1. Utilize Django REST Framework

○ Create necessary models for Ride, User, and RideEvent. ✓

○ Implement serializers for the models to enable JSON serialization/deserialization. ✓

○ Use Viewsets for managing CRUD operations. ✓

2. Authentication

○ Implement authentication to ensure only users with the role 'admin' can access the

API. ✓

3. Ride List API

○ Implement an API which returns a list of Rides ✓

○ Each Ride should include related RideEvents and Users (id\_rider, id\_driver) ✓

○ Support pagination ✓

○ Support filtering Ride status, Rider email ✓

○ Support for sorting by pickup\_time ✓ and distance to pickup given a GPS position as

input in the same API. Must implement both sorting options as efficiently as possible

with the assumption that the ride table will be very large. The sorting implementation

should still support pagination.

4. Performance

○ Since we anticipate that the RideEvent table will be very large, the Ride List API

should return an additional field on Ride named 'todays\_ride\_events' which only

retrieves RideEvents that occurred in the last 24 hours. For performance reasons, we

should never retrieve the full list of RideEvents in the SQL queries that Django

generates. You should also use advanded Django features to retrieve this data in a

way that minimizes the number of SQL queries generated.

○ The number of database queries for the Ride List API should be minimized.

Retrieving the ride list with the related driver, rider, and RideEvents can be achieved

with 2 queries (3 if you include the query required to get the total count used in

Pagination).