Django Class Notes

Clarusway

Flight Project 3 - Flight App

Nice to have VSCode Extentions:

- Djaneiro - Django Snippets
- SQLite Viewer

Needs

- Python
- pip
- virtualenv
- .gitignore file
- .env file
- Postman

Summary

- Tasks
- Create Flight App
- Models
- Flight
    - Serializer
    - View
    - URL
    - Custom Permission
- Reservation

    - Serializer
    - View
    - URL
    - Nested serializer
    - Filtering against the current user
    - Filter by time

Tasks

- Flights
    - Users can;
        - list upcoming flights
    - Staff members can;
        - list all flights with reservations
        - CRUD flights
- Reservations
    - Logged in users can;
        - create reservation
        - list their own reservations
    - Staff members can;
        - list all reservations
        - CRUD reservations

Create Flight App

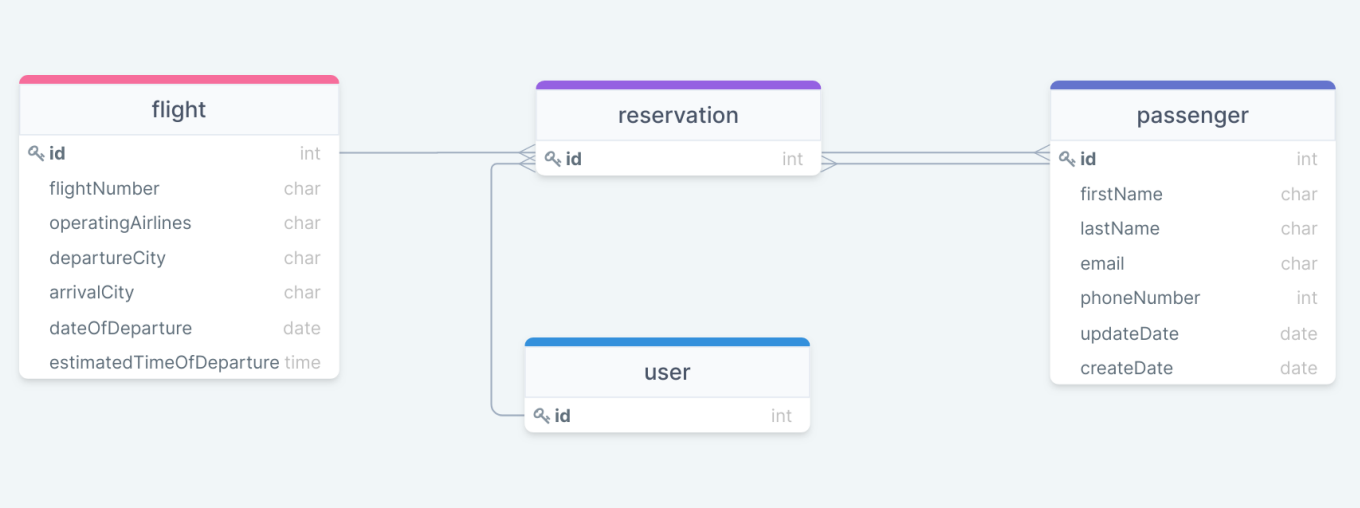
- Create app

```
python manage.py startapp flight
```

- Go to settings/base.py and add under INSTALLED_APPS:

```
# My apps:
'flight',
```

Models



According to the ERD, we will create three models; flight, reservation, and passenger. Also we will use Django's default User model.

```
from django.db import models
from django.contrib.auth.models import User

# Create your models here.
class Flight(models.Model):
    flight_number = models.CharField(max_length=6)
    airlines = models.CharField(max_length=50)
    departure_city = models.CharField(max_length=50)
    arrival_city = models.CharField(max_length=50)
    date_of_departure = models.DateField()
    etd = models.TimeField()

    def __str__(self):
        return self.flight_number + ' from ' + self.departure_city + ' to ' +
self.arrival_city

class Passenger(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(null=True, blank=True)
    phone_number = models.CharField(max_length=15)
    created = models.DateTimeField(auto_now_add = True)

    def __str__(self):
        return f"{self.last_name} - {self.first_name}"

class Reservation(models.Model):
    user = models.ForeignKey(User, on_delete = models.PROTECT)
    passenger = models.ManyToManyField(Passenger,
related_name="reservations_of_passenger")
    flight = models.ForeignKey(Flight, on_delete = models.CASCADE,
related_name="reservations_of_flight")

    def __str__(self):
        return f"{self.flight} {self.user.username}"
```

