when i click the food recommendation page from the main page, it should take me to a page where:

1. there are search feilds for food spots on the bassis of location, price, name

2. below that, there are small boxes of different restuarents with toppest ratings on the top. we only need to display the first few top rated restaurents.

3. in the box of a restauernt, it needs the attributes:

a. Restaurent name

b. Location

c. price(avg price per head)

d. Overall Rating(out of 5)

4. if we click a box, it should show further details like:

a. Restaurant name

b. Location

c. price(avg price per head)

d. Food taste rating(out of 5)

e. service rating(out of 5)

f. Ambience Rating(out of 5)

g. overall rating(out of 5)(average of food, service and ambience rating)

h. Top personnel recommendation

i. Description

now i will tell u how the front end should look like. according to that, tell me how to implement it in django:

1. I wnat to add a recommendation about a food spot or search about a particular rrestaurent

2. i open the 'food recomnedation' option from the main page and i am directed towards the food recomnedationpage

3. By default, there are small box like structed for different hotels which shows the hotel name, location, overall rating, overall price, image. which are arranged in the on the basis of their overall\_rating(top rated restaurants will come first)

4. on the top part, there are search options for restaurants base on:

a. Restaurant name

b. Location

c. overall\_price

after a search is entered, only the restaurents whihc match the search will come as boxes.

5. now after further deatils of the restaurent will be shown:

a. name, location, overall\_price, overall\_rating and images

b. recommendations made by different users(User name and other details, 3 tytpes of rating, users overall rating(from the 3), price oer head, top recommendation, description) about that restaurant.

6. A current logged in user can "Add a Review" in two ways:

a. under the detailed restuarent/recommendation page(of a particular restaurennt), there will be an option called "add your review", where the user can add reviews about that particular restaurent

Details inputted:

food\_rating, service\_rating, ambience\_rating, top\_recommendation, avg\_user\_price\_per\_head, description, images

b. there is an option in the main food recomendation page itself(along with the search button and the boxes) called "Add a review", where we can add a review about a new restaurent.

two ways this can go:

(i) while inputting the restaurent name, if it already exists, then you will be redirected to page mentioned in option 6. a).

(i) if its a new restaurent, the input the follwoing details:

name, location, food\_rating, service\_rating, ambience\_rating, top\_recommendation, avg\_user\_price\_per\_head, description, images

Based on the frontend functionality you've described, here's how you can implement it in your Django food recommendation app:

1. \*\*Recommendation Model and Serializers:\*\*

- You have already created the `Recommendation` model and serializer, which is a good start.

2. \*\*Restaurant Model and Serializers:\*\*

- You have also created the `Restaurant` model and serializer, which is suitable for storing restaurant details.

3. \*\*API Views and URLs:\*\*

- Create API views for listing, creating, updating, and deleting recommendations and restaurants. You have already set up serializers and models, so creating views should be straightforward.

- Configure the URLs in your `urls.py` file for these views. This will define the API endpoints for interacting with recommendations and restaurants.

4. \*\*Frontend Integration (Flutter):\*\*

- In your Flutter frontend, create screens and UI components for the following:

a. Display a list of restaurants with their names, locations, overall ratings, prices, and images. You can fetch this data from the Django backend using API requests.

b. Implement search functionality for restaurants based on name, location, and overall price. This can be done by making API requests to filter restaurants based on user input.

c. Display restaurant details: When a user clicks on a restaurant, navigate to a screen that shows restaurant-specific details, including name, location, overall price, overall rating, and images.

d. List recommendations for the selected restaurant, including user names, ratings, overall user ratings, price per head, top recommendations, descriptions, and images. Make an API request to fetch and display this data.

e. Allow users to add a review for a restaurant. Create a form for users to input food rating, service rating, ambience rating, top recommendation, average price per head, description, and images. When a user submits a review, send a POST request to the Django backend to create a new recommendation.

f. Provide an option to "Add a Review" from the main food recommendation page. This form should include fields for the restaurant name (existing or new), and when a user submits this form, check whether the restaurant exists in the database. If it exists, navigate to the specific restaurant detail page for adding a review. If it's a new restaurant, create the restaurant first and then allow the user to add a review as described in step e.

