

# 1. 首页

## 1.1 轮播图

admin站点配置支持图片上传

```
pip install Pillow
```

默认情况下，Django会将上传的图片保存在本地服务器上，需要配置保存的路径。我们可以将上传的文件保存在静态文件statics目录中，添加如下上传保存目录信息

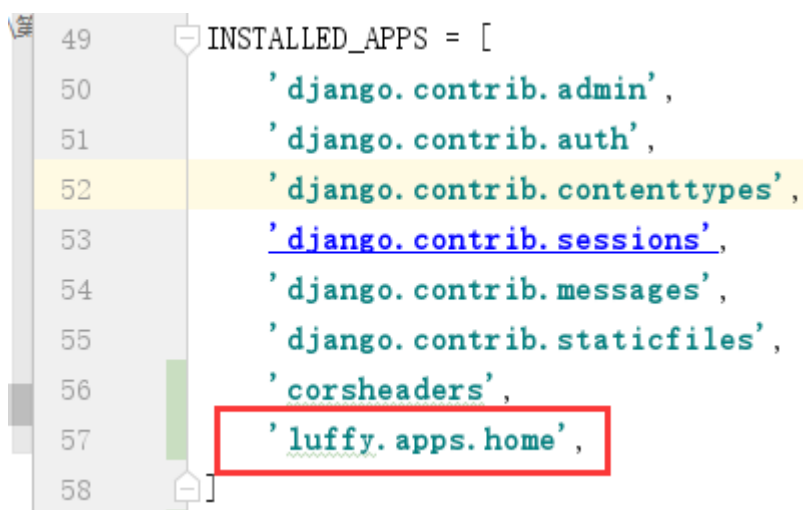
配置代码：

```
# 访问静态文件的url地址前缀
STATIC_URL = '/static/'
# 项目中存储上传文件的根目录[暂时配置]，注意，static目录需要手动创建否则上传文件时报错
MEDIA_ROOT=os.path.join(BASE_DIR,"luffy/static")

# 设置django的静态文件目录
STATICFILES_DIRS = [
    os.path.join(BASE_DIR,"luffy/static")
]
```

把Home子应用注册到settings的INSTALLED\_APPS中，并在models.py里面创建模型。

```
INSTALLED_APPS = [
    ...
    'luffy.apps.home',
]
```



```
49 INSTALLED_APPS = [
50     'django.contrib.admin',
51     'django.contrib.auth',
52     'django.contrib.contenttypes',
53     'django.contrib.sessions',
54     'django.contrib.messages',
55     'django.contrib.staticfiles',
56     'corsheaders',
57     'luffy.apps.home',
58 ]
```

使用图片字段 `ImageFiled`，字段选项使用 `upload_to` 可以设置保存图片的子目录，数据模型代码：

```

class bannerInfo(models.Model):
    """
    轮播图
    """
    # upload_to 存储子目录, 真实存放地址会使用配置中的MEDIA_ROOT+upload_to
    image = models.ImageField(upload_to='banner', verbose_name='轮播图', null=True)
    name = models.CharField(max_length=150, verbose_name='轮播图名称')
    link = models.CharField(max_length=150, verbose_name='轮播图广告地址')
    orders = models.IntegerField(verbose_name='显示顺序')
    is_show=models.BooleanField(verbose_name="是否上架",default=False)
    is_delete=models.BooleanField(verbose_name="逻辑删除",default=False)

    class Meta:
        db_table = 'ly_banner'
        verbose_name = '轮播图'
        verbose_name_plural = verbose_name

    def __str__(self):
        return self.name

```

模型数据迁移:

```

python manage.py makemigrations
python manage.py migrate

```

## 在xadmin站点添加轮播图数据

通过如下命令安装xadmin的最新版

```

pip install https://codeload.github.com/sshsfc/xadmin/zip/django2