The `related_name` attribute specifies the name of the reverse relation from the Passerger or Flight model back to your model.

If you don't specify a related_name, Django automatically creates one using the name of your model with the `suffix_set`, for instance `Passenger.reservation_set.all()`.

If you do specify a related name for passenger like `related_name=reservations_of_passenger` on the Reservation model, `Passenger.reservations_of_passenger.all()` will obviously a bit cleaner and less clunky.

So for example, if you had a Passenger object first_name, you could use `first_name.reservations_of_passenger.all()` to get all instances of your Reservation model that have a relation to first_name.

Following relationships “backward”

- Create tables and apply them to the db:

```
python manage.py makemigrations
python manage.py migrate
```

- Register models to admin dashboard:

```
from .models import Passenger, Flight, Reservation

admin.site.register(Passenger)
admin.site.register(Flight)
admin.site.register(Reservation)
```

- Go to admin panel and create a flight for testing.

Flight

Serializer

- Create flight/serializers.py.
- Write a serializer to list flights.

```
from rest_framework import serializers
from .models import Flight

class FlightSerializer(serializers.ModelSerializer):

    class Meta:
        model = Flight
```

```
        # fields = '__all__'
        # exclude = ('airlines',)
        fields = (
            "flight_number",
            "airlines",
            "departure_city",
            "arrival_city",
            "date_of_departure",
            "etd"
        )
```

View

- Create the view:

```
from .serializers import FlightSerializer
from .models import Flight
from rest_framework.viewsets import ModelViewSet

class FlightView(ModelViewSet):
    queryset = Flight.objects.all()
    serializer_class = FlightSerializer
```

URL

- Include flight.urls to main url patterns path('flight/', include('flight.urls')),
- Create flight/urls.py

```
from rest_framework import routers
from .views import FlightView

# Do not forget to put paranthesis at the end of DefaultRouter()!
router = routers.DefaultRouter()
router.register('flights', FlightView)

# urlpatterns = router.urls

urlpatterns = [
    # some paths
]

urlpatterns += router.urls
```

- Test endpoint, list and create http://localhost:8000/flight/flights/
- Currently anybody can do anything in this app. There is no authorization and permission.

- According to our first task, users can list upcoming flights only, but staff can make other operations.
- Search documentation if there is any available permissions for our use case.

Custom Permission

- We need a custom permission to do our task.
- Create flight/permissions.py and a custom permission class.

```
from rest_framework import permissions

class IsAdminOrReadonly(permissions.IsAdminUser):

    def has_permission(self, request, view):

        if request.method in permissions.SAFE_METHODS:
            return True
        else:
            return bool(request.user and request.user.is_staff)
```

- Add this permission to view:

```
from .permissions import IsAdminOrReadonly

permission_classes = [IsAdminOrReadonly]
```

- Test this endpoint, try to make get, post request as a regular user and staff user

Reservation

Serializer

- Continue writing Reservation serializer

```
class ReservationSerializer(serializers.ModelSerializer):

    class Meta:
        model = Reservation
        fields = '__all__'
```

View

- Write views:

```
from .serializers import ReservationSerializer
from .models import Reservation


class ReservationView(ModelViewSet):
    queryset = Reservation.objects.all()
    serializer_class = ReservationSerializer
```

URL

- Add url

```
from .views import ReservationView
router.register('reservations', ReservationView)
```

- Add two passengers and a reservation on admin panel
- Test the endpoint.
- Requirement is to see the username, flight details, and passenger details on this endpoint. So we need to modify our serializer to send these data.

