**Project Implementation**

As per the architecture diagram we have used micro-services architecture, each service is inside separate folder and there are some common utility which are used by each of these services (present in common\_utils folder)

Flask is used for exposing http end points for each of these services, “service.py” needs to be executed to start the service for example: python3 user\_service/service.py will start user services.

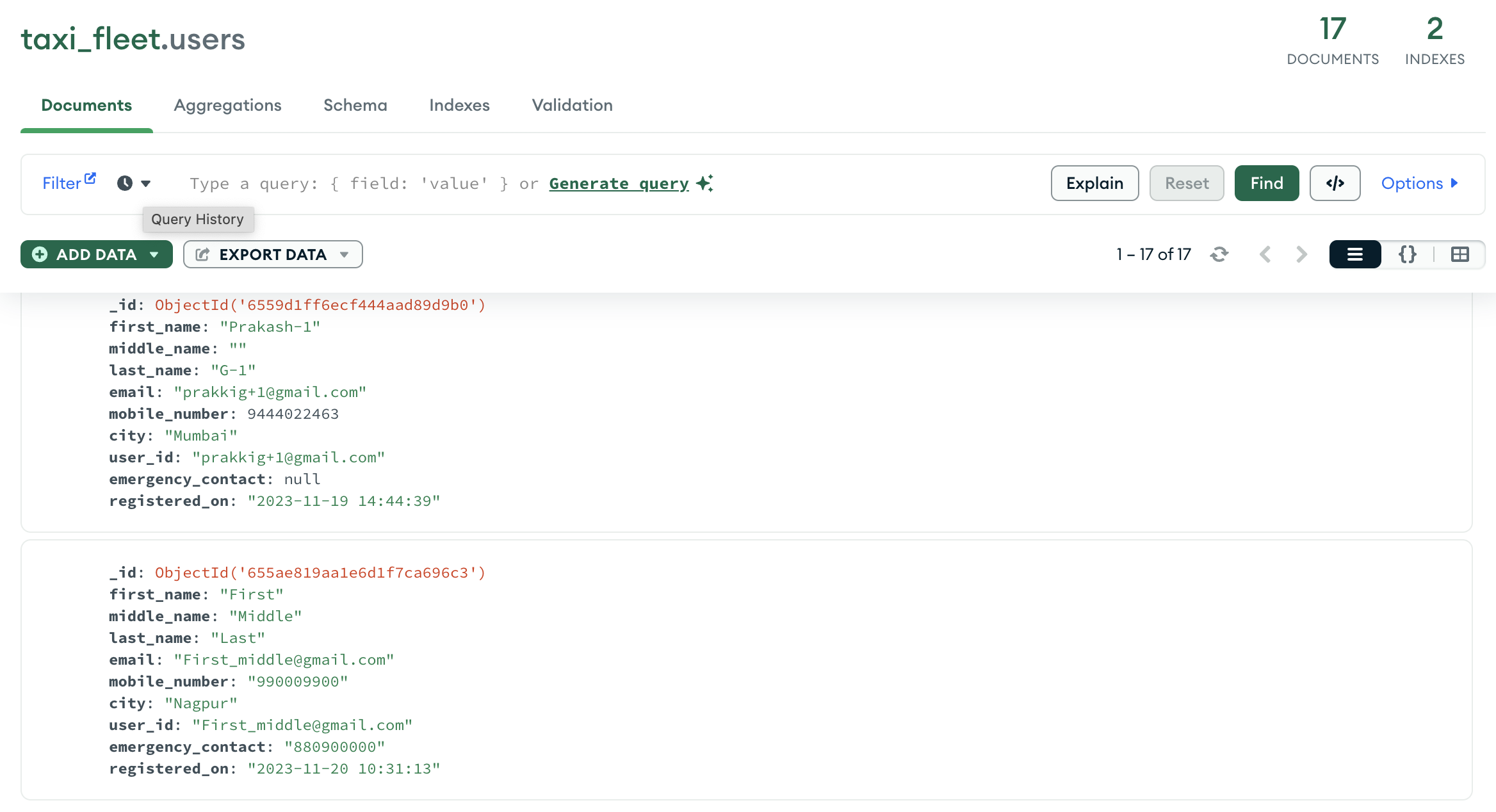
1. **User service (user\_service folder)**