5. \*\*API Endpoints for Creating Recommendations:\*\*

- Your Django backend should handle the creation of new recommendations. Define API endpoints to support adding recommendations for existing and new restaurants.

6. \*\*User Authentication:\*\*

- Implement user authentication in your Django backend to identify users when they add reviews. This can be done using token-based authentication or another authentication method of your choice.

7. \*\*Rating and Review Calculations:\*\*

- Calculate average user ratings and average price per head for restaurants based on recommendations from users. Update these values in the `Restaurant` model accordingly. You can use Django signals, as you've mentioned earlier, to calculate and update these values when new recommendations are added.

8. \*\*Testing:\*\*

- Test your frontend screens and API requests to ensure that you can create, list, and view restaurants and recommendations.

By following these steps, you can create the food recommendation system in your Django app that matches the frontend functionality you've described. Make sure to add error handling and validation to your forms and API views to provide a smooth user experience.

imagine there is no data in my both recommendation and restaurent db. the first entry in the restaurent db is created only when a user tries to make a recommendation for that restaurant. so i want to implement "Add review from main page" logic like this:

Provide an option to "Add a Review" from the main food recommendation page. This form should include fields for the :

-Restaurant name(saved to the restaurant db)

-Restuarent location(saved to the restaurant db)

-food\_rating

-service\_rating

-ambience\_rating

-top\_recommendation

-description

-image

after the user enters this form, the following cases should be properly handled:

1. the restaurant name should be cross checked with the existing restaurent db. if restaurant not found, then create the restaurant instance first(by providing the given restaurant name and location, and also set overall\_price, overall\_rating to 0 as of now). And then, create an entry in the Recommendation table for that corresponsing restaurant. (using the values given in the form by the user).

2. If the restaurant name already exists(then it means somebody has already made a recommendation for that restaurant). then update the recommendation table with the users current entry).

it is still not working. I'll elaborate on this more:

there is are tables called foodrecommendation\_recommendation and foodrecommendation\_restaurant(assume there are no entries as of now). here when i input the following values:

restaurant\_name - Thaal Kitchen

restaurant\_location - Kakkanad

food\_rating - 4.8

service\_rating - 4.9

ambience\_rating - 4.6

top\_recommendation - Shawarma

avg\_user\_price\_per\_head - 125

description - Good and budget place for college students

images - download (l).jpg

an entry in both the restaurant(because there is no same restaurant with the same name in the table) and recommendation table. and after that, consider i put the follwoing entry:

restaurant\_name - Thaal Kitchen

restaurant\_location - Kakkanad

food\_rating - 4.8

service\_rating - 4.9

ambience\_rating - 4.6

top\_recommendation - Shawarma

avg\_user\_price\_per\_head - 140(different price)

description - Good and budget place for college students

images - download (l).jpg

"key" "created" "user\_id"

"1924f6ea25f128f0be9d2769895453a7c097f2d5" "2023-11-15 11:44:21.745731+05:30" 21

"22a66a02753c1dad5ee6ad2ba3d5af0db182e6a1" "2023-10-24 22:54:34.001151+05:30" 16

"2973b07b9c75f6f3d741fbc8eda65e0cb89e3a96" "2023-11-15 12:35:34.212983+05:30" 23

"43d00e043af6125f161913884dc23fa2607c394d" "2023-11-15 18:07:33.50905+05:30" 26

"767922b822aa7892aeea1f252d4e3e5540147a39" "2023-11-03 09:28:11.448438+05:30" 20

"9513497baac9cc33d65dc00976cc244a213fdaf0" "2023-10-24 15:46:49.232472+05:30" 14

"a314931ba342408620863a250c7c9bb309db9788" "2023-10-27 09:19:42.380774+05:30" 18

"b8a38793784de897e1cbfb6d319402e4dae04b27" "2023-10-24 21:17:17.737691+05:30" 15

"c0e0592b04adb153db7742f5657812c6032110ca" "2023-10-26 19:59:16.062645+05:30" 17

"ef02538a47a6c868b2c59020e339498d3942a5a6" "2023-10-27 09:14:00.202571+05:30" 19

“7819c8c64b829e8cbab8f8fc99f35e5bc1591d68”