Nested serializer

- First we need to create a passenger serializer and use it inside the reservation serializer. Be careful creating passenger serializer before reservation serializer.
- Passsenger serializer:

```
from .models import Passenger


class PassengerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Passenger
        fields = "__all__"
```

- Modifications on Reservation serializer

```
class ReservationSerializer(serializers.ModelSerializer):

    passenger = PassengerSerializer(many=True, required=True)
    flight = serializers.StringRelatedField()
    user = serializers.StringRelatedField()

    class Meta:
```

```
        model = Reservation
        fields = '__all__'
```

- StringRelatedField() is by default read_only=True, when we want to create a reservation, we have to send flight and user. So we need extra fields to handle creation process. Those will be write_only=True. User will be also required=False because we are getting user info from logged in user.

```
class ReservationSerializer(serializers.ModelSerializer):

    passenger = PassengerSerializer(many=True, required=True)
    flight = serializers.StringRelatedField()
    user = serializers.StringRelatedField()

    flight_id = serializers.IntegerField(write_only=True)
    user_id = serializers.IntegerField(write_only=True, required=False)

    class Meta:
        model = Reservation
        fields = '__all__'
```

- May modify fields

```
    class Meta:
        model = Reservation
        fields = (
            "id",
            "flight",
            'flight_id',
            "user",
            "user_id",
            "passenger"
        )
```

- We need to overwrite the create method of reservation serializer to create passengers when we create reservation.

```
    def create(self, validated_data):
        passenger_data = validated_data.pop('passenger')
        validated_data['user_id'] = self.context.get('request').user.id
        reservation = Reservation.objects.create(**validated_data)

        for passenger in passenger_data:
            new_passenger = Passenger.objects.create(**passenger)
            reservation.passenger.add(new_passenger)

        reservation.save()
        return reservation
```

- Get this endpoint from swagger and test it on postman
- Optional owerwrite update()

Filtering against the current user

- A regular user curently can see all other reservations. We need to restrict that. Users can only see thir own reservations. And staff can see all the reservations. We can do that in Reservation view.

Filtering against the current user

```
class ReservationView(ModelViewSet):
    queryset = Reservation.objects.all()
    serializer_class = ReservationSerializer

    def get_queryset(self):
        """
        This view should return a list of all the reservations
        for the currently authenticated user.
        """
        queryset = super().get_queryset()
        if self.request.user.is_staff:
            return queryset
        else:
            user = self.request.user
            return queryset.filter(user=user)
```

- Test this changes using regular user, and staff
- Test to see the flights.
- We want our staff can see the reservations on flights!!! first create a new serializer and add a conditional to flight view.

```
class StaffFlightSerializer(serializers.ModelSerializer):
    reservations_of_flight = ReservationSerializer(many=True, read_only=True)

    class Meta:
        model = Flight
        fields = (
            "id",
            "flight_number",
            "airlines",
            "departure_city",
            "arrival_city",
            "date_of_departure",
            "etd",
            "reservations_of_flight",
        )
```

- Override get_serializer_class()

```
from .serializers import StaffFlightSerializer

    def get_serializer_class(self):
        serializer = super().get_serializer_class()

        if self.request.user.is_staff:
            return StaffFlightSerializer

        return serializer
```

- Test this feature with regular user and staff

Filter by time

- It is useless to see previous flights while making reservations. So we need to edit flight view again. Instead of getting all the objects, we can filter by time. Staff will see all of them.

```
from datetime import datetime, date
from django.db.models import Q

    def get_queryset(self):
        queryset = super().get_queryset()

        now = datetime.now()
        current_time = now.strftime('%H:%M:%S')
        today = date.today()

        if self.request.user.is_staff:
            return queryset
        else:
            return Flight.objects.filter(Q(date_of_departure__gt=today) |
Q(date_of_departure=today, etd__gt=current_time))
```

☺ **Happy Coding!** ✍