAlchemy. Prefer using `session.execute(select())` instead.

class models.**CalendarEntry**(***kwargs*)

Bases : `Model`

Modèle représentant une entrée dans le calendrier d'un utilisateur.

Chaque entrée correspond à un jour (défini par l'année, le numéro de semaine ISO et le jour ISO) et contient des informations sur les horaires, adresses de départ/destination, et le rôle (conducteur ou passager).

Attributs :

- id (str) : Identifiant UUID de l'entrée.
- user_id (str) : Référence à l'utilisateur.
- year (int) : Année (ex: 2025).
- week_number (int) : Numéro de semaine ISO (1..53).
- day_of_week (int) : Jour de la semaine ISO (1 = lundi, 7 = dimanche).
- start_hour (int) : Heure de début (optionnel).
- end_hour (int) : Heure de fin (optionnel).
- depart_aller (str) : Adresse de départ pour l'aller.
- destination_aller (str) : Adresse de destination pour l'aller.
- depart_retour (str) : Adresse de départ pour le retour.
- destination_retour (str) : Adresse de destination pour le retour.
- disabled (bool) : Indique si l'entrée est désactivée.
- role_aller (str) : Rôle à l'aller (par exemple, « passager » ou « conducteur »).
- role_retour (str) : Rôle au retour.
- validated_aller (bool) : Indique si l'entrée a été validée pour l'aller.
- validated_retour (bool) : Indique si l'entrée a été validée pour le retour.

day_of_week

depart_aller

depart_retour

destination_aller

destination_retour

disabled

end_hour

id

query: *t.ClassVar[Query]*

A SQLAlchemy query for a model. Equivalent to `db.session.query(Model)`. Can be customized per-model by overriding `query_class`.

> **Avertissement:**

> The query interface is considered legacy in SQLAlchemy. Prefer using `session.execute(select())` instead.

role_aller

role_retour

start_hour

user_id

validated_aller

validated_retour

week_number

year

class models.**DriverOffer**(***kwargs*)

Bases : **Model**

Modèle représentant une offre de covoiturage proposée par un conducteur en réponse à une demande de trajet.

Attributs :

- id (str) : Identifiant UUID de l'offre.
- driver_id (str) : Référence à l'utilisateur conducteur.
- ride_request_id (str) : Référence à la demande de trajet.
- status (str) : Statut de l'offre (ex: « offered », « accepted », « declined »).
- departure_hour (int) : Heure de départ proposée.

departure_hour

driver_id

id

query: *t.ClassVar[Query]*

A SQLAlchemy query for a model. Equivalent to `db.session.query(Model)`. Can be customized per-model by overriding **query_class**.

> **Avertissement:**
>
> The query interface is considered legacy in SQLAlchemy. Prefer using `session.execute(select())` instead.

ride_request_id

status

class models.**RideRequest**(***kwargs*)

Bases : **Model**

Modèle représentant une demande de covoiturage.

Attributs :

- id (str) : Identifiant UUID de la demande.
- user_id (str) : Référence à l'utilisateur (passager).
- day (str) : Date de la demande (ex: « 2025-02-03 »).
- address (str) : Adresse de départ.

- destination (str) : Adresse de destination.
- lat (float) : Latitude de l'adresse de départ.
- lon (float) : Longitude de l'adresse de départ.
- start_hour (int) : Heure de début (ex: 9).
- end_hour (int) : Heure de fin (ex: 17).
- matched_driver_id (str) : Identifiant du conducteur ayant accepté la demande (optionnel).

address

day

destination

end_hour

id

lat

lon

matched_driver_id

query: *t.ClassVar[Query]*

A SQLAlchemy query for a model. Equivalent to `db.session.query(Model)`. Can be customized per-model by overriding `query_class`.

> **Avertissement:**
>
> The query interface is considered legacy in SQLAlchemy. Prefer using `session.execute(select())` instead.

start_hour

timeslot

user_id

class models.**User**(***kwargs*)

Bases : `Model`

Modèle représentant un utilisateur.

Attributs :

- id (str) : Identifiant UUID de l'utilisateur.
- email (str) : Adresse e-mail unique.
- first_name (str) : Prénom.
- last_name (str) : Nom.
- password (str) : Mot de passe (en clair ou haché).
- address (str) : Adresse de l'utilisateur.
- is_driver (bool) : Indique si l'utilisateur est conducteur.
- calendar (str) : Texte de calendrier (optionnel).

address

calendar

email

first_name

id

is_driver

last_name

password

query: *t.ClassVar[Query]*

A SQLAlchemy query for a model. Equivalent to `db.session.query(Model)`. Can be customized per-model by overriding `query_class`.

