BookMyShow - Models

Classes

In this series, we are going to be creating end to end REST APIs for a movie ticket booking system such as BookMyShow.
We are going to be using `Django REST Framework` for creating the APIs and `MySQL` for the database. More on that later.

Setup

Django REST

A web framework is a library that makes web development faster and easier by providing common patterns for building

reliable, scalable and maintainable applications. Django is a high-level Python Web framework that encourages rapid
development and clean, pragmatic design. Django REST Framework is a powerful and flexible toolkit for building Web APIs.
It provides out of the box support for serializing and deserializing data to and from JSON, validation, authentication, permissions, and much more.

To get started, we will create a virtual environment and install Django REST Framework in it.

```
$ python -m venv bms
$ source bms/bin/activate
$ pip install django djangorestframework
```

MySQL

Before we get started, we need to install MySQL. You can download it from here or use homebrew to install it.

```
$ brew install mysql
```

You also need to install the MySQL client for Python.

```
$ pip install mysqlclient
```

So the complete pip command would be:

```
$ pip install django djangorestframework mysqlclient
```

Create a Django project

Django is structured as a set of applications that are installed into a single project. To create a project, run the following command:

```
$ django-admin startproject book_my_show
```

The above command will create a directory called `book_my_show` and boilerplate code for a Django project inside it.
The folder structure will look like this:

```
book_my_show
│   manage.py
├── book_my_show1
│   │   asgi.py
│   │   __init__.py
│   │   settings.py
│   │   urls.py
│   │   wsgi.py
```

We will be looking at each file in detail later.

Connect to MySQL

We need to create a database in MySQL and connect our Django project to it. To do that, open the MySQL shell:

```
$ mysql -u root -p
```

You will be prompted to enter the password for the root user. Enter the password and you will be inside the MySQL shell.

```
mysql> CREATE DATABASE book_my_show;
```

This will create a database called `book_my_show`. Now, we need to connect our Django project to this database. To do that, open the `settings.py` file inside the `book_my_show` folder and update the `DATABASES` variable as follows:

```
# your_project_name/settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'your_database_name',
        'USER': 'your_database_user',
        'PASSWORD': 'your_database_password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Database migrations

Django comes with a built-in database migration system that lets you create, modify and delete database tables and their fields. To create the initial tables in the database, run the following command:

```
$ python manage.py migrate
```

The above commands will create the tables in the database. You can verify this by opening the MySQL shell and running the following command:

```
mysql> USE book_my_show;
mysql> SHOW TABLES;
```

Run the server

Before we run the server add the following line to the `settings.py` file:

```
# your_project_name/settings.py

INSTALLED_APPS = [
    ...
    'book_my_show',
    'rest_framework',
]
```

Now, run the following command to start the server:

```
$ python manage.py runserver
```

If you open the browser and go to `http://localhost:8000`, you should see a generic Django page.

Models

Let us now create the models for our application.

> **Django ORM**
>
> Django comes with a built-in ORM (Object Relational Mapper) that lets you interact with the database using Python objects. An ORM is a library that lets you interact with a relational database using an object-oriented paradigm. It lets you define classes that map to database tables and lets you create, read, update and delete records in the database using these classes. Django's ORM is called `django.db.models`. We will be using this to create our models.

User

We will start with creating the `User` model. Create a file called `models.py` inside the `book_my_show` folder and add the following code to it:

```
# book_my_show/models.py

from django.db import models

class User(models.Model):
    name = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=255)
```

The above code creates a `User` model with three fields - `name`, `email` and `password`.

- The `name` field is a `CharField` which is used to store strings.
- The `email` field is an `EmailField` which is used to store strings with email validations. The `unique=True` argument specifies that the `email` field should be unique i.e. no two users can have the same email address.
- The `password` field is also a `CharField` which is used to store passwords.

Now, we need to create the tables in the database for this model. To do that, run the following command:

```
$ python manage.py makemigrations book_my_show
$ python manage.py migrate
```

The above commands will create the tables in the database. You can verify this by opening the MySQL shell and running the `SHOW TABLES` command.

Enums

Let's see how to create enums in Django. Create a file called `enums.py` inside the `book_my_show` folder and add the following code to it:

```python
from django.db import models

class PaymentMode(models.TextChoices):
    UPI = "UPI"
    CREDIT_CARD = "CREDIT_CARD"
    NET_BANKING = "NET_BANKING"

class PaymentStatus(models.TextChoices):
    SUCCESS = "SUCCESS"
    FAILED = "FAILED"

class BookingStatus(models.TextChoices):
    BOOKED = "BOOKED"
    CANCELLED = "CANCELLED"

class MovieFeatures(models.TextChoices):
    THREE_D = "3D"
    FOUR_D = "4D"
    IMAX = "IMAX"
    DOLBY = "DOLBY_ATMOS"

class SeatType(models.TextChoices):
    GOLD = "GOLD"
    DIAMOND = "DIAMOND"
    PLATINUM = "PLATINUM"
```

Enums in Django

Django comes with a built-in enum class called `django.db.models.TextChoices`. We will be using this to create our enums.
A sequence consisting itself of iterables of exactly two items `(e.g. [(A, B), (A, B) ...])` to use as choices for this field. If choices are given, they're enforced by model validation.

The first element in each tuple is the actual value to be set on the model, and the second element, if given, is the human-readable name

Structural entities (City, Theater, Hall, Seat)

Let's now create the structural entities. Edit the file called `models.py` inside the `book_my_show` folder and add the following code to it:

```
# book_my_show/models.py

from django.db import models

class City(models.Model):
    name = models.CharField(max_length=255)

class Theater(models.Model):
    name = models.CharField(max_length=255)
    address = models.CharField(max_length=255)
    city = models.ForeignKey(City, on_delete=models.CASCADE)

class Hall(models.Model):
    name = models.CharField(max_length=255)
    theater = models.ForeignKey(Theater, on_delete=models.CASCADE)

class Seat(models.Model):
    number = models.CharField(max_length=255)
    seat_type = models.CharField(choices=SeatType.choices, max_length=50)
    hall = models.ForeignKey(Hall, on_delete=models.CASCADE)
```

The above code creates the `City`, `Theater`, `Hall` and `Seat` models. Let's look at each of them in detail:

- The `City` model has a `name` field which is a `CharField` used to store the name of the city.
- The `Theater` model has the `name` and `address` fields which are `CharField`. It also has a `city` field which is a `ForeignKey` used to store the city in which the theater is located. A `ForeignKey` is used to create a many-to-one relationship between two models. In this case, a city can have many theaters, but a theater can only have one city. It also has a `on_delete=models.CASCADE` argument which specifies that if a city is deleted, all the theaters in that city should also be deleted.

- The `Hall` model has the `name` field which is a `CharField`. It also has a `theater` field which is a `ForeignKey` used to store the theater in which the hall is located. A theater can have many halls, but a hall can only have one theater.
- The `Seat` model has the `number` and `seat_type` fields which are `CharField`. It also has a `hall` field which is a `ForeignKey` used to store the hall in which the seat is located. A hall can have many seats, but a seat can only have one hall.

Movie and Show

Let's now create the `Movie` and `Show` models. Edit the file called `models.py` inside the `book_my_show` folder and add the following code to it:

```
# book_my_show/models.py

from django.db import models

class Language(models.Model):
    name = models.CharField(max_length=50)

class Movie(models.Model):
    name = models.CharField(max_length=255)
    rating = models.IntegerField()
    category = models.CharField(max_length=50)
    languages = models.ManyToManyField(Language)
    features = models.ManyToManyField(MovieFeature)

class Show(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    start_time = models.DateTimeField()
    duration = models.IntegerField()
    language = models.ForeignKey(Language, on_delete=models.CASCADE)
    hall = models.ForeignKey(Hall, on_delete=models.CASCADE)
    features = models.ManyToManyField(MovieFeature)
```

You see a new type of field here - `ManyToManyField`. A `ManyToManyField` is used to create a many-to-many relationship between two models. In this case, a movie can have many languages and a language can have many movies. Similarly, a movie can have many features and a feature can have many movies. This will create a new table in the database called

`book_my_show_movie_languages` and `book_my_show_movie_features` which will store the relationship between movies and languages and movies and features respectively.

ShowSeat

To store the booking status of a seat, we need to create a new model called `ShowSeat`.