“f27e5cf588af76dbf0e2fccf643ffccf7c2242bc”

INTRODUCTION  
Project Title: CampusCatalyst - Student Lifestyle Management System ABSTRACT The ever-evolving demands of student life require an adaptable and comprehensive solution. CampusCatalyst is a student-centric web application meticulously designed to simplify and enrich the lives of college students. In an era where convenience and connectivity are paramount, our platform serves as a reliable companion for students seeking support and information within their campus community. CampusCatalyst offers a multifaceted approach to address common challenges faced by students. It provides a user-friendly interface with a range of essential features tailored to enhance student life. Housing Hub facilitates students in finding suitable accommodation options both for semester-long stays and short-term weekend getaways, bridging the gap between those offering lodging and those in need of it, and promoting a secure and convenient housing ecosystem. Local Insights offers curated recommendations for dining, sports, and other local services near the campus, focusing on personalised suggestions, and ensuring students can explore the best options that cater specifically to their preferences and needs. Transportation Pooling eases the financial burden of fuel expenses and fosters a sustainable approach to commuting. The Date Spot Finder simplifies the task of suggesting ideal locations for romantic outings near the campus, enhancing the overall dating experience. The Event Hub keeps students informed about the latest events happening around campus, encouraging active participation within the college community. It contains different users like admin, students and renters. In conclusion, CampusCatalyst serves as a versatile tool to streamline the daily lives of students, fostering a sense of community and collaboration within the campus. By providing centralized access to housing, local recommendations, transportation, dating, and event discovery, our web application aims to enhance the overall student experience and make college life more enjoyable. this is the project i have to impolement. i want you to remember it. the only difference is, the name CampusCatalyst is changed to 'CamBuzz'

im implementing this in the follwoing:

frontend: flutter

bacjkend: django

database: postgresql.

Testing apis: postman

now i want to focus on django and postgresql, front end will be done later. Now just remember that I am doing using these tools

### 1. User Adds Review from Main Page:

- \*\*Endpoint:\*\* `/addreviewfrommainpage/`

- \*\*Method:\*\* POST

1. The user submits a request to add a review from the main page through the specified API endpoint.

2. The `AddReviewFromMainPage` view handles the request. It validates the incoming data using the `AddReviewFromMainPageSerializer`.

3. The view then extracts information such as restaurant name, location, ratings, recommendations, and additional details from the validated data.

4. It checks if the restaurant already exists. If not, it creates a new restaurant, and if it exists, it updates the existing one.

5. Next, the view creates or updates a recommendation for the user and the specified restaurant.

6. It then calculates and updates the overall price and overall rating for the restaurant based on the latest recommendations.

7. The response indicates the success of the operation (HTTP status 201).

**2.** **User Adds Review for a Specific Restaurant(from the restaurants page**):

- \*Endpoint:\* `/restaurants/<int:restaurant\_id>/add\_review/`

- \*Method:\* POST

1. The user submits a request to add a review for a specific restaurant through the specified API endpoint.

2. The `AddReviewForRestaurant` view handles the request. It validates the incoming data using the `AddReviewFromRestaurantPageSerializer`.

3. The view extracts information such as ratings, recommendations, and additional details from the validated data.

4. It ensures that the specified restaurant exists.

5. The view creates or updates a recommendation for the user and the specified restaurant.

6. It calculates and updates the overall price and overall rating for the restaurant based on the latest recommendations.