> **Avertissement:**
>
> The query interface is considered legacy in SQLAlchemy. Prefer using `session.execute(select())` instead.

models.**convert_iso_string_to_calendar_slots**(*date_str*)

Convertit une chaîne ISO8601 (ex: « 2025-03-30T22:00:00Z ») en un tuple (year, week_number, day_of_week) en interprétant l'heure donnée comme étant en UTC, puis en la convertissant en heure locale (« Europe/Paris »). Cela permet de gérer correctement les décalages horaires et les changements d'heure (DST).

Paramètres:
date_str – Chaîne de date au format ISO8601.
Renvoie:
Tuple (year, week_number, day_of_week)

models.**local_midnight_to_utc_iso**(*year, week, day_of_week*)

Reconstruit la date correspondant à minuit local pour les paramètres donnés (year, week, day_of_week) en utilisant la zone « Europe/Paris », puis la convertit en heure UTC sous forme d'une chaîne ISO8601.

Par exemple, si minuit local en « Europe/Paris » est à 00:00 et que le décalage est de +2, la fonction renverra une chaîne avec « T22:00:00Z » correspondant au jour précédent en UTC.

Paramètres:
- **year** – Année (int)
- **week** – Numéro de semaine ISO (int)
- **day_of_week** – Jour de la semaine ISO (int, 1 = lundi, 7 = dimanche)

Renvoie:
Chaîne ISO8601 (str) représentant l'heure UTC.

models.**to_real_date**(*year, week_number, day_of_week*)

Convertit les valeurs ISO (year, week_number, day_of_week) en une date réelle.

Paramètres:
- **year** – Année (int)
- **week_number** – Numéro de semaine ISO (int)
- **day_of_week** – Jour de la semaine ISO (int, 1 = lundi, 7 = dimanche)

Renvoie:
Date (datetime.date)

Module de fonctions utilitaires pour la gestion du géocodage, du calcul d'itinéraire, et du remplacement de placeholders dans le projet.

utils.**geocode_address**(*address*)

Effectue le géocodage d'une adresse en utilisant l'API adresse.data.gouv.fr, avec mise en cache pour accélérer les calculs.

Cette fonction vérifie d'abord si l'adresse est présente dans le cache (table AddressCache). Si c'est le

cas, les coordonnées mises en cache sont retournées. Sinon, une requête est effectuée vers l'API, les coordonnées (latitude et longitude) sont extraites, arrondies à 6 décimales, enregistrées dans le cache, puis retournées.

Paramètres:
address – (str) Adresse à géocoder.
Renvoie:
(tuple) (lat, lon) si le géocodage est réussi, sinon (None, None).

`utils.get_route(start_coords, end_coords, label)`
Calcule un itinéraire entre deux points en utilisant le service OSRM.

Cette fonction envoie une requête à l'API OSRM pour obtenir l'itinéraire entre les coordonnées de départ et d'arrivée. Elle renvoie un dictionnaire contenant : - Le label donné, - La distance (en km), - La durée (en minutes), - Une géométrie simplifiée sous forme de liste de coordonnées.

Paramètres:
- **start_coords** – (tuple) Coordonnées de départ sous la forme (lat, lon).
- **end_coords** – (tuple) Coordonnées d'arrivée sous la forme (lat, lon).
- **label** – (str) Label pour identifier cet itinéraire.

Renvoie:
(dict) Dictionnaire avec « label », « distance », « duration » et « geometry », ou None si l'itinéraire n'est pas trouvé.

`utils.replace_placeholders(obj, user_address)`
Parcourt récursivement "obj" (peut être un dict, une liste ou une chaîne) et remplace certains mots-clés par des adresses complètes.

Notamment, si le mot « Maison » est rencontré, il sera remplacé par "user_address". D'autres remplacements prédéfinis sont appliqués (Villetaneuse, Bobigny, Saint-Denis).

Paramètres:
- **obj** – (dict, list, str, ou autre) Objet à traiter.
- **user_address** – (str) Adresse de l'utilisateur à utiliser pour le remplacement.

Renvoie:
L'objet mis à jour avec les remplacements effectués.

`pingMail.send_email_via_gmail(sender_email, sender_password, to_email, subject, body)`
Sends an email using Gmail's SMTP server.

`pingMail.send_offer_email(driver, passenger, departure_hour, sender_email, sender_password)`
Envoie deux emails : 1. Au passager pour lui proposer un trajet. 2. Au conducteur pour confirmer l'offre avec les infos du passager.

Indices and tables

- Index
- Index du module
- Page de recherche

CovoitUniv

Search Go

Navigation

Sommaire:

Modules