

Django 后端小作业文档

实验目的

- 了解前后端分离的设计思想
- 以Django框架为例，了解后端运作的基本原理以及基础的设计思路
- 学习基于HTTP的API设计、使用、测试方式
- 学习数据库的基本操作和设计思路
- 为日后软工大作业打基础

实验环境

python >= 3.6

Django >= 4.0

Pillow >= 9.0

数据库使用Django默认的sqlite3

实验要求

本次作业中，项目的基本框架已经提供给同学，项目中有若干个代码段需要同学补充。为了减轻同学们的工作量，我们在此次作业中不会重点考察后端的安全和性能问题，仅涉及后端的基本设计，以及一些简单的字段验证。

当你的环境配置完成后，我们提供的项目框架是可以直接运行的。在命令行中切换到项目目录并运行 `python3 manage.py runserver`，此时我们的项目已经运行起来了，打开浏览器在地址栏中输入 `localhost:8000`，当你看到如下界面时说明你的 Django 环境已经配置成功。

django

View [release notes](#) for Django 4.0



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



Django Documentation

Topics, references, & how-to's



Tutorial: A Polling App

Get started with Django



Django Community

Connect, get help, or contribute

接下来是本次作业四个主要功能点。

1 配置路由

在 board 应用中我们已经配置好了该应用的路由（请见 `board/urls.py`），之后我们需要将该应用的路由包含进整个项目的路由中，并且让 board 应用的路由前缀为 `api`。更多和路由相关的内容请见 [URL 调度器 | Django 文档 | Django](#)

此阶段，你需要修改的文件是 `messageboard/urls.py`。

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    # 利用include()函数为board应用添加前缀为api的路由
    # -----
    path('admin/', admin.site.urls),
]

```

最终效果：我们的项目中有一条路由为/api/message，这个url将会是我们暴露给前端的接口，后续工作将会围绕着这个接口进行。当你在浏览器中输入localhost:8000/api/message时，浏览器界面上会显示类似如下的错误信息，因为我们的后端代码还没有完善。但是此时你会发现，我们错误信息不是404 Not Found而是和后端代码相关的一些信息。这说明我们的路由已经配置成功。

FieldError at /api/message

Cannot resolve keyword 'pub_date' into field. Choices are: id

Request Method: GET

Request URL: http://127.0.0.1:8000/api/message

Django Version: 4.0.2

Exception Type: FieldError

Exception Value: Cannot resolve keyword 'pub_date' into field. Choices are: id

2 完善模型

我们需要给board应用完善模型。模型是真实数据的简单明确的描述。它包含了储存的数据所必要的字段和行为。模型的设计直接和数据库的存储逻辑相关。

更多和Django模型相关的内容请见[编写你的第一个 Django 应用，第 2 部分 | Django 文档 | Django](#)

在此阶段中，我们已经提供了User模型作为参考，同学们需要设计一个Message模型，该模型对应着留言板中的留言信息。

此阶段，你需要修改的文件是**board/models.py**。

```

from django.db import models
from django.forms import ModelForm

class User(models.Model):
    name = models.CharField(unique=True, max_length=20)
    register_date = models.DateTimeField(auto_now_add=True)
    avatar = models.ImageField(upload_to='avatars', blank=True) # AVATAR

    def __str__(self):
        return self.name

```

```
class Message(models.Model):
# -----
# 完善Message模型的代码，共有四个字段
# user: ForeignKey, on_delete策略使用CASCADE
# title: CharField, max_length=100
# content: CharField, max_length=500
# pub_date: DateTimeField, auto_now_add=True
# -----
```

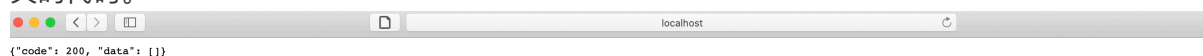
Message模型中共有四个字段：

- user：留言者，使用外键和User模型进行关联，一个User可以对应多个Message但是一个Message仅对应一个User，外键的on_delete策略使用CASCADE
- title：留言的标题，使用CharField，限制最大长度为100
- content：留言的内容，使用CharField，限制最大长度为500
- pub_date：留言的时间，使用DateTimeField，需要配置为在表项插入时自动添加

当你完成了模型的编码时，在命令行中运行