This service is responsible for user registration, driver registration and taxi registration, unique key is added for each of them to avoid duplication. This service will create/update three collections

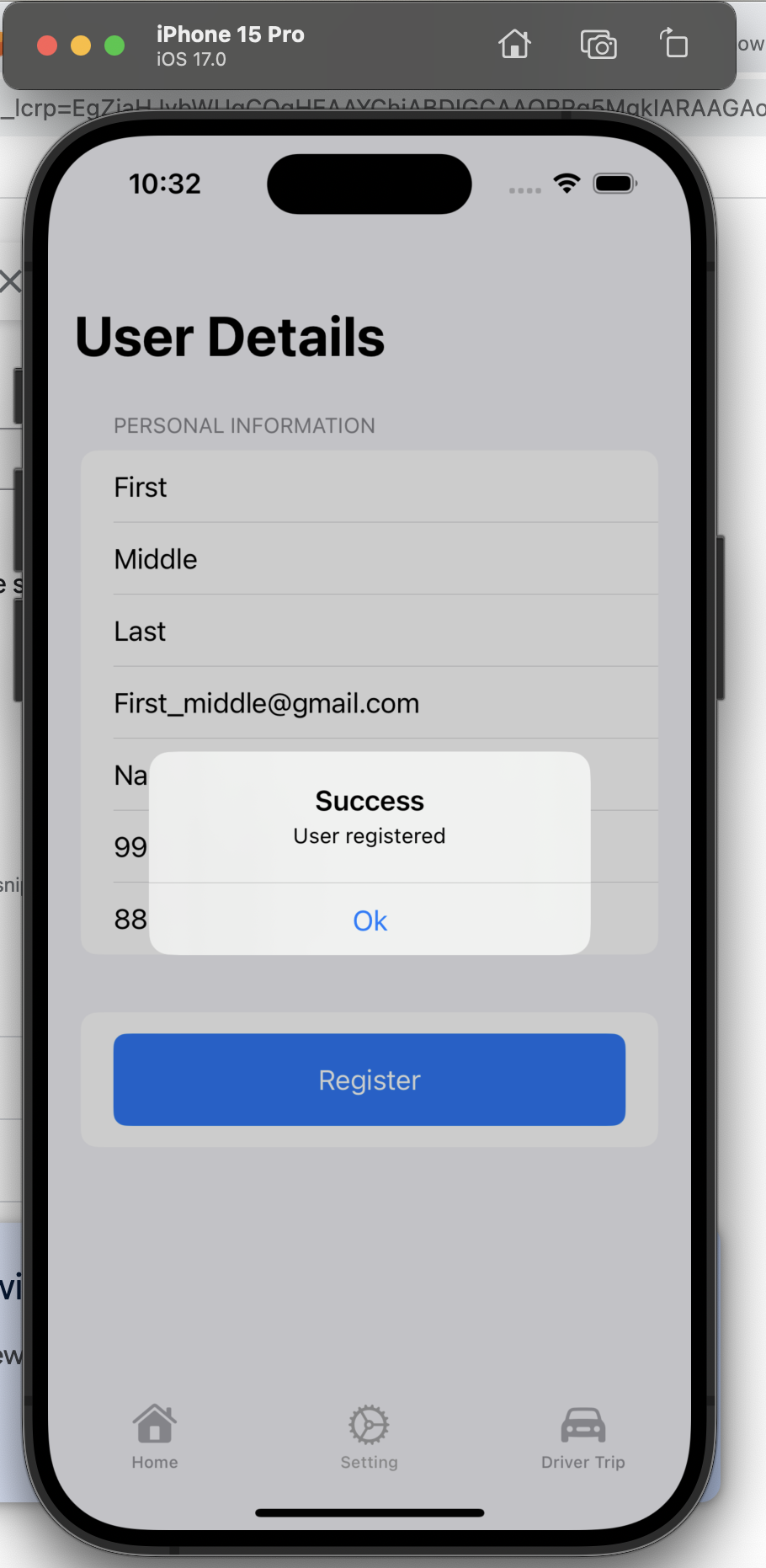
users- collection used to store cab users