7. The response indicates the success of the operation (HTTP status 201).

3. List Top-Rated Restaurants:

- \*Endpoint:\* `/list\_top\_rated\_restaurants/`

- \*Method:\* GET

1. The user sends a request to list the top-rated restaurants, with optional parameters for filtering (name, location, overall\_price).

2. The `ListTopRatedRestaurants` view handles the request. It retrieves and filters restaurants based on the optional parameters.

3. The response includes a serialized list of top-rated restaurants meeting the specified criteria (HTTP status 200).

### 4. Retrieve Restaurant Details with Recommendations:

- \*\*Endpoint:\*\* `/restaurants/<int:id>/`

- \*\*Method:\*\* GET

1. The user sends a request to retrieve details for a specific restaurant, including associated recommendations.

2. The `RestaurantDetailView` handles the request. It retrieves the restaurant and its recommendations.

3. The response includes serialized details of the restaurant along with a list of associated recommendations (HTTP status 200).

### Notes:

- Calculations such as average user rating, average price per head, and overall restaurant statistics are dynamically updated using Django signals and triggers in the PostgreSQL database.

- The flow ensures that user reviews contribute to the overall ratings and statistics of the associated restaurants.

- The API responses include relevant data or messages indicating the success or failure of the operations.

This flow provides a high-level overview of how users can interact with your 'foodrecommendation' app, add reviews for restaurants, retrieve top-rated restaurants, and view detailed information about specific restaurants. If you have any specific questions or if there are particular aspects you'd like to explore further, feel free to ask!

Event Hub:  
we also have another funcionality called "Event Hub" where the different events conducted in the college by different porganisations are properly shown to the users and the students can register through it. THere are 3 actors involved in this:1. Admin, 2. Student, 3. Organisation/club/society this is how this thing kinda works: 1. Orgaaniser/ club/ association \* While the registration process, there are two options available, u can register as an organisation or a student. For organisation, these are the follwoing attriubutes collected: (Name of organisation, username, password, email, about section(small description of the organisation), Website, LinkedIn, Instagram, photo(logo)). When an iorganisation registers, it is successful only if t is approved by the admin. \* After the organiser log in, it is directly redirected into the EventHub page(which is common for student and organiser except "add an event" funcitonality). \* THe organiser can add an event during which the following details are inputted: (Event name, Start Date, End Date, Start time, End time, Location, Registration Link, Details(description), Poster) \* The organiser will be able to edit and deltet this added event also. \* In the organiser profile, every details of the organiser is shown including the "Your events" section which shows all the events that are added by the organiser. \* In the main page of event hub, one organisation can view the events posted by other organisations. 2. Student \* For a student, in the event hub section, everything is same except they cannot add/edit/or delete any event. they can simply view and cliock the registration link. \* They can also click a abutton under any event they see call "I am intereseted". thus one can see how many people are interested in that event(it will not show who all are interested, but only the number)

Certainly! Here's a detailed overview of the "Event Hub" functionality for your "CamBuzz" project:

\*\*Event Hub Functionality Overview:\*\*

\*\*Introduction:\*\*

"CamBuzz" introduces the Event Hub, a centralized platform that efficiently manages and showcases various events organized by college clubs, associations, and societies. This feature facilitates seamless communication between event organizers and students, streamlining event discovery and participation.

\*\*Key Actors:\*\*

1. \*\*Admin:\*\*

- The administrator plays a pivotal role in approving and managing registrations from organizations.

2. \*\*Organisation/Club/Association:\*\*

- Entities interested in hosting events register with specific details and must gain approval from the admin.

- Once approved, organizers can add, edit, and delete events, maintaining an organized profile with event details and a comprehensive list of conducted events.

3. \*\*Student:\*\*

- Students explore a variety of events organized by different entities.

- They have the ability to express interest in an event, providing valuable insights into event popularity without revealing individual identities.