```

在配置文件中注册如下应用

```

INSTALLED_APPS = [
    ...
    'rest_framework',
    'xadmin',
    'crispy_forms',
    'reversion',
    ...
]

# 修改使用中文界面
LANGUAGE_CODE = 'zh-Hans'

# 修改时区
TIME_ZONE = 'Asia/Shanghai'

```

xadmin有建立自己的数据库模型类，需要进行数据库迁移

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

在总路由中添加xadmin的路由信息

```
import xadmin
xadmin.autodiscover()

# version模块自动注册需要版本控制的 Model
from xadmin.plugins import xversion
xversion.register_models()

urlpatterns = [
    path(r'xadmin/', xadmin.site.urls)
]
```

创建超级用户

```
python manage.py createsuperuser
```

## 创建轮播图的模型管理类

luffy/home/adminx.py

```
import xadmin
from xadmin import views

class BaseSetting(object):
    """xadmin的基本配置"""
    enable_themes = True # 开启主题切换功能
    use_bootswatch = True

xadmin.site.register(views.BaseAdminView, BaseSetting)

class GlobalSettings(object):
    """xadmin的全局配置"""
    site_title = "路飞学城" # 设置站点标题
    site_footer = "路飞学城有限公司" # 设置站点的页脚
    menu_style = "accordion" # 设置菜单折叠

xadmin.site.register(views.CommAdminView, GlobalSettings)

# 轮播图
```

```
from home.models import bannerInfo
class BannerInfoModelAdmin(object):
    list_display=["name","orders","is_show"]
xadmin.site.register(bannerInfo, BannerInfoModelAdmin)
```

## 路由代码:

子应用路由:

```
from django.urls import path, re_path
from . import views
urlpatterns = [
    path(r"banner/", views.BannerInfoAPIView.as_view()),
]
```

总路由:

```
urlpatterns = [
    ...
    path('home/', include("home.urls")),
    # include 的值必须是 模块名.urls 格式, 字符串中间只能出现一个圆点
]
```

## 视图代码:

```
from django.db.models import Q
from rest_framework.views import APIView
from rest_framework.response import Response
from .models import bannerInfo
class BannerInfoAPIView(APIView):
    """
    轮播图列表
    """
    def get(self, request):
        # 获取数据
        banners = bannerInfo.objects.filter(Q(is_show=True) &
        Q(is_delete=False)).order_by("-orders")
        # 调整banners的images字段

        # 序列化
        data = []
        for item in banners:
            data.append({
                # 拼接图片的url地址
```

```
        "image": "/static/" + item.image.url,  
        "link": item.link,  
        "orders": item.orders,  
    })  
    return Response(data)
```

前端代码:

```
<template>  
  <div class="banner">  
    <el-carousel trigger="click" height="506px">  
      <el-carousel-item v-for="item in banner_list">  
        <a :href="item.link"></a>  
      </el-carousel-item>  
    </el-carousel>  
  </div>  
</template>  
  
<script>  
  export default {  
    name: "Banner",  
    data() {  
      return {  
        banner_list: [],  
      };  
    },  
    created: function() {  
      // 获取轮播图  
      this.$axios.get("http://api.luffycity.cn:8000/home/banner/").then(res => {  
        this.banner_list = res.data  
      }).catch(error => {  
        console.log(error);  
      });  
    }  
  }  
</script>  
  
<style scoped>  
  .banner img {  
    width: 100%;  
  }  
</style>
```

效果:



## 2. 显示登陆页面

前端引入登陆页面的组件代码并补充路由地址

Login.vue, 代码:

```
<template>
  <div class="login box">
    
    <div class="login">
      <div class="login-title">
        
        <p>帮助有志向的年轻人通过努力学习获得体面的工作和生活!</p>
      </div>
      <div class="login_box">
        <div class="title">
          <span @click="login_type=1" :class="login_type==1?'current':''">密码
          登录</span>
          <span @click="login_type=2" :class="login_type==2?'current':''">短信
          登录</span>
        </div>
        <div class="inp" :class="login_type==1?'show':''">
          <input v-model = 'username' type="text" placeholder="用户名 / 手机号" class="user">
          <input v-model = 'password' type="password" name="" class="pwd" placeholder="密码">
          <div id="geetest1" title="验证码"></div>
          <div class="remember">
            <p>
              <input type="checkbox" class="no" name="a"></input>
              <span>记住密码</span>
            </p>
          </div>
        </div>
      </div>
    </div>
  </div>
```



```
    width: 500px;
    height: 400px;
    top: 50%;
    left: 50%;
    margin-left: -250px;
    margin-top: -300px;
}
.login .login-title{
    width: 100%;
    text-align: center;
}
.login-title img{
    width: 190px;
    height: auto;
}
.login-title p{
    font-family: PingFangSC-Regular;
    font-size: 18px;
    color: #fff;
    letter-spacing: .29px;
    padding-top: 10px;
    padding-bottom: 50px;
}
.login_box{
    width: 400px;
    height: auto;
    background: #fff;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.5);
    border-radius: 4px;
    margin: 0 auto;
    padding-bottom: 40px;
}
.login_box .title{
    font-size: 20px;
    color: #9b9b9b;
    letter-spacing: .32px;
    border-bottom: 1px solid #e6e6e6;
    display: flex;
    justify-content: space-around;
    padding: 50px 60px 0 60px;
    margin-bottom: 20px;
    cursor: pointer;
}
.login_box .title .current{
    color: #4a4a4a;
    border-bottom: 2px solid #84cc39;
}

.inp{
    width: 350px;
    margin: 0 auto;
    display: none;
}
```



```
.show{
  display: block;
}
.inp input{
  border: 0;
  outline: 0;
  width: 100%;
  height: 45px;
  border-radius: 4px;
  border: 1px solid #d9d9d9;
  text-indent: 20px;
  font-size: 14px;
  background: #fff !important;
}
.inp input.user{
  margin-bottom: 16px;
}
.inp .rember{
  display: flex;
  justify-content: space-between;
  align-items: center;
  position: relative;
  margin-top: 10px;
}
.inp .rember p:first-of-type{
  font-size: 12px;
  color: #4a4a4a;
  letter-spacing: .19px;
  margin-left: 22px;
  display: -ms-flexbox;
  display: flex;
  -ms-flex-align: center;
  align-items: center;
  /*position: relative;*/
}
.inp .rember p:nth-of-type(2){
  font-size: 14px;
  color: #9b9b9b;
  letter-spacing: .19px;
  cursor: pointer;
}

.inp .rember input{
  outline: 0;
  width: 30px;
  height: 45px;
  border-radius: 4px;
  border: 1px solid #d9d9d9;
  text-indent: 20px;
  font-size: 14px;
  background: #fff !important;
}
```

```

.inp .remember p span{
  display: inline-block;
  font-size: 12px;
  width: 100px;
  /*position: absolute;*/
  /*left: 20px;*/
}
#geetest{
  margin-top: 20px;
}
.login_btn{
  width: 100%;
  height: 45px;
  background: #84cc39;
  border-radius: 5px;
  font-size: 16px;
  color: #fff;
  letter-spacing: .26px;
  margin-top: 30px;
}
.inp .go_login{
  text-align: center;
  font-size: 14px;
  color: #9b9b9b;
  letter-spacing: .26px;
  padding-top: 20px;
}
.inp .go_login span{
  color: #84cc39;
  cursor: pointer;
}
</style>