drivers- collection used to store driver

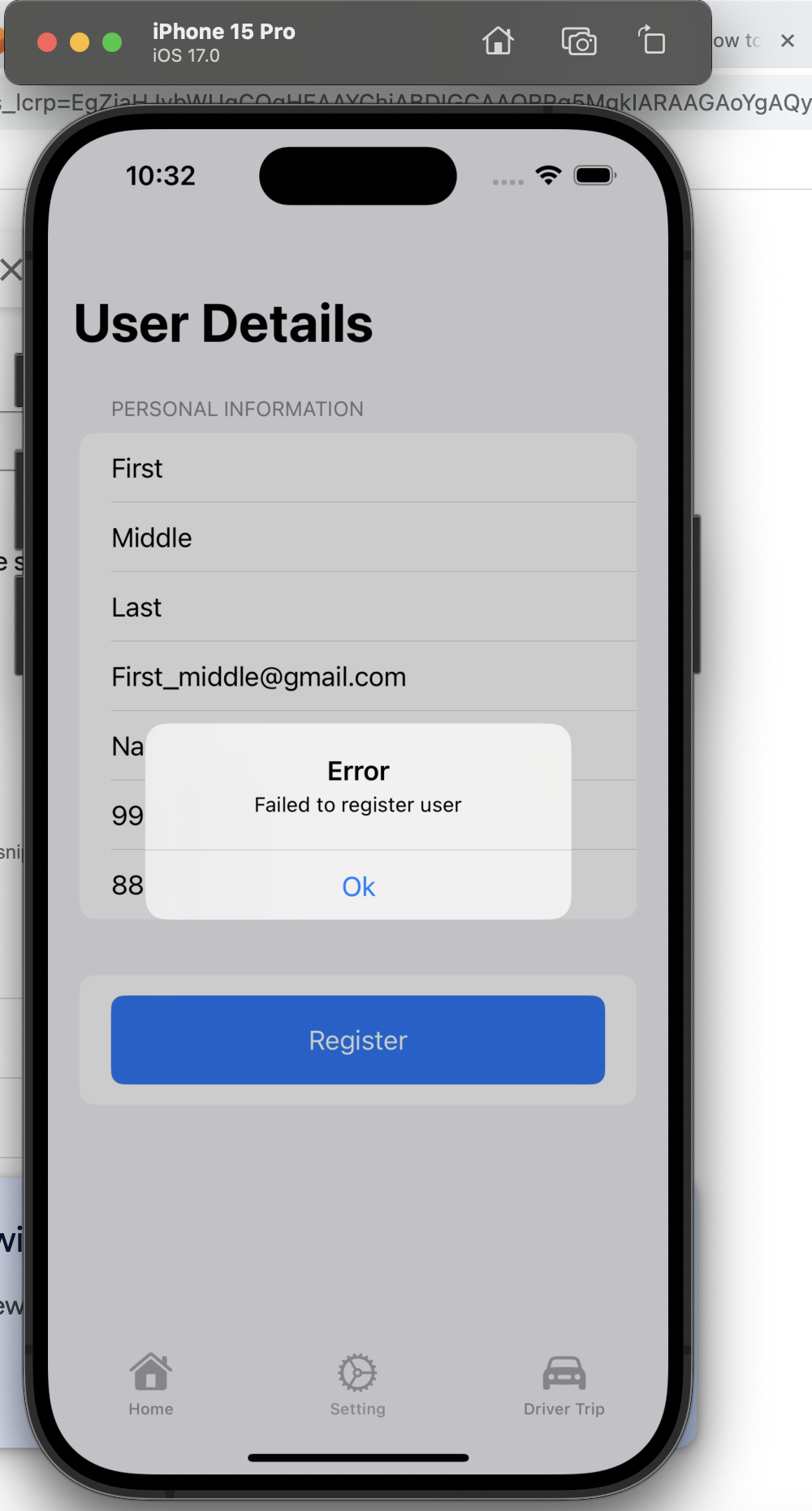
taxis – collection used to store taxi details