\*\*Organisation/Club/Association Registration:\*\*

- During registration, organizers provide essential details:

- Name of the organization

- Username

- Password

- Email

- About section (a brief description of the organization)

- Website

- LinkedIn

- Instagram

- Photo (logo)

- Admin approval is required for successful registration.

\*\*Organiser Dashboard:\*\*

- Upon login, organizers are redirected to the Event Hub page.

- They can add events with details like event name, start/end date, start/end time, location, registration link, details (description), and a poster.

- The organizer has the ability to edit and delete added events.

- The "Your Events" section in the organizer profile provides a comprehensive list of all events hosted by the organization.

\*\*Student Experience:\*\*

- Students access the Event Hub section, where they can view events hosted by various organizations.

- The interface allows students to click on events to view details and follow the registration link.

- Students can express interest in an event by clicking the "I am Interested" button. The platform aggregates the number of interested students without revealing individual identities.

\*\*Admin Approval:\*\*

- The admin has the authority to approve or reject organization registrations. This step ensures a curated list of trusted organizations.

\*\*Interactivity:\*\*

- The platform encourages interaction between organizers and students, creating a dynamic and engaging environment.

- Students can explore events, and organizers can showcase their offerings, fostering a sense of community and collaboration.

\*\*Benefits:\*\*

- Streamlined event discovery and participation.

- Transparent event details and registration processes.

- Organized profiles for both organizers and students.

- Efficient event management with editing and deletion options.

\*\*Conclusion:\*\*

"CamBuzz" Event Hub transforms event management within the college community. By providing a robust platform for organizations to showcase their events and students to discover and express interest, it fosters a vibrant and interactive campus culture. The streamlined process, coupled with admin oversight, ensures a trustworthy and engaging experience for all users.

\*\*\*\*\*\*\*\*\*EVENT HUB\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

event\_name:Mephestopheles

start\_date:2023-05-15

end\_date:2023-05-15

start\_time:08:00:00

end\_time:10:00:00

location:Chavara Hall

description:An Escape Room

registration\_link:https://example.com/register

name:IEDC

about:Innovation and entrepreneurship development centre in RSET

email:u2109053@rajagiri.edu.in

website\_link:https://rsetiedc.in/

linkedin\_profile\_link:http://www.linkedin.com/in/rsetiedc

instagram\_username:http://www.instagram.com/rsetiedc

facebook:

username:rsetiedc

password:rsetiedc123

name:IEEE

about:We are the official IEEE organisation in RSET

email:u2109053@rajagiri.edu.in

website\_link:https://ieeersetsb.org/

linkedin\_profile\_link:https://www.linkedin.com/company/ieee-student-branch-rajagiri-school-of-engineering-technology/

instagram\_username:http://www.instagram.com/rsetieee

facebook:

username:rsetieee

password:rsetieee123

Certainly! Let's go through the flow of your organization registration and admin approval process, considering the Django models and views you've provided.

1. \*\*User Registration:\*\*

- When a user registers as an organization, a new instance of the `Organisation` model is created.

- The user provides details such as name, email, about, etc.

- The `is\_active` field is set to `False` by default in the `save` method of the `Organisation` model.

2. \*\*Registration Request:\*\*

- Simultaneously, an instance of the `OrganisationRegistrationRequest` model is created, linked to the newly created `Organisation` instance.

- The default status is set to 'Pending' for this registration request.

3. \*\*Admin Approval:\*\*

- In the Django admin interface, the admin can view a list of pending registration requests.

- The admin can either approve or reject a registration request.

- If approved:

- The status of the corresponding `OrganisationRegistrationRequest` is updated to 'Approved'.

- The `is\_active` field of the associated `Organisation` is set to `True`.

- An approval email is sent to the registered organization.

- If rejected:

- The status of the corresponding `OrganisationRegistrationRequest` is updated to 'Rejected'.

- A rejection email is sent to the registered organization.