```

在routes/index.js中，添加路由

```

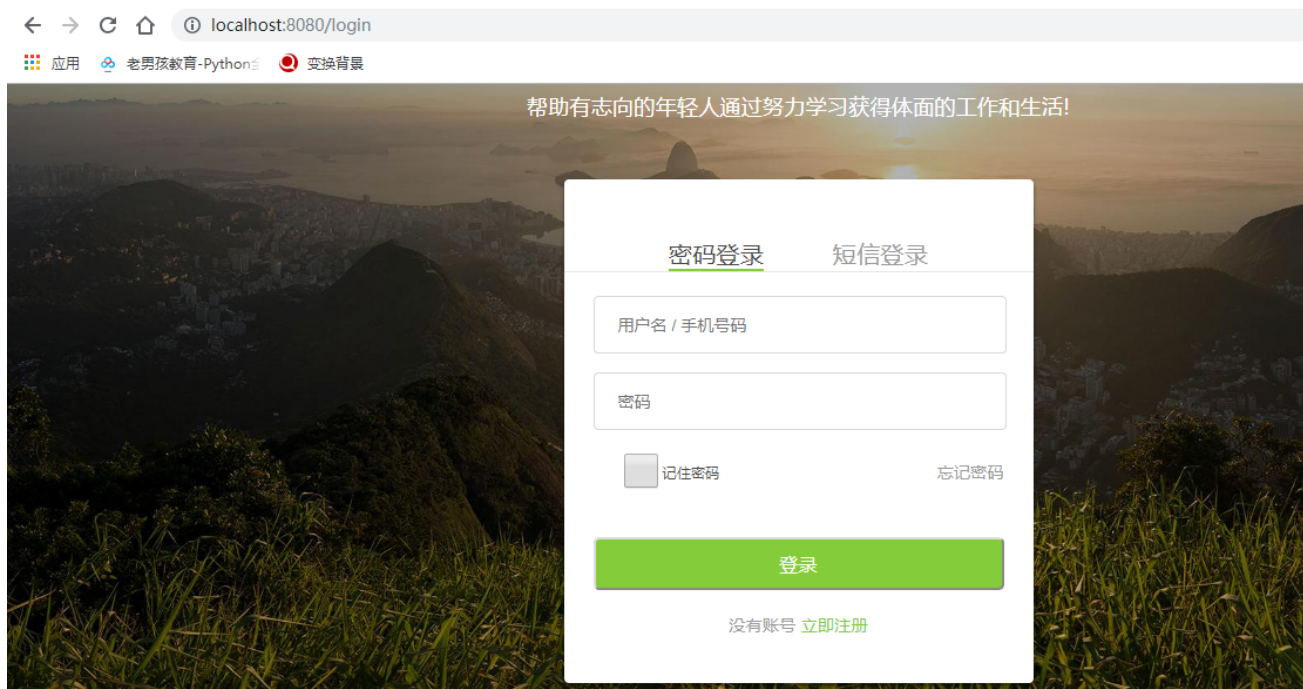
import Vue from "vue"
import Router from "vue-router"

// 导入需要注册路由的组件
import Home from "../components/Home"
import Login from "../components/Login"
Vue.use(Router);

// 配置路由列表
export default new Router({
  mode:"history",
  routes:[
    // 路由列表
    {
      name:"Home",
      path: "/home",

```

```
        component: Home,
    },
    {
        name: "Home",
        path: "/",
        component: Home,
    },
    {
        name: "Login",
        path: "/login",
        component: Login,
    }
]
})
```



## 2. 用户认证

Django提供了认证系统。认证系统包含：

- 用户
- 权限：二元（是/否）标志指示一个用户是否可以做一个特定的任务。
- 组：对多个用户运用标签和权限的一种通用的方式。
- 一个可配置的密码哈希系统
- 用户登录或内容显示的表单和视图
- 一个可插拔的后台系统

Django默认用户的认证机制依赖Session机制，我们在本项目中将引入JWT认证机制，将用户的登陆状态[认证信息]存放在Token字符串中，然后对接Django的认证系统，帮助我们来实现：