1 Sample User Document from Database



2 User Registration from mobile



3 Uniqueness, same user trying to register again

Note: You can refer to client\_cab\_user folder and script setup\_database.py, this file contains http call made to User service for adding user, driver and taxi.

1. **Live Location Service (location\_service folder)**

Every taxi will be sending their live location via http call to this service and this service will store that into the database, additionally there are API’s to mark taxi as booked (when driver confirms the booking) or marked taxi as free (when trip is ended/booking is cancelled), there are other additional API’s I have just mentioned 3 API which are critical to flow that was requested.

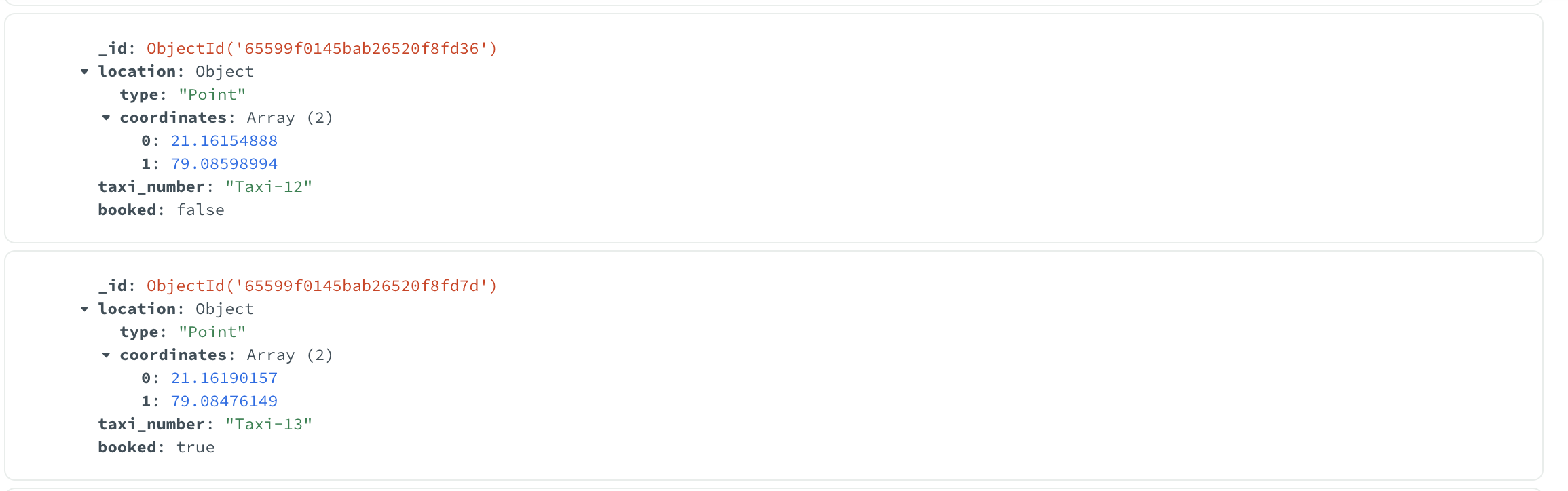


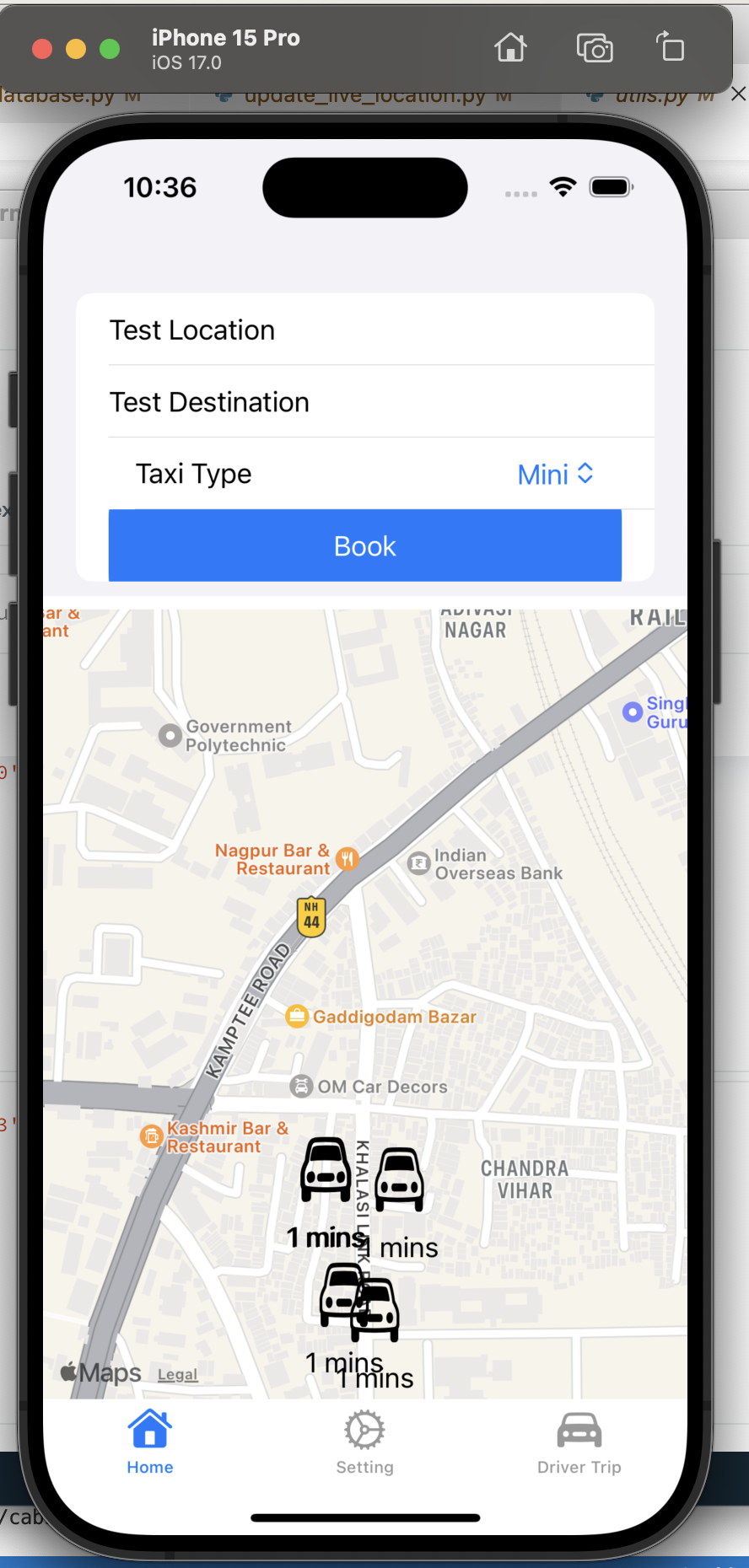
Figure 4Live Location Document sample for Booked and Free

Note: You can refer to client\_cab\_user folder update\_live\_location.py file on how live location is sent to service.