4. \*\*Email Notifications:\*\*

- The `send\_approval\_email` and `send\_rejection\_email` functions are responsible for sending approval and rejection emails, respectively.

- These functions use Django's `send\_mail` function to send emails.

5. \*\*REST API Endpoint for User Registration:\*\*

- There is a REST API endpoint for organization registration at `/api/organisations/register/`.

- When an organization registers through this API, a new instance of the `Organisation` model is created, and a corresponding `OrganisationRegistrationRequest` is also created.

- The registration request status is checked, and if it's pending or rejected, an appropriate response is sent.

6. \*\*REST API Endpoint for Admin Approval/Rejection:\*\*

- There are REST API endpoints for approving and rejecting registration requests (`/api/organisations/registration\_request/{id}/approve/` and `/api/organisations/registration\_request/{id}/reject/`).

- When the admin approves or rejects a registration request through these endpoints, the corresponding status is updated, and emails are sent as needed.

7. \*\*Error Handling:\*\*

- The code includes error handling to ensure that certain conditions are met before approving or rejecting a registration request.

8. \*\*Django Admin Interface:\*\*

- The Django admin interface is extended to include custom actions for approving and rejecting registration requests.

- The admin can perform these actions directly from the admin interface.

This flow outlines the process from user registration to admin approval or rejection, and how the data in the `Organisation` and `OrganisationRegistrationRequest` tables is updated based on these actions.

To implement a system with three user types (Student, Organisation, Admin), you can follow the model structure below. This involves creating a single `CustomUser` model and additional models for each user type. Each user type will have a one-to-one relationship with the `CustomUser` model. This approach allows you to store common information in the `CustomUser` model while maintaining specific details for each user type.

Here's an example implementation:

1. \*\*models.py in accounts app (or create a new app for user-related functionality):\*\*

```python

from django.contrib.auth.models import AbstractUser

from django.db import models

from django.core.validators import MinValueValidator, MaxValueValidator

from datetime import date

class CustomUser(AbstractUser):

# Common fields for all user types

first\_name = models.CharField(max\_length=30)

email = models.EmailField(unique=True)

username = models.CharField(max\_length=20, unique=True)

# Additional fields for all user types

last\_name = models.CharField(max\_length=30, blank=True, null=True)

# Add other common fields as needed

# Add any common methods as needed

class Student(models.Model):

user = models.OneToOneField(CustomUser, on\_delete=models.CASCADE, primary\_key=True)

# Additional fields specific to students

joining\_year = models.PositiveSmallIntegerField(

validators=[MinValueValidator(2000), MaxValueValidator(date.today().year)],

null=True,

blank=True

)

passout\_year = models.PositiveSmallIntegerField(

validators=[MinValueValidator(2000), MaxValueValidator(date.today().year + 4)]

)

branch\_choices = (

('CSBS', 'CSBS'),

('AI&DS', 'AI&DS'),

# Add other choices as needed

)

branch = models.CharField(max\_length=10, choices=branch\_choices)

# Add other fields specific to students

class Organisation(models.Model):

user = models.OneToOneField(CustomUser, on\_delete=models.CASCADE, primary\_key=True)

# Additional fields specific to organisations

about = models.TextField()

website\_link = models.URLField(blank=True, null=True)

linkedin\_profile\_link = models.URLField(blank=True, null=True)

instagram\_username = models.URLField(blank=True, null=True)

facebook = models.URLField(blank=True, null=True)

# Add other fields specific to organisations

class Admin(models.Model):

user = models.OneToOneField(CustomUser, on\_delete=models.CASCADE, primary\_key=True)

# Additional fields specific to admins

# Add other fields specific to admins

```

2. \*\*Adjust the `AUTH\_USER\_MODEL` in settings.py:\*\*

```python

AUTH\_USER\_MODEL = 'accounts.CustomUser'

```

3. \*\*Run Migrations:\*\*

After making these changes, create and apply migrations to update the database schema:

```bash

python manage.py makemigrations

python manage.py migrate

```

4. \*\*Usage:\*\*

Now you can use the `CustomUser` model for authentication, and each user type (Student, Organisation, Admin) will have its own associated model with specific fields.