- 用户的数据模型
- 用户密码的加密与验证
- 用户的权限系统

## 2.1 Django用户模型类

---

Django认证系统中提供了用户模型类User保存用户的数据，默认的用户包含以下常见的基本字段：

- `username`  
必选。150个字符以内。用户名可能包含字母数字，`_`，`@`，`+`，`.` 和 `-` 个字符。在Django更改1.10：`max_length` 从30个字符增加到150个字符。
- `email`  
可选（`blank=True`）。邮箱地址。
- `password`  
必选。密码的哈希及元数据。（Django 不保存原始密码）。原始密码可以无限长而且可以包含任意字符。
- `groups`  
与 `Group` 之间的多对多关系。
- `user_permissions`  
与 `Permission` 之间的多对多关系。
- `is_staff`  
布尔值。指示用户是否可以访问Admin 站点。
- `is_active`  
布尔值。指示用户的账号是否激活。我们建议您将此标志设置为 `False` 而不是删除帐户；这样，如果您的应用程序对用户有任何外键，则外键不会中断。它不是用来控制用户是否能够登录。在Django更改1.10：在旧版本中，默认`is_active`为`False`不能进行登录。
- `is_superuser`  
布尔值。指定这个用户拥有所有的权限而不需要给他们分配明确的权限。
- `last_login`  
用户最后一次登录的时间。
- `date_joined`  
账户创建的时间。当账号创建时，默认设置为当前的date/time。

### 常用方法：

- `set_password (raw_password)`  
设置用户的密码为给定的原始字符串，并负责密码的。不会保存 `User` 对象。当 `None` 为 `raw_password` 时，密码将设置为一个不可用的密码。
- `check_password (raw_password)`  
如果给定的`raw_password`是用户的真实密码，则返回`True`，可以在校验用户密码时使用。

### 管理器方法：

管理器方法即可以通过 `User.objects.` 进行调用的方法。

- `create_user(username, email=None, password=None, **extra_fields)`  
创建、保存并返回一个 `User` 对象。
- `create_superuser(username, email, password, **extra_fields)`  
与 `create_user()` 相同，但是设置 `is_staff` 和 `is_superuser` 为 `True`。

## 2.2 创建自定义的用户模型类

Django认证系统中提供的用户模型类及方法很方便，我们可以使用这个模型类，但是字段有些无法满足项目需求，如本项目中需要保存用户的手机号，需要给模型类添加额外的字段。

Django提供了 `django.contrib.auth.models.AbstractUser` 用户抽象模型类允许我们继承，扩展字段来使用Django认证系统的用户模型类。

**我们可以在apps中创建Django应用users，并在配置文件中注册users应用。**

在创建好的应用models.py中定义用户的用户模型类。

```
class User(AbstractUser):
    """用户模型类"""
    mobile = models.CharField(max_length=11, unique=True, verbose_name='手机号')

    class Meta:
        db_table = 'ly_users'
        verbose_name = '用户'
        verbose_name_plural = verbose_name
```

我们自定义的用户模型类还不能直接被Django的认证系统所识别，需要在配置文件中告知Django认证系统使用我们自定义的模型类。

在settings.py配置文件中设置

```
AUTH_USER_MODEL = 'users.User'
```

`AUTH_USER_MODEL` 参数的设置以 `点.` 来分隔，表示 `应用名.模型类名`。

**注意：**Django建议我们对于`AUTH_USER_MODEL`参数的设置一定要在第一次数据库迁移之前就设置好，否则后续使用可能出现未知错误。

执行数据库迁移

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

## 2.3 Django REST framework JWT



- **sub**: jwt所面向的用户
- **aud**: 接收jwt的一方
- **exp**: jwt的过期时间, 这个过期时间必须要大于签发时间
- **nbf**: 定义在什么时间之前, 该jwt都是不可用的.
- **iat**: jwt的签发时间
- **jti**: jwt的唯一身份标识, 主要用来作为一次性token,从而回避重放攻击。

**公共的声明**：公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密。

**私有的声明**：私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

定义一个payload:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

然后将其进行base64加密，得到JWT的第二部分。

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRTaw4iOnRydWV9
```

## 2.3.4 signature

JWT的第三部分是一个签证信息，这个签证信息由三部分组成：

- header (base64后的)
- payload (base64后的)
- secret

这个部分需要base64加密后的header和base64加密后的payload使用 `.` 连接组成的字符串，然后通过header中声明的加密方式进行加盐 `secret` 组合加密，然后就构成了jwt的第三部分。

