Project Execution Detail

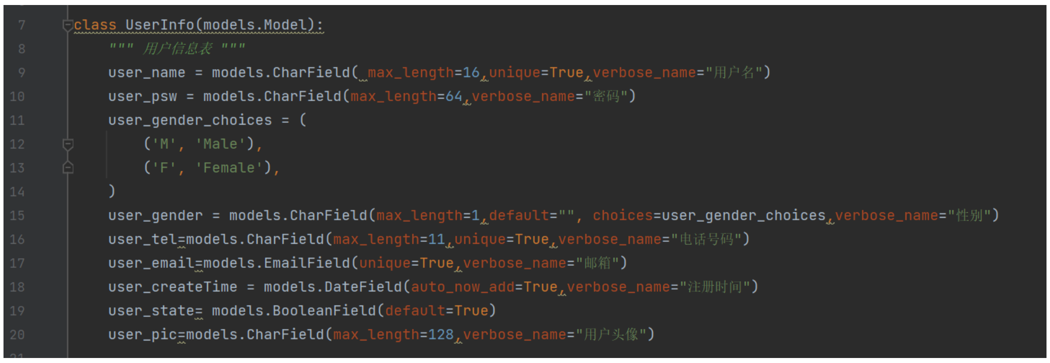
1 Realization of user login and registration module

1.1 Data Model Design

In the user login and registration module of this design, all relevant information of the user is required to be saved. Obviously, a user table User_Info needs to be built first , and the following information needs to be saved in the user table :

- user_id
- user_name
- user_psw
- user_gender
- user_tel
- user_email
- user_createTime
- user_state
- user_pic

The specific definition process of the above fields is as follows:



```
class UserInfo(models.Model):
    """ 用户信息表 """
    user_name = models.CharField( max_length=16,unique=True,verbose_name="用户名")
    user_psw = models.CharField(max_length=64,verbose_name="密码")
    user_gender_choices = (
        ('M', 'Male'),
        ('F', 'Female'),
    )
    user_gender = models.CharField(max_length=1,default="", choices=user_gender_choices,verbose_name="性别")
    user_tel=models.CharField(max_length=11,unique=True,verbose_name="电话号码")
    user_email=models.EmailField(unique=True,verbose_name="邮箱")
    user_createTime = models.DateField(auto_now_add=True,verbose_name="注册时间")
    user_state= models.BooleanField(default=True)
    user_pic=models.CharField(max_length=128,verbose_name="用户头像")
```

Figure 1 User table field definition code map

The specific meanings of the above fields are shown in the table below:

Table 1 Field meaning table of user information table

field name	field meaning	field requirements
user_id	user ID	char(11)
user_name	user name	nvarchar(16)
user_psw	user password	nvarchar(64)
user_gender	user gender	char(1)
user_tel	User phone number	char(11)

user_email	user mailbox	nvarchar(50)
user_createTime	User registration time	datetime(10)
user_state	user status	nvarchar(10)
user_pic	profile picture	char(128)

1.2 Implementation of URL Routing and View

(1) Implementation of URL routing

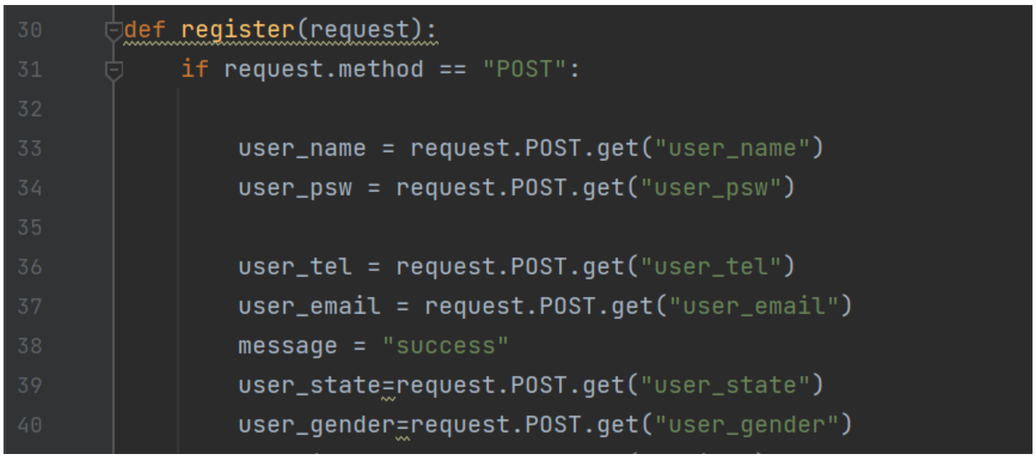
The initial assumption requires the following three URL routes:

Table2 Login registration module url table

URL	view	illustrate
/users /register/	views.register	Register
/users/login/	views.login	Log in
/users/logout/	views. logout	Sign out
/users/ user Info/	views. user_Info	User personal information

(2) Realization of registration view

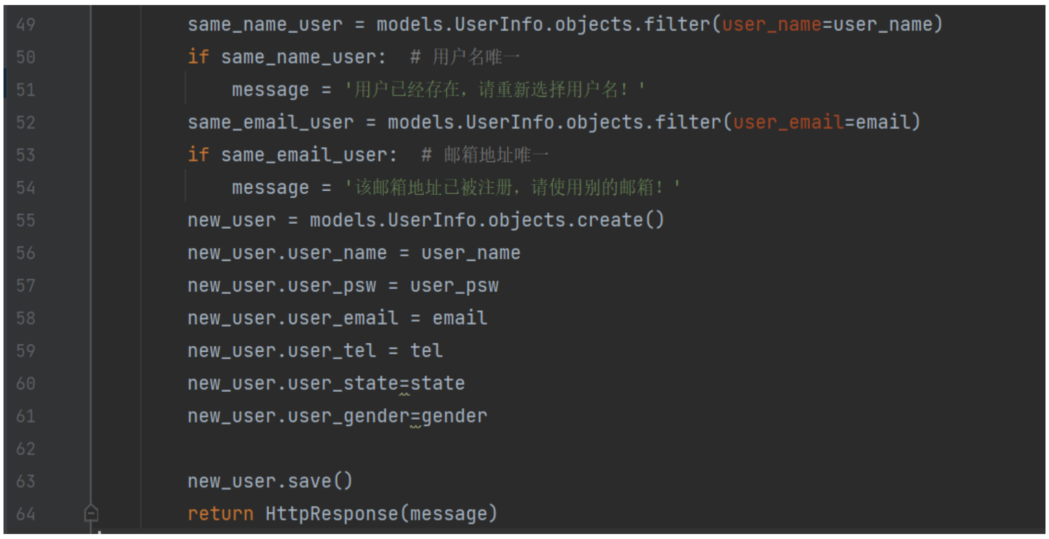
According to the design in the routing, the user fills in the user name , password, gender, mobile phone number, email address, and status through the form on the front-end interface of the APP , and sends it to the /users/register/ address of the server in the form of POST. The server receives and processes this request through the register view function in users/views.py. The specific acceptance process can be realized by the following code.



```
def register(request):
    if request.method == "POST":

        user_name = request.POST.get("user_name")
        user_psw = request.POST.get("user_psw")

        user_tel = request.POST.get("user_tel")
        user_email = request.POST.get("user_email")
        message = "success"
        user_state=request.POST.get("user_state")
        user_gender=request.POST.get("user_gender")
```

Figure 2 POST request processing for registration view

The registration view first uses Django's ORM model to query the database whether there is a username passed from the front end. If there is a match, it will return a username error message. If there is no match, it means that the username does not exist. You can let the user Create that username. Then query the email address and mobile phone number transmitted from the front end in the database one by one. If there is a matching item, an error message will be returned. If there is no matching item, the user will be allowed to create the account, and the relevant user name, password, and password will be saved in the database . Gender, mobile phone number, email address , status. The specific implementation process is as follows:

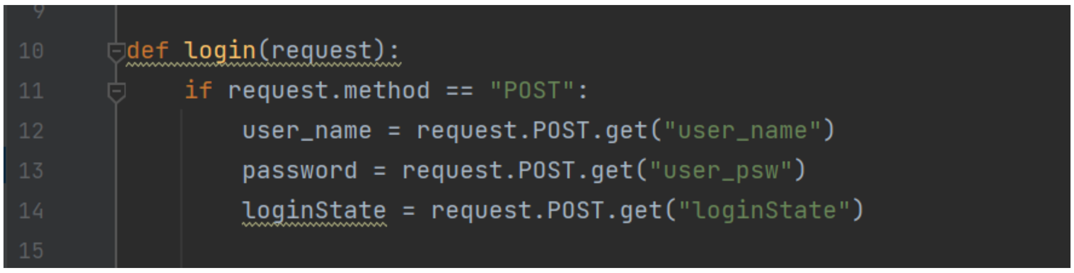


```
        same_name_user = models.UserInfo.objects.filter(user_name=user_name)
        if same_name_user:  # 用户名唯一
            message = '用户已经存在，请重新选择用户名！'
        same_email_user = models.UserInfo.objects.filter(user_email=email)
        if same_email_user:  # 邮箱地址唯一
            message = '该邮箱地址已被注册，请使用别的邮箱！'
        new_user = models.UserInfo.objects.create()
        new_user.user_name = user_name
        new_user.user_psw = user_psw
        new_user.user_email = email
        new_user.user_tel = tel
        new_user.user_state=state
        new_user.user_gender=gender

        new_user.save()
        return HttpResponse(message)
```

Figure 3 User information verification and storage

(3) Realization of login view

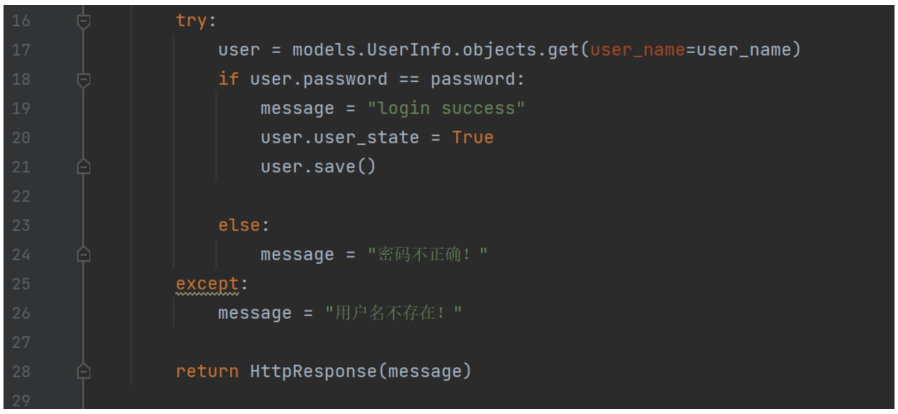
According to the design in the routing, the user fills in the user name and password through the form on the AP front interface, and sends them to the / users /login/ address of the server in the form of POST . The server receives and processes this request through the login view function in users/views.py. The specific acceptance process can be realized by the following code.



```
def login(request):
    if request.method == "POST":
        user_name = request.POST.get("user_name")
        password = request.POST.get("user_psw")
        loginState = request.POST.get("loginState")

```

Figure 4 POST request processing for login view

Django's ORM model to query user data in the database through the uniqueness of the username . If there is a match, the password is compared. If there is no match, the username does not exist. If the password comparison is wrong, the password is incorrect. The specific verification process is as follows:

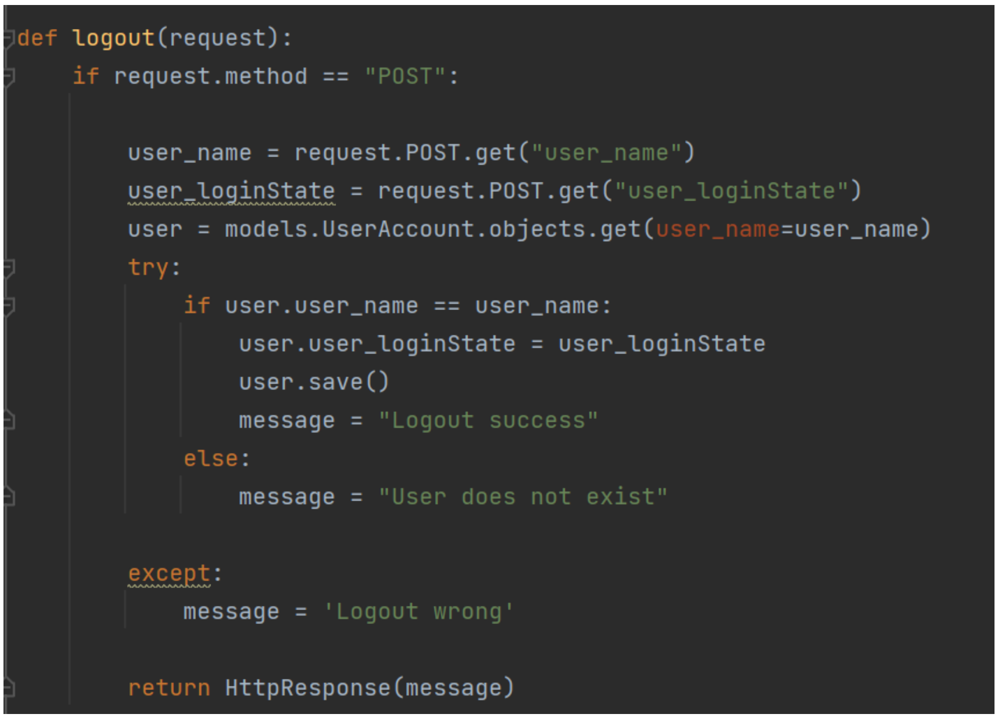


```
        try:
            user = models.UserInfo.objects.get(user_name=user_name)
            if user.password == password:
                message = "login success"
                user.user_state = True
                user.save()

            else:
                message = "密码不正确！"
        except:
            message = "用户名不存在！"

        return HttpResponse(message)
```

Figure 5 Login verification of login view

(4) Logout view design

In this design, whenever a user logs in successfully, the App client will send the online status to the server's / users /login/ address in the form of POST, and the server will save it in the user_state field of the database. Similarly, when the user logs out, the App client will send the offline status to the /users/logout/ address of the server in the form of POST, and the server will save it in the user_state field of the database again.



```
def logout(request):
    if request.method == "POST":

        user_name = request.POST.get("user_name")
        user_loginState = request.POST.get("user_loginState")
        user = models.UserAccount.objects.get(user_name=user_name)
        try:
            if user.user_name == user_name:
                user.user_loginState = user_loginState
                user.save()
                message = "Logout success"
            else:
                message = "User does not exist"

        except:
            message = 'Logout wrong'

        return HttpResponse(message)
```

Figure 6 Logout view

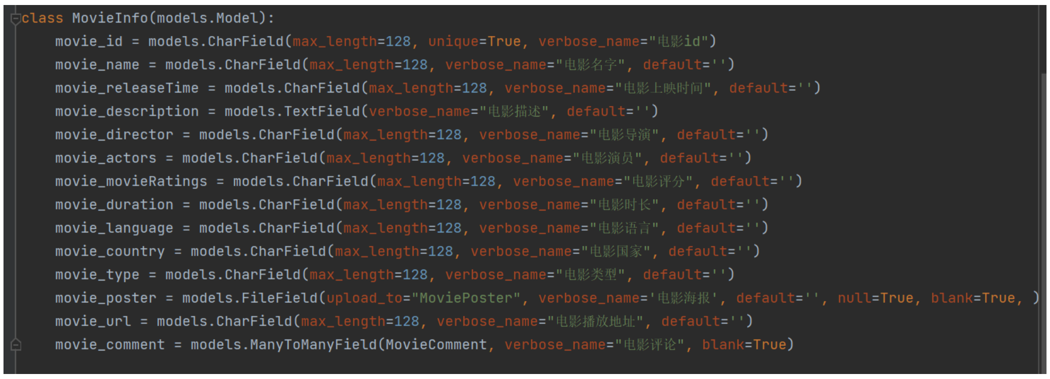
2 Realization of movie information module

