InnoFT: API Documentation

IUCD

February 10, 2024

Overview

This API provides functions to interact with a database containing information about users, routes, and places. The main functionalities include retrieving user, route, and place details, as well as subscribing to a route. The API is designed to be used in an asynchronous environment and is based on the SQLAlchemy library for database interactions.

Base URL

The base URL for this API is not applicable since it interacts directly with a database.

Authentication

No authentication is required to use this API.

Endpoints

1. Get User by ID

Endpoint:

`/get_user_by_id`

Method:

`POST`

Parameters:

- `user_id` (integer): The unique identifier of the user.

Response:

- `User` (object): Details of the user.

2. Get User by Telegram ID

Endpoint:

`/get_user`

Method:

`POST`

Parameters:

- `telegram_id` (integer): The Telegram ID of the user.

Response:

- `User` (object): Details of the user.

3. Get Route by ID

Endpoint:

`/get_route`

Method:

`POST`

Parameters:

- `route_id` (integer): The unique identifier of the route.

Response:

- `Route` (object): Details of the route.

4. Get Place by ID

Endpoint:

`/get_place`

Method:

`POST`

Parameters:

- `place_id` (integer): The unique identifier of the place.

Response:

- `Place` (object): Details of the place.

5. Get Place by Name

Endpoint:

`/get_place_by_name`

Method:

`POST`

Parameters:

- `place_name` (string): The name of the place.

Response:

- `Place` (object): Details of the place.

6. Get Passenger Routes

Endpoint:

`/get_passenger_routes`

Method:

`POST`

Parameters:

- `telegram_id` (integer): The Telegram ID of the passenger.

Response:

- `[Route]` (list of objects): List of routes associated with the passenger.

7. Get Driver Routes

Endpoint:

`/get_driver_routes`

Method:

`POST`

Parameters:

- `telegram_id` (integer): The Telegram ID of the driver.

Response:

- `[Route]` (list of objects): List of routes associated with the driver.

8. Get Routes Filtered

Endpoint:

`/get_routes_filtered`

Method:

`POST`

Parameters:

- `place_from` (string): The starting place.
- `place_to` (string): The destination place.
- `passengers_amount` (integer): The number of available passenger seats.
- `cost` (integer): The maximum cost of the route.
- `date` (string): The date of the route in the format "dd.mm.yyyy".
- `time` (string): The time interval of the route in the format "hh:mm" or "hh:mm-hh:mm".

Response:

- `[Route]` (list of objects): List of routes matching the specified criteria.

9. Subscribe to Route

Endpoint:

`/subscribe_route`

Method:

`POST`

Parameters:

- `telegram_id` (integer): The Telegram ID of the passenger.
- `chosen_route_id` (integer): The unique identifier of the chosen route.
- `passenger_amount` (integer): Number of passengers to be occupied.

Response:

- None

Notes:

- This function updates the database to reflect the passenger's subscription to the specified route.
- It decreases the `available_places` on the route by passenger_amount, indicating that a passenger has claimed a seat.
- The `chosen_route_id` should correspond to a valid and available route.
- After a successful subscription, the passenger is considered part of the route and is expected to board at the specified time and place.
- If the subscription is unsuccessful, an error will be raised, providing information about the issue.

10. Cancel subscription

Endpoint:

`/cancel_subscription`

Method:

`POST`

Parameters:

- `telegram_id` (integer): The Telegram ID of the passenger.
- `chosen_route_id` (integer): The unique identifier of the chosen route.

Response:

- None

Notes:

- This function updates the database to reflect the passenger's subscription cancellation to the specified route.
- It increases the `available_places`, indicating that a passenger has freed up a seat.
- The `chosen_route_id` should correspond to a valid and available route.
- After a successful subscription cancellation, the passenger is not considered as a part of the route.
- If the cancellation is unsuccessful, an error will be raised, providing information about the issue.

11. Get Route Passengers

Endpoint:

`/get_route_passengers`

Method:

`POST`

Parameters:

- `chosen_route_id` (integer): The unique identifier of the chosen route.

Response:

- `[User]` (list of objects): List of users associated with the route.

Data Models

User

- `id` (integer): Unique identifier of the user.
- `telegram_id` (integer): Telegram ID of the user.
- `username` (string): Username of the user.
- `is_admin` (boolean): Indicates whether the user is an admin.

Route

- `id` (integer): Unique identifier of the route.
- `driver_id` (integer): Unique identifier of the driver.
- `place_from_id` (integer): Unique identifier of the starting place.
- `place_to_id` (integer): Unique identifier of the destination place.
- `available_places` (integer): Number of available passenger seats.
- `cost` (integer): Cost of the route.
- `date_field` (string): Date of the route in the format "dd.mm.yyyy".
- `time_field` (Time): Time of the route.

Place

- `id` (integer): Unique identifier of the place.
- `name` (string): Name of the place.

Database Structure

The API interacts with a relational database using SQLAlchemy. Below is the structure of the database, including tables for users, routes, places, and the association table for passenger routes.

1. **Users Table (`users`)**
   - `id` (Integer): Unique identifier for the user.
   - `telegram_id` (BigInteger): Telegram ID of the user (non-nullable).
   - `username` (String): Username of the user (non-nullable).
   - `is_admin` (Boolean): Indicates whether the user is an admin (default is False).

   Relationships:

   - `routes` (Many-to-Many relationship): Associated routes for the user.

2. **Routes Table (`routes`)**
   - `id` (Integer): Unique identifier for the route.
   - `driver_id` (Integer): Unique identifier for the driver.

   Relationships:

   - `users` (Many-to-Many relationship): Associated users (passengers) for the route.
   - `place_from_id` (ForeignKey): Foreign key referencing the `id` column in the `places` table.
   - `place_to_id` (ForeignKey): Foreign key referencing the `id` column in the `places` table.

   Additional Route Details:

   - `place_from_id` (Integer): Unique identifier for the starting place.
   - `place_to_id` (Integer): Unique identifier for the destination place.
   - `available_places` (Integer): Number of available passenger seats.
   - `cost` (Integer): Cost of the route.
   - `date_field` (String): Date of the route in the format ”dd.mm.yyyy”.
   - `time_field` (Time): Time of the route.

3. Places Table (`places`)

- `id` (Integer): Unique identifier for the place.
- `name` (String): Name of the place.

4. Passenger Routes Association Table (`passenger_routes`)

- `user_id` (Integer): Foreign key referencing the `id` column in the `users` table.
- `route_id` (Integer): Foreign key referencing the `id` column in the `routes` table.
- `amount_of_passengers` (Integer): Value that indicates amount of occupied by user places.

5. Database Initialization (`db_engine_start` Function)

The function `db_engine_start` initializes the database, creating the necessary tables based on the defined schema.

Note:

- The database structure is designed to support relationships between users, routes, and places.
- The `passenger_routes` table serves as an association table to represent the many-to-many relationship between users and routes.
- Foreign key constraints are used to maintain data integrity and ensure proper relationships between tables.