```
// javascript
var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);

var signature = HMACSHA256(encodedString, 'secret'); //
TjVA95OrM7E2cBab30RMhRHDCEfXjoYZgeFONFh7HgQ
```

将这三部分用 `.` 连接成一个完整的字符串,构成了最终的jwt:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRTaw4iOnRydWV9.TjVA95OrM7E2cBab30RMhRHDCEfXjoYZgeFONFh7HgQ
```

**注意**：`secret`是保存在服务器端的，jwt的签发生成也是在服务器端的，`secret`就是用来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个`secret`，那就意味着客户端是可以自我签发jwt了。

关于签发和核验JWT，我们可以使用Django REST framework JWT扩展来完成。

文档网站<http://getblimp.github.io/django-rest-framework-jwt/>

<http://getblimp.github.io/django-rest-framework-jwt/>

## 2.3.5 安装配置JWT

安装

```
pip install djangorestframework-jwt
```

settings.py的配置代码中

```
REST_FRAMEWORK = {
    # 用户登陆认证方式
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}
import datetime
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
}
```

- JWT\_EXPIRATION\_DELTA 指明token的有效期

## 2.3.6 使用

Django REST framework JWT 扩展的说明文档中提供了手动签发JWT的方法

```
from rest_framework_jwt.settings import api_settings

jwt_payload_handler = api_settings.JWT_PAYLOAD_HANDLER
jwt_encode_handler = api_settings.JWT_ENCODE_HANDLER

payload = jwt_payload_handler(user)
token = jwt_encode_handler(payload)
```

在用户注册或登录成功后，在序列化器中返回用户信息以后同时返回token即可。



### 2.3.7 前端保存jwt

我们可以将JWT保存在cookie中，也可以保存在浏览器的本地存储里，我们保存在浏览器本地存储中浏览器的本地存储提供了sessionStorage 和 localStorage 两种：

- **sessionStorage** 浏览器关闭即失效
- **localStorage** 长期有效

使用方法

```
sessionStorage.变量名 = 变量值    // 保存数据
sessionStorage.变量名    // 读取数据
sessionStorage.clear()    // 清除所有sessionStorage保存的数据

localStorage.变量名 = 变量值    // 保存数据
localStorage.变量名    // 读取数据
localStorage.clear()    // 清除所有localStorage保存的数据
```

### 2.3.8 jwt后端认证

Django REST framework JWT提供了登录签发JWT的视图，可以直接使用

```
from rest_framework_jwt.views import obtain_jwt_token

urlpatterns = [
    url(r'^authorizations/$', obtain_jwt_token),
]
```

但是默认的返回值仅有token，我们还需在返回值中增加username和user\_id。

通过修改该视图的返回值可以完成我们的需求。

在users/utls.py 中，创建

```
def jwt_response_payload_handler(token, user=None, request=None):
    """
    自定义jwt认证成功返回数据
    """
    return {
        'token': token,
        'user_id': user.id,
        'username': user.username
    }
```

修改配置文件

```
# JWT
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
    'JWT_RESPONSE_PAYLOAD_HANDLER': 'users.utils.jwt_response_payload_handler',
}
```

## 2.3.9 多条件登录

JWT扩展的登录视图，在收到用户名与密码时，也是调用Django的认证系统中提供的`authenticate()`来检查用户名与密码是否正确。

我们可以通过修改Django认证系统的认证后端（主要是`authenticate`方法）来支持登录账号既可以是用户名也可以是手机号。

**修改Django认证系统的认证后端需要继承`django.contrib.auth.backends.ModelBackend`，并重写`authenticate`方法。**

`authenticate(self, request, username=None, password=None, **kwargs)` 方法的参数说明：

- request 本次认证的请求对象
- username 本次认证提供的用户账号
- password 本次认证提供的密码

**我们想要让用户既可以以用户名登录，也可以以手机号登录，那么对于`authenticate`方法而言，`username`参数即表示用户名或者手机号。**

重写`authenticate`方法的思路：

1. 根据`username`参数查找用户`User`对象，`username`参数可能是用户名，也可能是手机号
2. 若查找到`User`对象，调用`User`对象的`check_password`方法检查密码是否正确