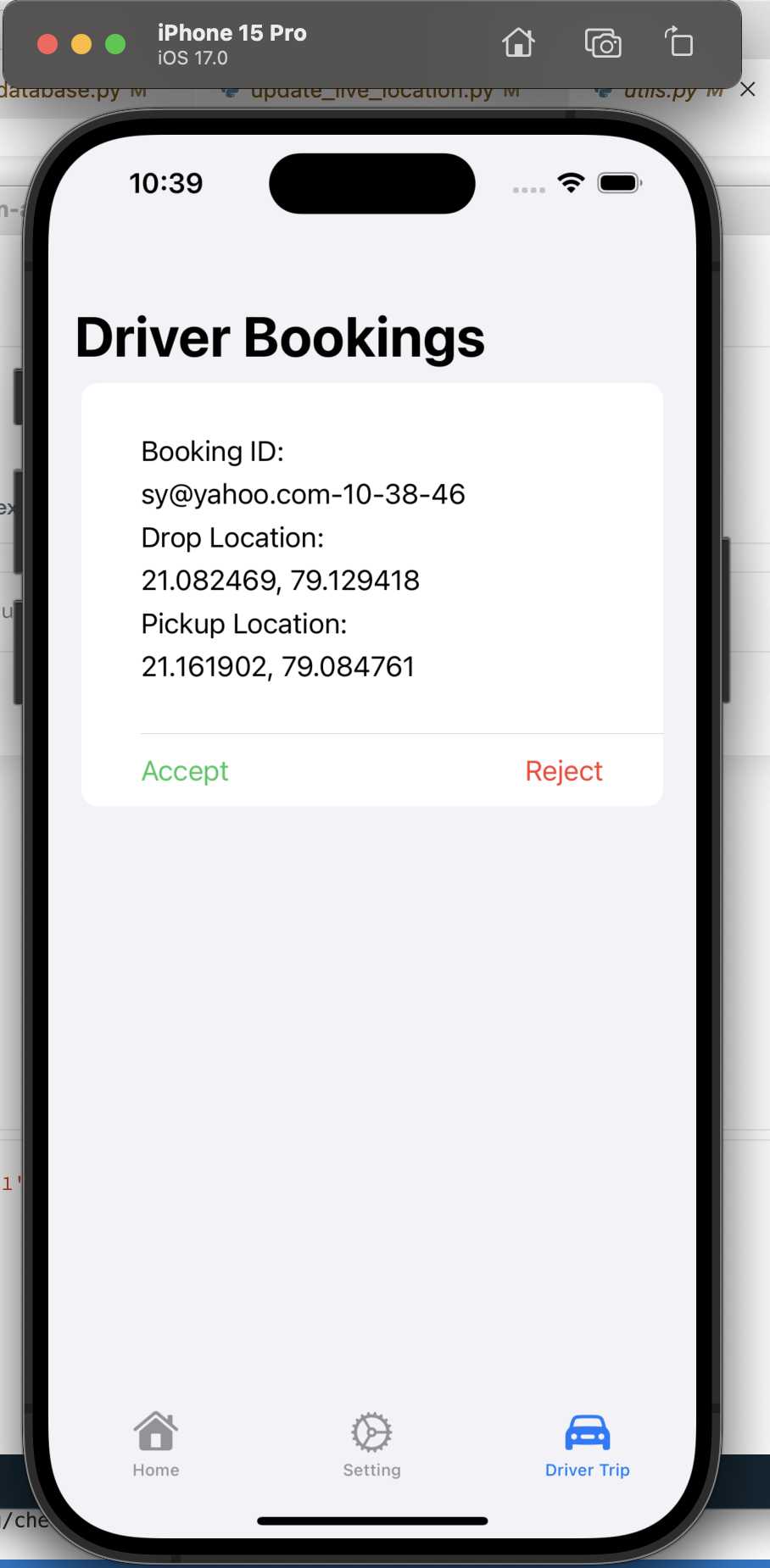
1. **Booking Service and Notification Service (booking\_service folder)**