For example, to get the student related to a user:

```python

student = Student.objects.get(user=user\_instance)

```

This structure allows you to extend and customize each user type while maintaining a unified authentication system.

i will tell you about my project structure:

The name of the project is 'CamBuzz'. it has the following apps:

1. CamBuzz(main project app which contains the settings.py etc etc)

2. accounts(used to manage different types of accounts)

3. student(to manage student related functioanlities)

4. organisations(to manage oprganisation related functioanliteis)

5. authentication(to manage admin related functioanliteis)

and so on....

as of now, i want to implement registraiton for the two types of users: student and organisations. since they have different attributes and features, i want to do it seperately. so how will i do it?

first\_name:John

last\_name:Doe

email:johndoe@example.com

username:johndoe

password:johndoe123

joining\_year:2022

branch: CSE

division: ALpha

gender: Male

phone\_number:1234567890

{

"user": {

"first\_name": "John",

"last\_name": "Doe",

"email": "john.doe@example.com",

"username": "johndoe",

"password": "securepassword"

},

"joining\_year": 2022,

"branch": "CSE",

"division": "Alpha",

"gender": "Male",

"phone\_number": "1234567890",

"photo": "base64\_encoded\_image\_data"

}

then, i want to implement registration functionality for student user(in DRF).

first\_name:

last\_name:

email:

username:

password:

joining\_year:

branch:

division:

gender:

phone\_number:

photo:

out of this, from first\_name till password is defined in the customuser and the rest is defined in the student model. so implement the registration functionality appropriately.

{

  "first\_name": "John",

  "last\_name": "Doe",

  "email": "johndoe@gmail.com",

  "username": "johndoe",

  "password": "johndoe123",

  "student\_data": {

    "joining\_year": 2022,

    "branch": "CSE",

    "division": "Alpha",

    "phone\_number": "1234567890",

    "gender": "Male"

  }

}

{

  "about": "Innovation and entreienrvno development centre",

  "website\_link": "http://www.rsetiedc.com",

  "linkedin\_profile\_link": "http://www.linkedin.com/rseriedc",

  "instagram\_username": "http://www.linkedin.com/rsetiedc",

  "facebook": "http://www.facebook.com/rsetiedc",

  "user\_data": {

    "first\_name": "IEDC",

    "email": "U2109053@rajagiri.edu.in",

    "username": "rsetiedc",

    "password": "rsetiedc123"

  }

}

{

  "about": "We are the official IEEE organisation in RSET",

  "website\_link": "https://ieeersetsb.org/",

  "linkedin\_profile\_link": "https://www.linkedin.com/company/ieee-student-branch-rajagiri-school-of-engineering-technology/",

  "instagram\_username": "http://www.instagram.com/rsetieee",

  "facebook": "http://www.facebook.com/rsetieee",

  "user\_data": {

    "first\_name": "IEEE",

    "email": "u2109053@rajagiri.edu.in",

    "username": "rsetieee",

    "password": "rsetieee123"

  }

}

::::::::::::::::::::::::RIDE BOOKING:::::::::::::::::::::::::::::::::

I want to implement the 'Book ride' functionality. how should i do that? this is the idea i have in mind:

The 'Book ride' functionality involves several steps from the user initiating a booking request to updating the booking to cancelling the booking etc etc. Here's a high-level overview of the flow and functioning:

1. \*\*User Initiates Booking Request:\*\*

- The user (passenger) selects a ride they want to book.

2. \*\*Check Ride Availability:\*\*

- The system checks if there are available seats for the selected ride.

- If there are no available seats, the booking process is halted, and an appropriate message is returned.

3. \*\*Initiate Booking Request:\*\*

- The user submits a booking request, providing any required information (e.g., pickup/drop-off locations).