在`users/utils.py`中编写：

```
def get_user_by_account(account):
    """
    根据帐号获取user对象
    :param account: 账号，可以是用户名，也可以是手机号
    :return: User对象 或者 None
    """
    try:
        if re.match('^1[3-9]\d{9}$', account):
            # 帐号为手机号
            user = User.objects.get(mobile=account)
        else:
            # 帐号为用户名
            user = User.objects.get(username=account)
    except User.DoesNotExist:
        return None
    else:
        return user
```

```

        return user

class UsernameMobileAuthBackend(ModelBackend):
    """
    自定义用户名或手机号认证
    """

    def authenticate(self, request, username=None, password=None, **kwargs):
        user = get_user_by_account(username)
        if user is not None and user.check_password(password):
            return user

```

在配置文件中告知Django使用我们自定义的认证后端

```

AUTHENTICATION_BACKENDS = [
    'users.utils.UsernameMobileAuthBackend',
]

```

## 3. 课程模块

ContentType

### 3.1 模型构建

```

from django.db import models

# Create your models here.

from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import GenericForeignKey, GenericRelation

##### 课程表 #####

class CourseCategory(models.Model):

    name = models.CharField(max_length=64, unique=True)
    def __str__(self):
        return "%s" % self.name

    class Meta:
        verbose_name = "课程类"
        verbose_name_plural = "课程类"

```

```

class Course(models.Model):
    """
    专题课程
    """
    name = models.CharField(max_length=128, unique=True, verbose_name="模块")
    course_img = models.CharField(max_length=255)
    course_type_choices = ((0, '付费'), (1, 'VIP专享'), (2, '学位课程'))
    course_type = models.SmallIntegerField(choices=course_type_choices)
    brief = models.TextField(verbose_name="课程概述", max_length=2048)
    level_choices = ((0, '初级'), (1, '中级'), (2, '高级'))
    level = models.SmallIntegerField(choices=level_choices, default=1)
    pub_date = models.DateField(verbose_name="发布日期", blank=True, null=True)
    period = models.PositiveIntegerField(verbose_name="建议学习周期(days)", default=7)
    order = models.IntegerField(verbose_name="课程顺序", help_text="从上一个课程数字往后排")
    attachment_path = models.CharField(max_length=128, verbose_name="课件路径",
    blank=True, null=True)
    status_choices = ((0, '上线'), (1, '下线'), (2, '预上线'))
    status = models.SmallIntegerField(choices=status_choices, default=0)

    course_category = models.ForeignKey("CourseCategory", on_delete=models.CASCADE, null=True,
    blank=True)
    order_details = GenericRelation("OrderDetail", related_query_name="course")
    coupon = GenericRelation("Coupon")
    price_policy = GenericRelation("PricePolicy")

    def __str__(self):
        return "%s(%s)" % (self.name, self.get_course_type_display())

class CourseDetail(models.Model):
    """课程详情页内容"""

    course = models.OneToOneField("Course", on_delete=models.CASCADE)
    hours = models.IntegerField(verbose_name="课时")
    course_slogan = models.CharField(max_length=125, blank=True, null=True)
    video_brief_link = models.CharField(max_length=255, blank=True, null=True)
    why_study = models.TextField(verbose_name="为什么学习这门课程")
    what_to_study_brief = models.TextField(verbose_name="我将学到哪些内容")
    career_improvement = models.TextField(verbose_name="此项目如何有助于我的职业生涯")
    prerequisite = models.TextField(verbose_name="课程先修要求", max_length=1024)
    recommend_courses = models.ManyToManyField("Course", related_name="recommend_by",
    blank=True)
    teachers = models.ManyToManyField("Teacher", verbose_name="课程讲师")

    def __str__(self):
        return "%s" % self.course

class Teacher(models.Model):
    """讲师、导师表"""

    name = models.CharField(max_length=32)
    role_choices = ((0, '讲师'), (1, '导师'))
    role = models.SmallIntegerField(choices=role_choices, default=0)