This service caters to

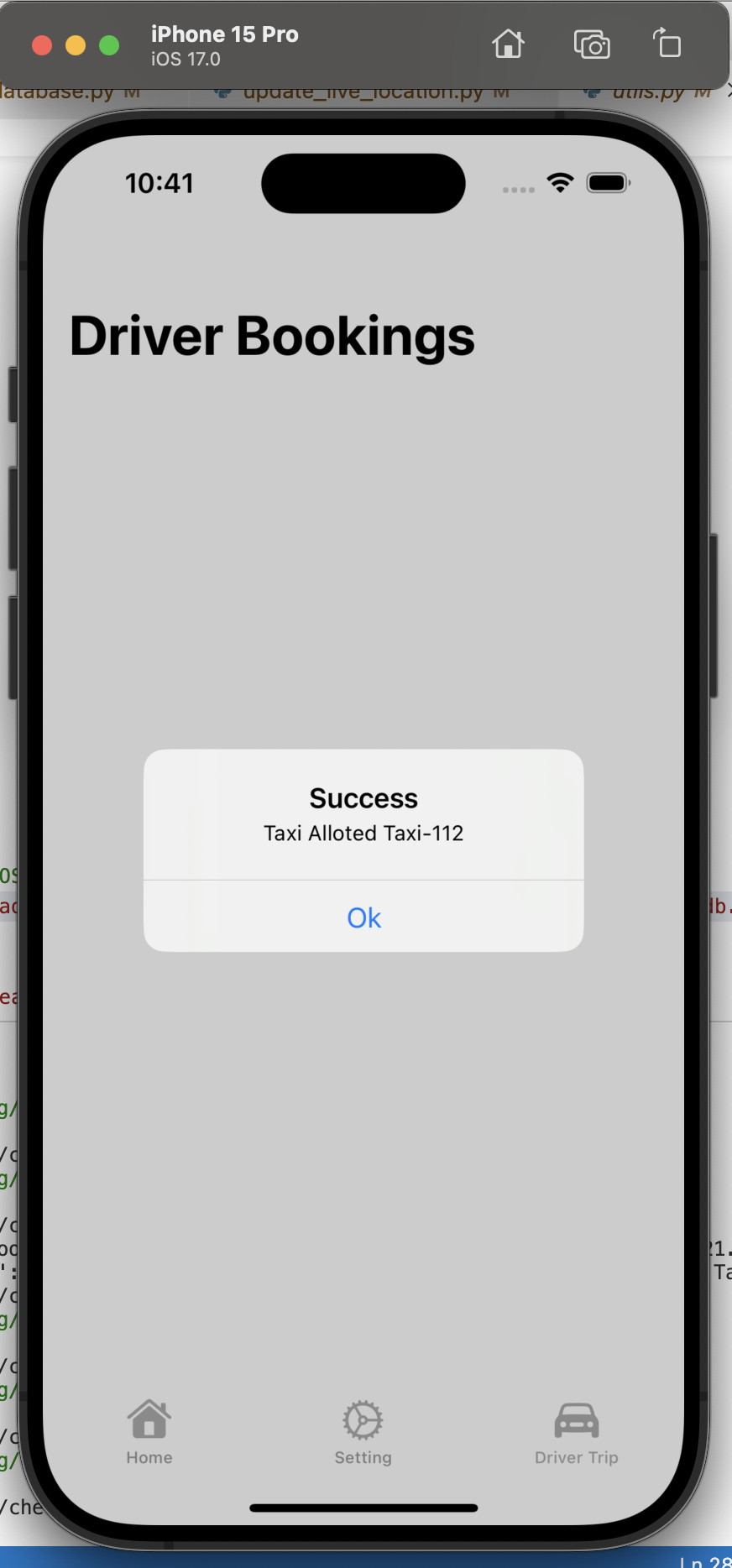
1. Find nearest cab given user location
2. User requesting to book a cab
3. Driver polling in to see if there is any request for them (once you find the cab nearest to user they are added in cache which driver queries)
4. Driver Accept/Reject for a particular booking
5. Providing user with cab confirmation