```
# Django会根据你设计的模型生成一系列指令，这些指令告诉数据库该如何创建表项，生成的指令文件存于board/migrations
python3 manage.py makemigrations board
# 根据上一步生成的指令创建数据库，此命令执行后和board中模型相关的数据库表项真正被创建
python3 manage.py migrate
```

最终效果：当Message模型补充完成之后，此次作业的模式设计即完成。此时你可以通过Django命令行工具（运行python3 manage.py shell）操作进行数据库表项的增删改查。当你在浏览器中输入localhost:8000/api/message时，浏览器界面上会显示如下信息。这说明你已经完成和模型相关的代码。



3 完善视图

在board应用中，我们定义了message视图，该视图为board应用的接口。message视图中包含着接口的核心业务逻辑。

message视图的大体逻辑已经提供给同学（请见board/views.py），业务逻辑细节需要同学们进行填充。

该视图支持两种类型的请求，分别是GET和POST。其中GET用来获得留言板中的信息以便前端进行展示，而POST用来向留言板中添加一条留言。更多和视图相关的内容请见[编写你的第一个 Django](#)

[应用, 第3部分 | Django 文档 | Django](#)

此阶段, 你需要修改的文件是**board/views.py**。

```
from django.shortcuts import render
from django.http import JsonResponse, HttpResponse
import json
from .models import User, Message
from django.core.exceptions import ValidationError

# Create your views here.
def message(request):
    def gen_response(code: int, data: str):
        return JsonResponse({
            'code': code,
            'data': data
        }, status=code)
    # GET的完整实现已经给出, 同学们无需修改
    if request.method == 'GET':
        limit = request.GET.get('limit', default='100')
        offset = request.GET.get('offset', default='0')
        if not limit.isdigit():
            return gen_response(400, '{} is not a number'.format(limit))
        if not offset.isdigit():
            return gen_response(400, '{} is not a number'.format(offset))

        return gen_response(200, [
            {
                'title': msg.title,
                'message': msg.content,
                'user': msg.user.name,
                'timestamp': int(msg.pub_date.timestamp())
            }
            for msg in Message.objects.all().order_by('-pk')
            [int(offset) : int(offset) + int(limit)]
        ])

    elif request.method == 'POST':
        # 从cookie中获得user的名字, 如果user不存在则新建一个
        # 如果cookie中没有user则使用"Unknown"作为默认用户名
        name = request.COOKIES['user'] if 'user' in request.COOKIES else
'Unknown'
        user = User.objects.filter(name=name).first()
        if not user:
```

```

        user = User(name = name)
        try:
            # 注意在调用full_clean()时Django会自动检测字段的有效性, 这个有效性检测包括检测CharField是否满足最大长度限制
            user.full_clean()
            # 存入数据库
            user.save()
        except ValidationError as e:
            return gen_response(400, "Validation Error of user: {}".format(e))

        # 验证请求的数据格式是否符合JSON规范(请求体可通过json.loads()即可), 如果不符合则返回code 400, data字段内容自定义即可
        # -----

        # 验证请求数据是否满足接口要求, 若通过所有的验证, 则将新的消息添加到数据库中。如果不符合要求则返回code 400, data字段内容自定义即可
        # PS: 请求数据体应该为{"title": "something", "content": "something"}, 请确保title和content字段存在, 并且title和content均有最大长度限制。
        # PS: 检测方式可以参考user, 使用Django提供的full_clean()方法进行检测
        # -----