```
# book_my_show/models.py

from django.db import models

class ShowSeat(models.Model):
    seat = models.ForeignKey(Seat, on_delete=models.CASCADE)
    show = models.ForeignKey(Show, on_delete=models.CASCADE)
    status = models.CharField(choices=SeatStatus.choices, max_length=50)
```

The `status` field is a `CharField` with choices. The `choices` argument specifies the list of choices for this field. The `choices` argument can be a list of tuples or a list of lists. We are using the enum we created earlier to specify the choices for this field.

Booking and Payment

Let's now create the `Booking` and `Payment` models. Edit the file called `models.py` inside the `book_my_show` folder and add the following code to it:

```python
# book_my_show/models.py

from django.db import models

class Booking(models.Model):
    amount = models.FloatField()
    seats = models.ManyToManyField(ShowSeat)
    show = models.ForeignKey(Show, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    status = models.CharField(choices=BookingStatus.choices)
    booked_at = models.DateTimeField(auto_now_add=True)

class Payment(models.Model):
    amount = models.FloatField()
    mode = models.CharField(choices=PaymentMode.choices)
    status = models.CharField(choices=PaymentStatus.choices)
    booking = models.ForeignKey(Booking, on_delete=models.CASCADE)
```

Finishing up

We have now created all the models for our application. Our models file now looks like:

```python
# book_my_show/models.py
from django.db import models

class User(models.Model):
    name = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=255)


class City(models.Model):
    name = models.CharField(max_length=255)


class Theater(models.Model):
    name = models.CharField(max_length=255)
    address = models.CharField(max_length=255)
    city = models.ForeignKey(City, on_delete=models.CASCADE)


class Hall(models.Model):
    name = models.CharField(max_length=255)
    theater = models.ForeignKey(Theater, on_delete=models.CASCADE)


class Seat(models.Model):
    number = models.CharField(max_length=255)
    seat_type = models.CharField(max_length=255)
    hall = models.ForeignKey(Hall, on_delete=models.CASCADE)


class Language(models.Model):
    name = models.CharField(max_length=50)


class MovieFeatures(models.Model):
    name = models.CharField(max_length=50)


class Movie(models.Model):
```

```
    name = models.CharField(max_length=255)
    rating = models.IntegerField()
    category = models.CharField(max_length=50)
    languages = models.ManyToManyField(Language)
    features = models.ManyToManyField(
        MovieFeatures, limit_choices_to=MovieFeature.choices
    )


class Show(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    start_time = models.DateTimeField()
    duration = models.IntegerField()
    language = models.ForeignKey(Language, on_delete=models.CASCADE)
    hall = models.ForeignKey(Hall, on_delete=models.CASCADE)
    features = models.ManyToManyField(
        MovieFeatures, limit_choices_to=MovieFeature.choices
    )


class ShowSeat(models.Model):
    seat = models.ForeignKey(Seat, on_delete=models.CASCADE)
    show = models.ForeignKey(Show, on_delete=models.CASCADE)
    status = models.CharField(choices=SeatStatus.choices, max_length=50)


class Booking(models.Model):
    amount = models.FloatField()
    seats = models.ManyToManyField(ShowSeat)
    show = models.ForeignKey(Show, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    status = models.CharField(choices=BookingStatus.choices, max_length=50)
    booked_at = models.DateTimeField(auto_now_add=True)


class Payment(models.Model):
    amount = models.FloatField()
    mode = models.CharField(choices=PaymentMode.choices, max_length=50)
    status = models.CharField(choices=PaymentStatus.choices, max_length=50)
```

```
    booking = models.ForeignKey(Booking, on_delete=models.CASCADE)
```

To create the tables in the database, run the following commands:

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

Conclusion

In this session, we created the models for our book my show application using Django ORM. We also looked at how to setup Django and MySQL for our application. In the next session, we will look at how to create the REST APIs for our application using Django REST Framework.