****

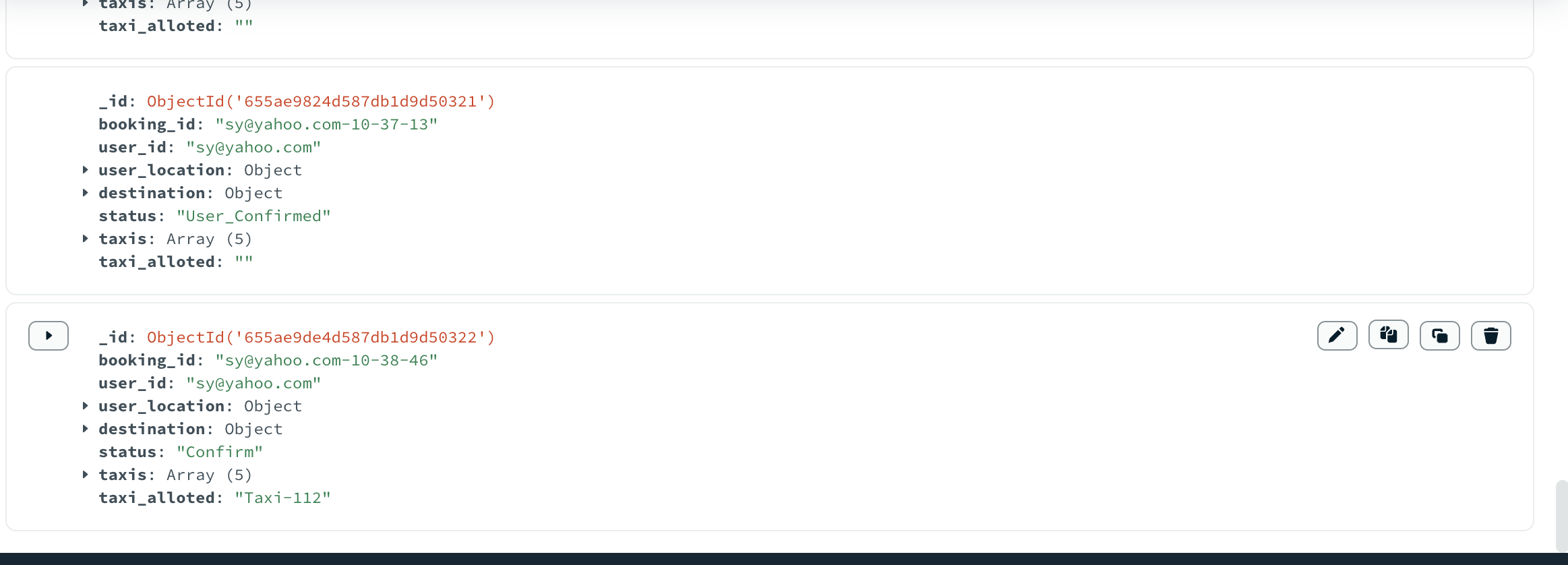
**5** Nearest cab to User along with time

****

**6** Notification shown to driver for booking with Accept and reject

****

**7** Cab confirmation sent to user

****

**8** Sample Booking Document

**GUI Code**

GUI screen shot that are shown is implemented in SWIFTUI on XCODE, so if you don’t have Macbook then you will not be able to see GUI code.

We have attached screen shot and had given live demo of the working GUI in project demo session.