```

```

        title = models.CharField(max_length=64, verbose_name="职位、职称")
        signature = models.CharField(max_length=255, help_text="导师签名", blank=True,
null=True)
        image = models.CharField(max_length=128)
        brief = models.TextField(max_length=1024)

    def __str__(self):
        return self.name

class PricePolicy(models.Model):
    """价格与有课程效期表"""
    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE) # 关联course
or degree_course
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey('content_type', 'object_id')
    # course = models.ForeignKey("Course")
    valid_period_choices = ((1, '1天'), (3, '3天'),
                            (7, '1周'), (14, '2周'),
                            (30, '1个月'),
                            (60, '2个月'),
                            (90, '3个月'),
                            (120, '4个月'),
                            (180, '6个月'), (210, '12个月'),
                            (540, '18个月'), (720, '24个月'),
                            (722, '24个月'), (723, '24个月'),
                            )
    valid_period = models.SmallIntegerField(choices=valid_period_choices)
    price = models.FloatField()

    class Meta:
        unique_together = ("content_type", 'object_id', "valid_period")

    def __str__(self):
        return "%s(%s)%s" % (self.content_object, self.get_valid_period_display(),
self.price)

class CourseChapter(models.Model):
    """课程章节"""
    course = models.ForeignKey("Course",
related_name='coursechapters', on_delete=models.CASCADE)
    chapter = models.SmallIntegerField(verbose_name="第几章", default=1)
    name = models.CharField(max_length=128)
    summary = models.TextField(verbose_name="章节介绍", blank=True, null=True)
    is_create = models.BooleanField(verbose_name="是否创建题库进度", default=True)
    pub_date = models.DateField(verbose_name="发布日期", auto_now_add=True)

    class Meta:
        unique_together = ("course", 'chapter')

    def __str__(self):
        return "%s:(第%s章)%s" % (self.course, self.chapter, self.name)

class CourseSection(models.Model):

```

```

"""课时目录"""
chapter = models.ForeignKey("CourseChapter",
related_name='coursesections', on_delete=models.CASCADE)
name = models.CharField(max_length=128)
order = models.PositiveSmallIntegerField(verbose_name="课时排序", help_text="建议每个
课时之间空1至2个值，以备后续插入课时")
section_type_choices = ((0, '文档'), (1, '练习'), (2, '视频'))
section_type = models.SmallIntegerField(default=2, choices=section_type_choices)
section_link = models.CharField(max_length=255, blank=True, null=True,
help_text="若是video, 填vid,若是文档, 填link")
video_time = models.CharField(verbose_name="视频时长", blank=True, null=True,
max_length=32) # 仅在前端展示使用
pub_date = models.DateTimeField(verbose_name="发布时间", auto_now_add=True)
free_trail = models.BooleanField(verbose_name="是否可试看", default=False)
is_flash = models.BooleanField(verbose_name="是否使用FLASH播放", default=False)
player_choices = ((0, "CC"), (1, "POLYV"), (2, "ALI"))
player = models.SmallIntegerField(choices=player_choices, default=1, help_text="视频
播放器选择")

def course_chapter(self):
    return self.chapter.chapter

def course_name(self):
    return self.chapter.course.name

class Meta:
    unique_together = ('chapter', 'section_link')

def __str__(self):
    return "%s-%s" % (self.chapter, self.name)

```

## 3.2 课程内容展示