2.1 Data Model Design

The information module needs to provide various types of movie information data for the front end, mainly divided into text data and picture data, and the picture data is stored in the database in the form of url. Obviously, we need at least one movie information table movie_Info , and the following information needs to be saved in the information table:

- movie_id
- movie_name
- movie_releaseTime
- movie_description
- movie_director
- movie_actors
- movie_movieRatings
- movie_duration
- movie_language
- movie_country
- movie_type
- movie_poster
- movie_url
- movie_comment

The specific definition process of the above fields is as follows:



```python
class MovieInfo(models.Model):
    movie_id = models.CharField(max_length=128, unique=True, verbose_name="电影id")
    movie_name = models.CharField(max_length=128, verbose_name="电影名字", default='')
    movie_releaseTime = models.CharField(max_length=128, verbose_name="电影上映时间", default='')
    movie_description = models.TextField(verbose_name="电影描述", default='')
    movie_director = models.CharField(max_length=128, verbose_name="电影导演", default='')
    movie_actors = models.CharField(max_length=128, verbose_name="电影演员", default='')
    movie_movieRatings = models.CharField(max_length=128, verbose_name="电影评分", default='')
    movie_duration = models.CharField(max_length=128, verbose_name="电影时长", default='')
    movie_language = models.CharField(max_length=128, verbose_name="电影语言", default='')
    movie_country = models.CharField(max_length=128, verbose_name="电影国家", default='')
    movie_type = models.CharField(max_length=128, verbose_name="电影类型", default='')
    movie_poster = models.FileField(upload_to="MoviePoster", verbose_name='电影海报', default='', null=True, blank=True, )
    movie_url = models.CharField(max_length=128, verbose_name="电影播放地址", default='')
    movie_comment = models.ManyToManyField(MovieComment, verbose_name="电影评论", blank=True)
```

Figure 7 Definition code of each field in the information table

The specific meanings of the above fields are shown in the table below:

Table 3 Field meaning table of movie information table

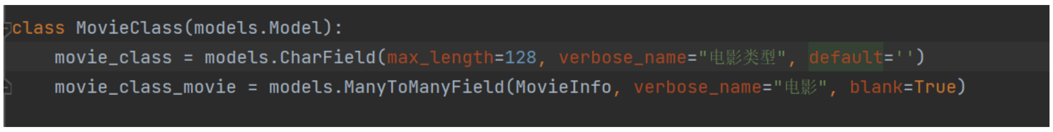
field name	field meaning	field requirements
movie_id	movie number	char(11)
movie_name	movie title	nvarchar(100)

movie_releaseTime	movie showtime	datetime (10)
movie_description	movie description	nvarchar(1024)
movie_director	Movie director	nvarchar(64)
movie_actors	Movie actor	nvarchar(64)
movie_movieRatings	movie rating	nvarchar(10)
movie_duration	movie duration	nvarchar(11)
movie_language	movie language	nvarchar(128)
movie_country	movie country	nvarchar(128)
movie_type	movie type	MoviePoster
movie_poster	movie poster	char(128)
movie_url	movie playback address	char(128)
movie_comment	movie review	Movie Comment

The movie type is saved in the movie information table movie_Info . For the convenience of searching, we create a movie type table MovieClass so that the movie type number in the movie type table corresponds to the movie number in the movie information table. The following information needs to be saved in the movie type table MovieClass:

- movie type
- Movie

The specific definition process of the above fields is as follows:



```
class MovieClass(models.Model):
    movie_class = models.CharField(max_length=128, verbose_name="电影类型", default='')
    movie_class_movie = models.ManyToManyField(MovieInfo, verbose_name="电影", blank=True)
```

Figure 8 Definition codes for each field in the movie type table

2.2 Implementation of URL Routing and View

(1) Implementation of URL routing

The initial assumption requires the following URL routing:

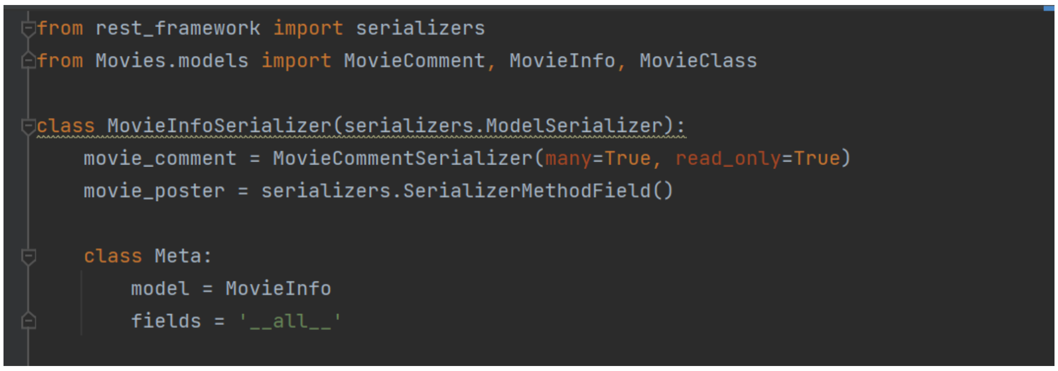
Table 4 Movie information module url table

URL	view	illustrate
/Movies/MovieInfo/	views.MovieInfoView	Provide movie information data for the front end

(2) Implementation of module serialization

In this design, the Django Rest Framework framework will be used to complete the serialization of the model.

serializers.py file under the application folder , and add the following code to complete the serialization of the model.



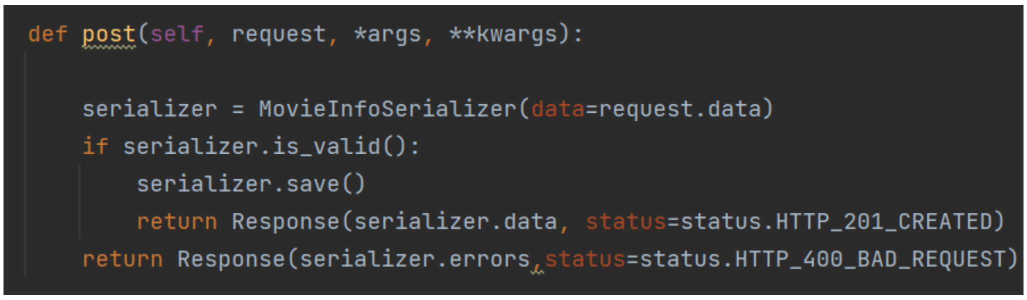
```
from rest_framework import serializers
from Movies.models import MovieComment, MovieInfo, MovieClass

class MovieInfoSerializer(serializers.ModelSerializer):
    movie_comment = MovieCommentSerializer(many=True, read_only=True)
    movie_poster = serializers.SerializerMethodField()

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

Figure 9 Model serialization

In the future development process, you only need to modify the value of the model in the above code and replace it with the data model that needs to be serialized to complete the serialization of the model.

(3) Realization of information view

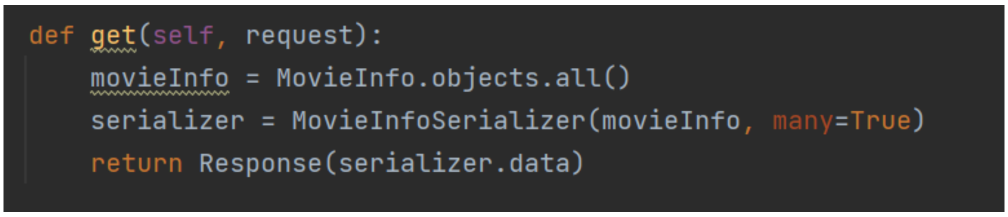
According to the design in the route, the administrator can add relevant movie information through the Admin background, and the background will send a POST request to the server's /Movies/MovieInfo/ address, and the server will first verify the rationality of the received data after accepting the request. When the verification is successful, the server will save the data to the database. The specific code is as follows:



```
def post(self, request, *args, **kwargs):

    serializer = MovieInfoSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors,status=status.HTTP_400_BAD_REQUEST)
```

Figure 10 Data Validation

Similarly, when the front-end information interface sends a GET request to the server , the server will transmit the information data in the database to the front-end interface in the form of json. The specific code is as follows:



```
def get(self, request):
    movieInfo = MovieInfo.objects.all()
    serializer = MovieInfoSerializer(movieInfo, many=True)
    return Response(serializer.data)
```

Figure 11 get request processing of information view

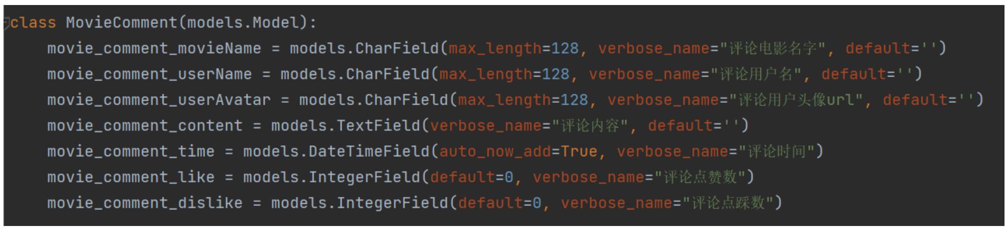
3 Realization of movie review function

3.1 Data Model Design

The movie comment function needs to provide the existing comments for the front end, and transfer the newly added comments to the back end to store in the data, mainly text data. Therefore, a movie comment table MovieComment is required, and in the information table , it needs Save the following information:

- movie_comment_movieName
- movie_comment_userName
- movie_comment_userAvatar
- movie_comment_content
- movie_comment_time
- movie_comment_like
- movie_comment_dislike

The specific definition process of the above fields is as follows:



```
class MovieComment(models.Model):
    movie_comment_movieName = models.CharField(max_length=128, verbose_name="评论电影名字", default='')
    movie_comment_userName = models.CharField(max_length=128, verbose_name="评论用户名", default='')
    movie_comment_userAvatar = models.CharField(max_length=128, verbose_name="评论用户头像url", default='')
    movie_comment_content = models.TextField(verbose_name="评论内容", default='')
    movie_comment_time = models.DateTimeField(auto_now_add=True, verbose_name="评论时间")
    movie_comment_like = models.IntegerField(default=0, verbose_name="评论点赞数")
    movie_comment_dislike = models.IntegerField(default=0, verbose_name="评论点踩数")
```

Figure 12 Definition codes for each field in the movie review table

The specific meanings of the above fields are shown in the table below:

Table 5 Field meaning table of movie review table

field name	field meaning	field requirements
movie_comment_movieName	review movie name	char(128)
movie_comment_userName	comment username	char(128)
movie_comment_userAvatar	review movie name	char(128)
movie_comment_content	comments	char(128)
movie_comment_time	comment time	datetime (10)
movie_comment_like	Comment likes	none
movie_comment_dislike	Comments likes	none

5.3.2 Implementation of URL Routing and View

(1) Implementation of URL routing

The initial assumption requires the following URL routing:

Table 6 Movie review module url table

URL	view	illustrate
/Movies/addMovieComment/	views.addMovieComment /	Provide movie review data for the front end

(2) Realization of movie review view

According to the design in the routing, the user fills in and submits the comment content in the movie comment area through the front-end interface of the APP , and then the system will send the comment information to the review routing address of the view in the form of POST . The server will receive and process this request through the view function, and store the evaluation content transmitted from the front-end interface in the database. At the same time, the front end will also send a GET request to the back end to obtain the current comment list and display it.

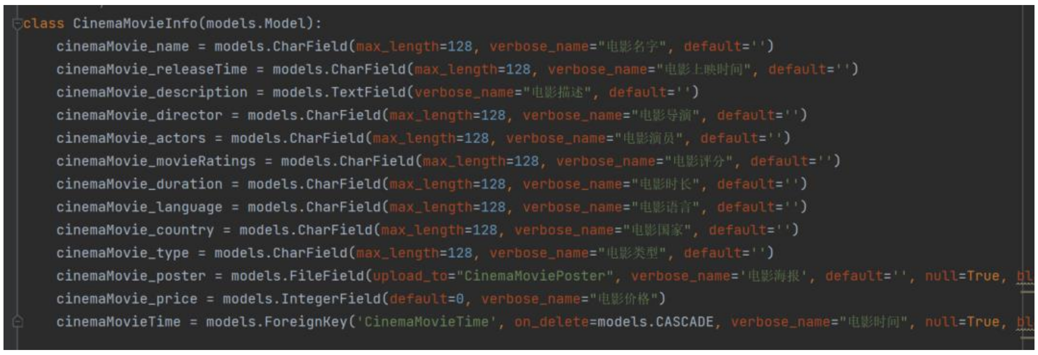
4 Realization of cinema movie ticket purchase module

4.1 Data Model Design

The theater movie ticket purchase module needs to provide various theater movie information data and ticket purchase data for the front end, mainly divided into text data and picture data, and the picture data is stored in the database in the form of url. Obviously, we need at least one theater movie information table CinemaMovie _Info, and the following information needs to be saved in the information table:

- cinemaMovie_name
- cinemaMovie_releaseTime
- cinemaMovie_description
- cinemaMovie_director
- cinemaMovie_actors
- cinemaMovie_movieRatings
- cinemaMovie_duration
- cinemaMovie_language
- cinemaMovie_country
- cinemaMovie_type
- cinemaMovie_poster
- cinemaMovie_price
- cinemaMovieTime

The specific definition process of the above fields is as follows:



```python
class CinemaMovieInfo(models.Model):
    cinemaMovie_name = models.CharField(max_length=128, verbose_name="电影名字", default='')
    cinemaMovie_releaseTime = models.CharField(max_length=128, verbose_name="电影上映时间", default='')
    cinemaMovie_description = models.TextField(verbose_name="电影描述", default='')
    cinemaMovie_director = models.CharField(max_length=128, verbose_name="电影导演", default='')
    cinemaMovie_actors = models.CharField(max_length=128, verbose_name="电影演员", default='')
    cinemaMovie_movieRatings = models.CharField(max_length=128, verbose_name="电影评分", default='')
    cinemaMovie_duration = models.CharField(max_length=128, verbose_name="电影时长", default='')
    cinemaMovie_language = models.CharField(max_length=128, verbose_name="电影语言", default='')
    cinemaMovie_country = models.CharField(max_length=128, verbose_name="电影国家", default='')
    cinemaMovie_type = models.CharField(max_length=128, verbose_name="电影类型", default='')
    cinemaMovie_poster = models.FileField(upload_to="CinemaMoviePoster", verbose_name='电影海报', default='', null=True, bl
    cinemaMovie_price = models.IntegerField(default=0, verbose_name="电影价格")
    cinemaMovieTime = models.ForeignKey('CinemaMovieTime', on_delete=models.CASCADE, verbose_name="电影时间", null=True, bl
```

Figure 13 Definition code of each field in the information table

The specific meanings of the above fields are shown in the table below:

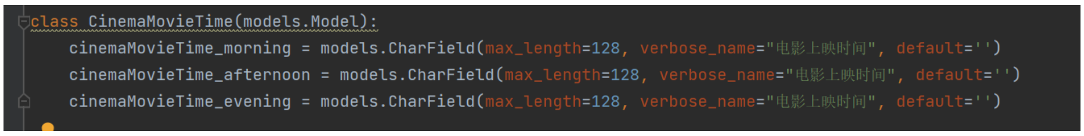
Table 7 Field meaning table of movie information table

field name	field meaning	field requirements
cinemaMovie_name	movie name	nvarchar(100)
cinemaMovie_releaseTime	movie showtime	datetime (10)
cinemaMovie_description	movie description	nvarchar(1024)
cinemaMovie_director	Movie director	nvarchar(64)
cinemaMovie_actors	Movie actor	nvarchar(64)
cinemaMovie_movieRatings	movie rating	nvarchar(10)
cinemaMovie_duration	movie duration	nvarchar(11)
cinemaMovie_language	movie language	nvarchar(128)
cinemaMovie_country	movie country	nvarchar(128)
cinemaMovie_type	movie type	Cinema Movie Poster
cinemaMovie_poster	movie poster	char(128)
cinemaMovie_price	movie price	decima(10,2)
cinemaMovieTime	movie time	CinemaMovieTime

The ticket purchase function of theater movies should provide a movie time schedule. To this end, we need to create a movie schedule CinemaMovieTime . To simplify the data table, we assume that there are 3 options for movie times in theaters, namely morning and afternoon. And at night, the following information needs to be saved in the information table:

- cinemaMovieTime_morning
- cinemaMovieTime_afterning
- cinemaMovieTime_evening

The specific definition process of the above fields is as follows:



```python
class CinemaMovieTime(models.Model):
    cinemaMovieTime_morning = models.CharField(max_length=128, verbose_name="电影上映时间", default='')
    cinemaMovieTime_afternoon = models.CharField(max_length=128, verbose_name="电影上映时间", default='')
    cinemaMovieTime_evening = models.CharField(max_length=128, verbose_name="电影上映时间", default='')
```

Figure 14 Definition codes for each field of the cinema movie schedule

The specific meanings of the above fields are shown in the table below:

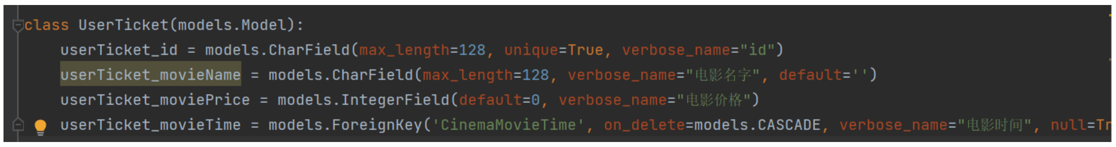
Table 8 Field meaning table of ticket purchase information table

field name	field meaning	field requirements
cinemaMovieTime_morning	Movie show time in the morning	datetime (10)
cinemaMovieTime_afternoon	Movie show time in the afternoon	datetime (10)
cinemaMovieTime_evening	movie showtime evening	datetime (10)

In order to realize the user's ticket purchase function for cinema movies, we need to create a user ticket information table user Ticket , and the following information needs to be saved in the information table:

- userTicket_id
- userTicket_movieName
- userTicket_moviePrice
- userTicket_movieTime

The specific definition process of the above fields is as follows:



```
class UserTicket(models.Model):
    userTicket_id = models.CharField(max_length=128, unique=True, verbose_name="id")
    userTicket_movieName = models.CharField(max_length=128, verbose_name="电影名字", default='')
    userTicket_moviePrice = models.IntegerField(default=0, verbose_name="电影价格")
    userTicket_movieTime = models.ForeignKey('CinemaMovieTime', on_delete=models.CASCADE, verbose_name="电影时间", null=Tr
```

Figure 15 Definition codes for each field in the ticket purchase information table

The specific meanings of the above fields are shown in the table below:

Table 9 Field meaning table of ticket purchase information table

field name	field meaning	field requirements
userTicket_id	Ticket information number	char(11)
userTicket_movieName	movie name	nvarchar(100)
userTicket_moviePrice	movie price	d ecimal(10)
userTicket_movieTime	movie showtime	datetime (10)

5. 4.2 Implementation of URL Routing and View

(1) Implementation of URL routing

The initial assumption requires the following URL routing:

Table 10 url table of cinema movie ticket purchase module

URL	view	illustrate
/ Cinema Movie/cinema Movie Info/	views. cinemaMovieInfo/	Provide theater movie information data for the front end

(2) Realization of movie ticket purchase function in theaters

According to the design in the routing, the user selects the movie and the viewing time period in the movie ticket purchase area through the front-end interface of the APP . After the user completes the ticket purchase operation, the system will send the ticket purchase information to the ticket purchase routing address in the form of POST . The server will receive and process this request through the view function, store the ticket purchase information transmitted from the front-end interface in the database, and at the same time, the front-end will also send a GET request to the back-end to obtain the current ticket purchase information and display it to the user .

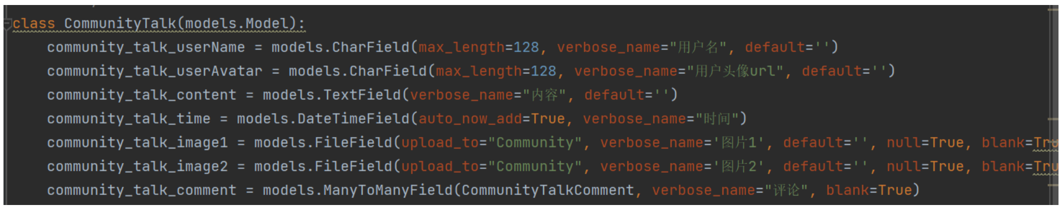
5 Realization of Movie Community Module

5.1 Data Model Design

The movie community module mainly realizes the functions of users posting and following posts. It needs to provide the front-end with posted data, and save the newly posted content of users, including text data and picture data, to the database and display them on the front-end. The picture data is in the URL The form is saved in the database. Obviously, we need at least one community communication table CommunityTalk , and the following information needs to be saved in the community communication table CommunityTalk:

- community_talk_userName
- community_talk_userAvatar
- community_talk_content
- community_talk_time
- community_talk_image1
- community_talk_image2
- community_talk_comment

The specific definition process of the above fields is as follows:



```
class CommunityTalk(models.Model):
    community_talk_userName = models.CharField(max_length=128, verbose_name="用户名", default='')
    community_talk_userAvatar = models.CharField(max_length=128, verbose_name="用户头像url", default='')
    community_talk_content = models.TextField(verbose_name="内容", default='')
    community_talk_time = models.DateTimeField(auto_now_add=True, verbose_name="时间")
    community_talk_image1 = models.FileField(upload_to="Community", verbose_name='图片1', default='', null=True, blank=Tru
    community_talk_image2 = models.FileField(upload_to="Community", verbose_name='图片2', default='', null=True, blank=Tru
    community_talk_comment = models.ManyToManyField(CommunityTalkComment, verbose_name="评论", blank=True)
```

Figure 16 Definition codes for each field in the community communication table

The specific meanings of the above fields are shown in the table below:

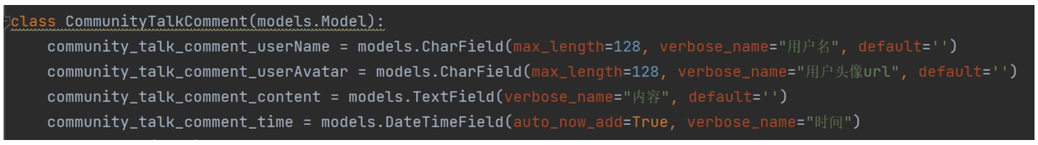
Table 11 Field meaning table of community communication table

field name	field meaning	field requirements
community_talk_userName	username	char(11)
community_talk_userAvatar	user avatar url	nvarchar(128)
community_talk_content	content	nvarchar(100)
community_talk_time	time	datetime (10)
community_talk_image1	Picture 1	char (128)
community_talk_image2	picture 2	char (128)
community_talk_comment	Comment	CommunityTalkComment

the community communication form CommunityTalk , after posting a post, users are allowed to post more after this post, so we need a comment form to save the content that users post more. We have created a community communication comment form CommunityTalkComment to save the following information:

- community_talk_comment_userName
- community_talk_comment_userAvatar
- community_talk_comment_content
- community_talk_comment_time

The specific definition process of the above fields is as follows:



```
class CommunityTalkComment(models.Model):
    community_talk_comment_userName = models.CharField(max_length=128, verbose_name="用户名", default='')
    community_talk_comment_userAvatar = models.CharField(max_length=128, verbose_name="用户头像url", default='')
    community_talk_comment_content = models.TextField(verbose_name="内容", default='')
    community_talk_comment_time = models.DateTimeField(auto_now_add=True, verbose_name="时间")
```

Figure 17 Definition codes for each field in the community exchange comment form

The specific meanings of the above fields are shown in the table below:

Table 12 Field meaning table of community exchange comment table

field name	field meaning	field requirements
community_talk_comment_userName	username	char(11)
community_talk_comment_userAvatar	user avatar url	nvarchar(128)
community_talk_comment_content	content	nvarchar(100)
community_talk_comment_time	time	datetime (10)

5.2 Implementation of URL Routing and View

(1) Implementation of URL routing

The initial assumption requires the following URL routing:

Table 13 Movie community module url table

URL	view	illustrate
/MovieCommunity/communityTalk/	views.MovieCommunityView/	Provide community data for the front end
/MovieCommunity/ add CommunityComment/	views.add CommunityComment/	Provide more content for the front end

(2) Realization of the functions of the movie community module

According to the design in routing, users post or follow up in the movie community through the front-end interface of the APP . After the user completes the posting or following operation, the system will send the content of the post bar to the ticket purchase routing address in the form of POST . The server will receive it through the view function, and store the content data transmitted from the front-end interface in the corresponding database. At the same time, the front-end will also send a GET request to the back-end to obtain the latest post bar content and display it to the user .

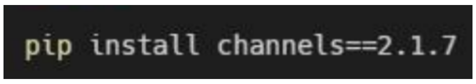
6 Realization of community communication function

The community communication function aims to provide users with an online chat place. In this design, the WebSocket protocol is used to realize the community online chat communication function, but Django itself does not support the WebSocket protocol, so the Channels framework is used to implement WebSocket.

6.1 Environment configuration

(1) Install Channels

in the system cmd to install.

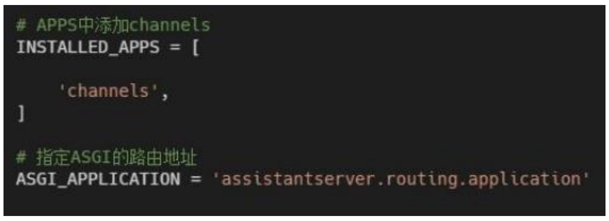


```
pip install channels==2.1.7
```

Figure 18 channels installation instructions

(2) Modify the settings.py file

First add channels to INSTALLED_APP in the settings.py file, and then specify the path address of ASGI. Among them, ASGI_APPLICATION specifies that the location of the main route is the chat application in the routing.py file under assistantserver. The specific code is as follows:

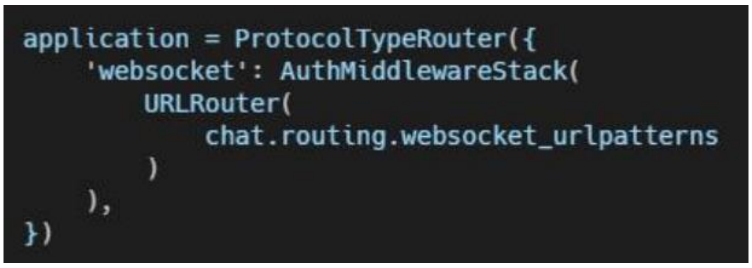


```
# APPS中添加channels
INSTALLED_APPS = [

    'channels',
]

# 指定ASGI的路由地址
ASGI_APPLICATION = 'assistantserver.routing.application'
```

Figure 19 Register channels

(3) Specify the route of the websocket protocol

Create a routing.py routing file in the same directory as setting.py. routing.py is similar to url.py in Django, indicating the routing of the websocket protocol. The specific code is as follows:



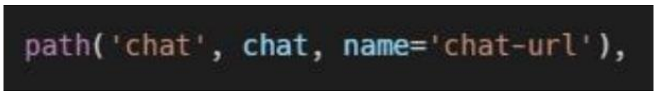
```
application = ProtocolTypeRouter({
    'websocket': AuthMiddlewareStack(
        URLRouter(
            chat.routing.websocket_urlpatterns
        )
    ),
})
```

Figure 20 Specify the route of the websocket protocol

6.2 Construction of Websocket server

(1) Use the python manage.py startapp chat command to create a new application named chat and register it.

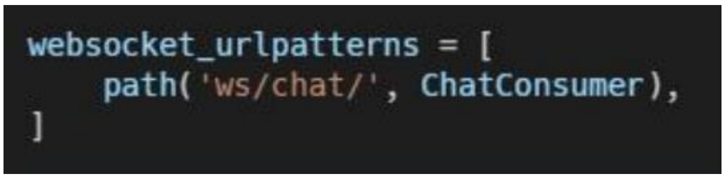
(2) Add the following routes to the urls.py file in the project directory.



```
path('chat', chat, name='chat-url'),
```

Figure 21 Register route

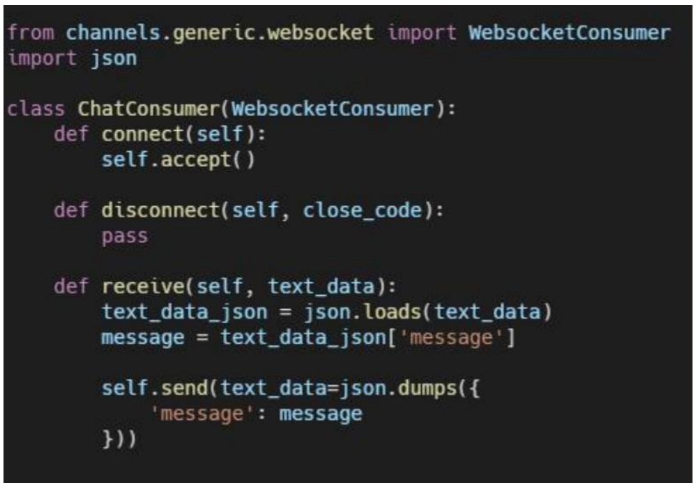
(3) Create a new routing.py file in the chat application directory, and add the following code:



```
websocket_urlpatterns = [
    path('ws/chat/', ChatConsumer),
]
```

Figure 22 websocket routing

(4) Create a new consumer.py file in the chat application directory, and add the following code:



```python
from channels.generic.websocket import WebsocketConsumer
import json

class ChatConsumer(WebsocketConsumer):
    def connect(self):
        self.accept()

    def disconnect(self, close_code):
        pass

    def receive(self, text_data):
        text_data_json = json.loads(text_data)
        message = text_data_json['message']

        self.send(text_data=json.dumps({
            'message': message
        }))
```

Figure 23 consumer.py code

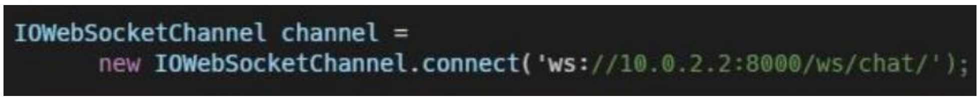
In the ChatConsumer class, the connect method is triggered when the connection is established, the disconnect is triggered when the connection is closed, and the receive method is triggered after receiving a message.

6.3 Construction of Websocket server

The community online chat communication function is divided into the following four steps:

- Connect to a WebSocket server
- Listen for messages from the server
- send data to server
- Close the WebSocket connection

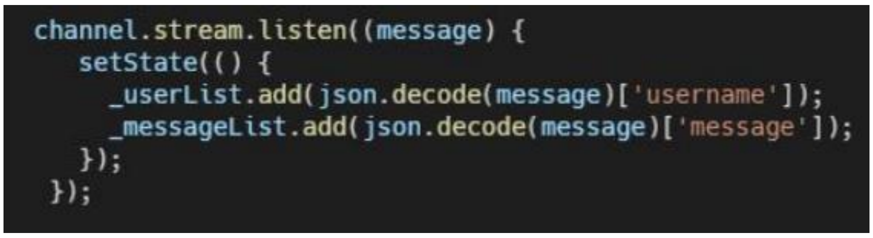
(1) Connect to the WebSocket server
In Flutter, the web_socket_channel plug-in can provide a WebSocketChannel method that allows us to both listen to messages from the server and send messages to the server. Therefore, here you can first create a WebSocketChannel to connect to a server. code show as below:



```dart
IOWebSocketChannel channel =
      new IOWebSocketChannel.connect('ws://10.0.2.2:8000/ws/chat/');
```

Figure 24 WebSocket server address

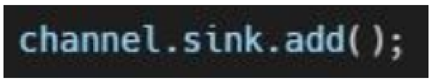
(2) Listen for messages from the server
When the connection has been established with the server, you can listen for messages from the server, and after sending a message to the server, it will return the same message. In order to receive and display these messages, this design will use a channel.stream.listen() method in the init method to monitor the messages, save them in a list, and finally display them in the Text component. code show as below:



```
channel.stream.listen((message) {
   setState(() {
     _userList.add(json.decode(message)['username']);
     _messageList.add(json.decode(message)['message']);
   });
 });
```

Figure 25 flutter listener message code

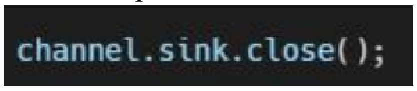
(3) Send the data to the server
In order to send data to the server, this design will use the sink method provided in WebSocketChannel to add messages. code show as below:



```
channel.sink.add();
```

Figure26 Send message code

(4) Close the WebSocket connection
When finished using WebSocket, the connection should be closed. Here you need to add the following code in the dispose method :



```
channel.sink.close();
```

Figure 27 Close websocket server connection

7 Realization of movie search function

The movie search function is realized based on the movie information function. Users search for relevant movie information in the movie information through the front-end interface of the APP . The front-end encapsulates the content of the search information into json format and sends it to the back-end in the form of POST. Compare the database data in the database, query all the movie names that meet the conditions according to the keywords, and package the corresponding movie content into json format and return it to the front end, and the user can query all the movies that meet the search conditions in the front end Movie, which realizes the movie search function.

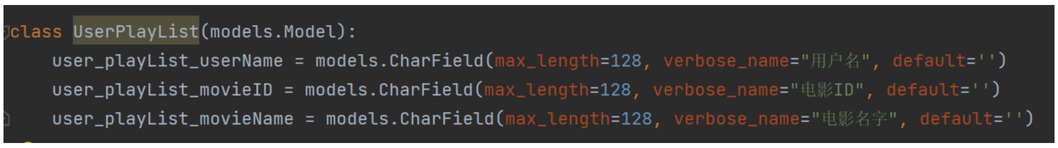
8 Realization of User Playlist Function

8.1 Data Model Design

The user playlist function is designed to create a playlist for the user. It not only needs to provide the front-end with the movies that the user has added to the playlist, but also needs to transfer the newly added movie information to the backend to store in the data. Therefore, it needs a A user playlist UserPlayList , the following information needs to be saved in the user playlist :

- user_playList_userName
- user_playList_movieID
- user_playList_movie Name

The specific definition process of the above fields is as follows:



```python
class UserPlayList(models.Model):
    user_playList_userName = models.CharField(max_length=128, verbose_name="用户名", default='')
    user_playList_movieID = models.CharField(max_length=128, verbose_name="电影ID", default='')
    user_playList_movieName = models.CharField(max_length=128, verbose_name="电影名字", default='')
```

Figure 28 Definition codes for each field in the user playlist

The specific meanings of the above fields are shown in the table below :

Table 14 User playlist field meaning table

field name	field meaning	field requirements
user_playList_userName	username	char(11)
user_playList_movieID	movie number	char(11)
user_playList_movieName	movie name	nvarchar(100)

8.2 Implementation of URL Routing and View

(1) Implementation of URL routing

The initial assumption requires the following URL routing:

Table 15 URL table of user playlist

URL	view	illustrate
/ users /playList/	views.playList/	Provide user playlist data to the front end

(2) Realization of user playlist function

According to the design in the routing, the user selects the movie he wants to watch in the movie information area through the front-end interface of the APP and adds it to the playlist. After completing this operation, the system will send the movie information to the routing address of the playlist in the form of POST . The server will receive it through the view function, and store the data transmitted from the front-end interface in the database of the user's playlist, and at the same time, the front-end will also send a GET request to the back-end to obtain the current latest playlist information and display it .

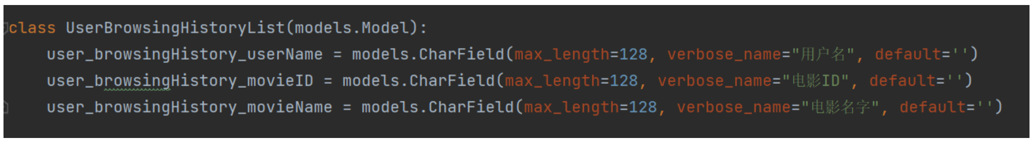
9 Realization of User Browsing History Function

9.1 Data Model Design

The user's browsing history function is designed to save the historical information of the user's watching movies. It is necessary to provide the user's browsing history data for the front end. Every time the user watches the movie, it will be saved in this database. Therefore, we need a user's browsing history data table User BrowsingHistoryList , the following information needs to be saved in the user browsing history data table :

- user_browsingHistory_userName
- user_browsingHistory_movieID
- user_browsingHistory_movieName

The specific definition process of the above fields is as follows:



```
class UserBrowsingHistoryList(models.Model):
    user_browsingHistory_userName = models.CharField(max_length=128, verbose_name="用户名", default='')
    user_browsingHistory_movieID = models.CharField(max_length=128, verbose_name="电影ID", default='')
    user_browsingHistory_movieName = models.CharField(max_length=128, verbose_name="电影名字", default='')
```

Figure 29 Definition codes for each field in the user browsing history data table

The specific meanings of the above fields are shown in the table below:

Table 16 Field meaning table of user browsing history data table

field name	field meaning	field requirements
user_browsingHistory_userName	username	char(11)
user_browsingHistory_movieID	movie number	char(11)
user_browsingHistory_movieName	movie name	nvarchar(100)

9.2 Implementation of URL Routing and View

(1) Implementation of URL routing

The initial assumption requires the following URL routing:

Table 17 URL table of user browsing history list

URL	view	illustrate
/ users /browsingHistory/	views.browsingHistory/	Provide user browsing history data for the front end

(2) Realization of user browsing history function

According to the design in the routing, the movie information watched by the user through the APP front-end interface is added to the browsing history list. Every time a user watches a movie, the system will send the movie information to the routing address of the browsing history list in the form of POST . The server will receive it through the view function, and store the movie data transmitted from the front-end interface in the database of the user's browsing history list. At the same time, the front-end will also send a GET request to the back-end to obtain the current latest browsing history list and display it .

10 Realization of daily recommendation function

The daily recommendation function aims to recommend movies worth watching for users. Putting this recommendation information on the APP homepage can attract users' attention and promote movies better. In this design, the daily recommendation function is pushed by the administrator to a movie every day, and the specific information of the movie is stored in the movie information table.

This feature is yet to be improved in the future.

11 Design Style and Philosophy

In the design of "Enchanted Movie", we draw inspiration from the enchantment and mystique of the cinematic world, encapsulating its rich diversity through color. Indigo and pink have been selected as our primary colors, serving respectively as the themes for light and dark modes based on user preference. The depth and mystery of indigo are fully expressed in the light mode, while in the dark mode, the warmth and intimacy of pink stand out against the contrast.

We chose Apple's SF Symbols for our icon design, ensuring not only consistency with the iOS platform, but also emphasizing our commitment to clean and efficient design principles. SF Symbols are widely incorporated in our navigation bars, menus, and a series of action buttons such as 'favorites' and 'share'.

In our interface design, we deliberately selected ultra-thin material as the primary style. Its high transparency allows background content to show through the foreground, enhancing the three-dimensional feel of the interface. At the same time, it provides users with a light and premium visual experience.

In essence, our design philosophy is about finding the perfect balance between elegance and simplicity. We hope that the design of "Enchanted Movie" can evoke the magical feelings associated with cinema while also delivering a smooth and intuitive user experience.

12 Key SwiftUI Components and Techniques

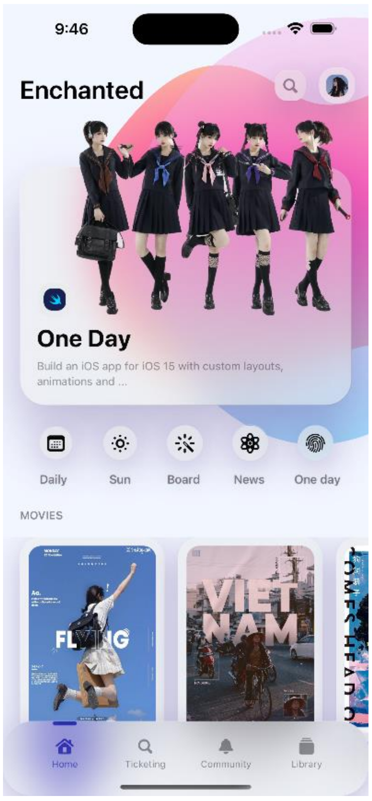
In the development of "Enchanted Movie", we utilize NavigationView and NavigationViewLink for navigation management, employing List and ForEach to display movie lists. Each entry is structured with HStack, containing elements like movie cover, title, and ratings. ScrollView and VStack are used for designing the layout of the detail page. We manage the state of the application through @State and @Binding, and handle network requests with Combine and URLSession to retrieve movie data from the server.

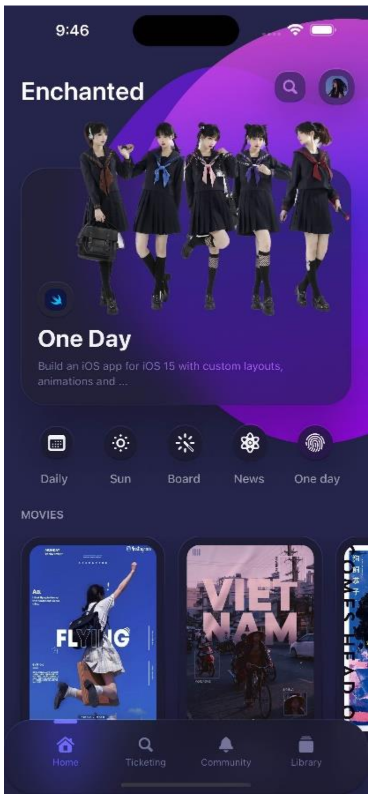
13 Interaction Design

In the 'Enchanted Movie' app, we've designed an intuitive navigation system, a practical search feature, and personalized options for favorites and sharing. Additionally, the app supports the system's default Dark and Light themes, ensuring a smooth, intuitive, and personalized user experience that caters to varying visual needs in different environments.

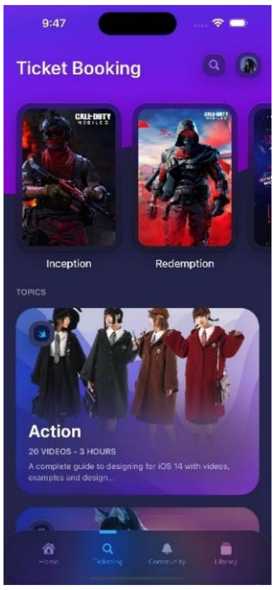
14 Front-end Main Interface Display

The following two pictures show two different themes of the main interface of the app, where the left picture is the bright theme and the right picture is the dark theme.



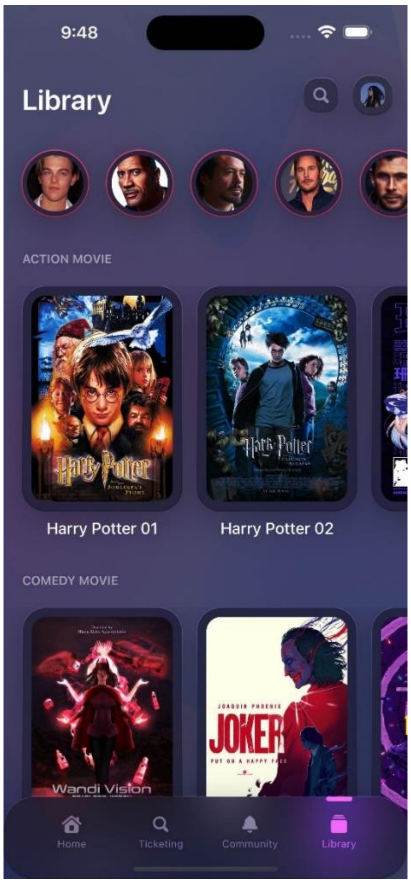


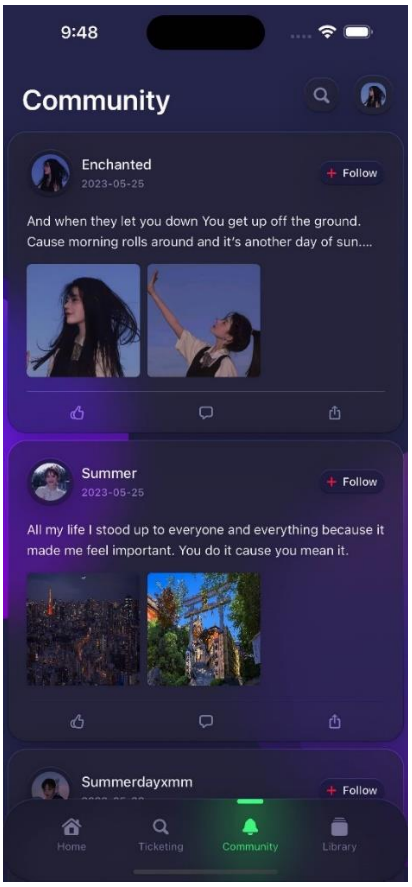
The following figure shows the movie ticketing interface in the APP system. This interface shows a variety of cinema movies for users and provides movie ticketing function.



The picture on the left below shows the movie watching function in APP, users can select movies in this interface and then watch them.

The picture on the right below shows the movie community function in APP, users can post or follow the movie in different communities to express their views on the movie and realize the mutual communication among users.





And when they let you down You get up off the ground. Cause morning rolls around and it's another day of sun....

All my life I stood up to everyone and everything because it made me feel important. You do it cause you mean it.