4. \*\*Update Ride Information:\*\*

- If the booking request is valid, the system updates the ride information, reducing the number of available seats by 1.

- This update ensures that multiple users cannot book the same seat simultaneously.

5. \*\*Create Booking Record:\*\*

- The system creates a booking record, associating the booked ride and the user making the booking.

6. \*\*Confirmation and Notification:\*\*

- The system confirms the booking and notifies the user and the publisher of the successful booking.

- Notifications could be through the app, email, or other communication channels.

7. \*\*Display Booking Details:\*\*

- The user can view the details of their booked ride, including pickup/drop-off locations, date, time, and any other relevant information.

8. \*\*Manage Bookings:\*\*

- Users (both ride owners and passengers) can view and manage their bookings.

- This includes viewing upcoming bookings, canceling bookings, and any other relevant actions.

9. \*\*Update Ride Status:\*\*

- If needed, the system updates the status of the ride (e.g., marked as booked) to reflect the changes.

10. \*\*Handle Cancellations:\*\*

- If a user cancels a booking, the system updates the ride information to increase the number of available seats.

- Notifications are sent to relevant parties to inform them of the cancellation.

11. \*\*Handle Edge Cases:\*\*

- The system should handle edge cases, such as race conditions where multiple users attempt to book the last seat simultaneously.

- Proper error handling and validation are crucial to ensure data consistency.

Certainly! To implement the updated logic where a passenger can directly book a ride without confirmation from the publisher, you can modify the steps as follows:

### 1. \*\*User Initiates Booking Request:\*\*

- Create a new API endpoint, e.g., `/api/vehiclepooling/book-ride/`.

### 2. \*\*Check Ride Availability:\*\*

- Check if the number of seats required by the passenger is less than or equal to the available seats for the selected ride.

- If the required seats are available, proceed with the booking; otherwise, return a response indicating unavailability.

### 3. \*\*Initiate Booking Request:\*\*

- Accept the booking request with necessary details (e.g., pickup/drop-off locations) in the request payload.

### 4. \*\*Update Ride Information:\*\*

- If the booking is valid, update the ride's available seats by reducing the count.

### 5. \*\*Create Booking Record:\*\*

- Create a new model `Booking` with fields like `ride` (ForeignKey to `VehicleListing`), `passenger` (ForeignKey to `Student`), and any additional information.

### 6. \*\*Confirmation and Notification:\*\*

- Respond with a success message.

- Send notifications to the publisher and the passenger about the successful booking.

- Optionally, send notifications via email or app notifications.

### 7. \*\*Display Booking Details:\*\*

- Create an API endpoint, e.g., `/api/vehiclepooling/my-bookings/`, where users can view details of their booked rides.

### 8. \*\*Manage Bookings:\*\*

- Allow users to view and manage their bookings through `/api/vehiclepooling/my-bookings/`.

- Implement actions like canceling bookings.

### 9. \*\*Update Ride Status:\*\*

- Optionally, update the ride status to reflect the booking status.

### 10. \*\*Handle Cancellations:\*\*

- Implement an API endpoint, e.g., `/api/vehiclepooling/cancel-booking/`, to handle booking cancellations.

- Update the ride information to increase available seats.

- Send notifications about the cancellation.

### 11. \*\*Handle Edge Cases:\*\*

- Implement proper error handling to handle scenarios like race conditions.

- Ensure data consistency with proper validation.

Remember that both passengers and publishers are students, and the logic should be adjusted accordingly in the serializers, views, and models. If you have specific questions or need further clarification on any part of the implementation, feel free to ask!

i have restructures my user models and other functionalites. i will provide you wiht the code and you shuld learn it and understadn it for future references

now i want to mainly implement the vehicle pooling functioanlity for students. i have already creeated many of the functoionalities in vehicle pooling before restructuring the user models. now i will provide you the code of my current vehihcle pooling, and modify it according to the restructured usermodels. mainly, studetns are perorming the functionality