        # 添加成功返回code 201
        return gen_response(201, "message was sent successfully")

    else:
        return gen_response(405, 'method {} not allowed'.format(request.method))

```

GET请求的接口规定为:

GET /api/message

参数:

- offset: 数值类型, 表示从第offset个(从0开始计数, 以主键为索引)开始, 可省略, 默认值为0。
- limit: 数值类型, 表示一次GET请求最多limit条留言, 可省略, 默认值为100。
offset和limit主要作用为限制一次GET请求获得的留言范围, 例如, 当offset=10, limit=50, 表示获取从第10条(从0开始计数, 包括第10条)开始, 之后最多50条留言。

响应内容:

响应主体内容为一个JSON, 该JSON包含两个字段, 返回码“code”和数据“data”

- 请求成功时: 返回码200, 数据段为一个列表, 其中每个元素对应一条留言。列表的每一个元

素为一个字典，字典结构为{"title": "<留言的标题>", "content": "<留言的内容>", "user": "<留言者的用户名>", "timestamp": "<留言时间，以秒为单位的整数时间戳>"}. 列表中的元素按照留言时间从晚到早进行排序（早代表时间戳数值较小，晚则反之，即时间戳降序排列）。

- 请求失败时：此方法中仅验证参数的有效性，即错误返回码只有400，数据段中包含错误信息。

注意：此部分代码作业框架中已经提供，同学们无需修改。

POST请求的接口规定为：

POST /api/message

参数：

- title：字符串类型，留言标题
- content：字符串类型，留言的内容
- user：字符串类型，留言者的用户名，可省略（注意：**user**参数并不出现在请求体中，而是放在请求的**cookie**中，当**cookie**中没有**user**项时，则会使用默认用户“Unknown”）

响应内容：

响应主体内容为一个JSON，该JSON包含两个字段，返回码“code”和数据“data”

- 请求成功时：返回码201，数据段为成功信息“message was sent successfully”。
- 请求失败时：此方法中仅验证参数的有效性，即错误返回码只有400，数据段中包含错误信息，错误信息自定义即可。在POST方法中我们需要验证：
 - user字段的有效性，注意user字段有最大长度限制
 - 请求体的数据是否符合JSON规范
 - 请求体中“title”和“content”是否存在并符合长度要求。

TIPS: 在API开发过程中，我们强烈建议使用Postman之类的工具对你写的接口进行测试。这会大大减少你debug的时间。

最终效果：当视图补充完成时，此次实验编码阶段接近尾声，你可以运行预先写好的单元测试（python3 manage.py test）来验证实现的正确性。当你完成上述所有功能时，会有类似下图的结果，此时你可以通过7个测试点。

```
System check identified 2 issues (0 silenced).
.EF....F..
=====
ERROR: test_avatar_for_user (board.tests.MessageModelTests)
=====
```

4 添加一个接口：清空留言板中留言

当你完成上述三个功能点之后，想必已经熟悉了Django框架的基本操作逻辑。此时，请你添加一个新的接口，该接口需求如下：

GET /api/clearmessage

无需参数。该接口的功能为清空留言板中所有的Message。当清空留言板成功后返回一个响应。

响应内容：

响应主体内容为一个JSON，该JSON包含两个字段，返回码“code”和数据“data”

- 返回码200，数据段内容自定义。

TIPS： 在添加此接口过程中，你需要修改的文件有board/urls.py以及board/views.py

最终效果：当该功能成功添加后，你可以运行预先写好的单元测试（python3 manage.py test）来验证实现的正确性。此时你可以通过8个测试点。

```
System check identified 2 issues (0 silenced).
.E.....F..
=====
ERROR: test_avatar_for_user (board.tests.MessageModelTests)
=====
```

5 添加一个接口：返回某个用户的所有留言

POST /api/messages_for_user

返回某个用户的所有留言。

参数：

- user：字符串类型，留言者的用户名（此接口user参数在请求体中）

响应内容：

响应主体内容为一个JSON，该JSON包含两个字段，返回码“code”和数据“data”

- 请求成功时：返回码200，数据段为一个列表，其中每个元素对应一条留言。列表的每一个元素为一个字典，字典结构为{"title": "<留言的标题>", "content": "<留言的内容>", "timestamp": "<留言时间，以秒为单位的整数时间戳>"}.列表中的元素按照留言时间从晚到早进行排序（早代表时间戳数值较小，晚则反之，即时间戳降序排列）。
- 请求失败时：如果没有相应用户，或者请求格式错误则说明请求失败，返回码400，数据段自定义。

TIPS： 在添加此接口过程中，你需要修改的文件有board/urls.py以及board/views.py

```
System check identified 2 issues (0 silenced).
.E.....
=====
ERROR: test_avatar_for_user (board.tests.MessageModelTests)
=====
```


6 显示用户的头像

GET /api/avatar

获得某个用户的头像。

响应内容：

返回一个图像资源。

参数：

- user：路径参数，留言者的用户名。例如，访问/api/pic/avatars/Alice，应当返回Alice的头像。头像不存在时，返回任意非2XX的返回码即可。

对于这个接口，所有需要显示的图像由接口7上传。你不需要考虑一些与操作系统相关的问题，例如文件名的大小写等，我们测试时会尽量避开这些情况。

TIPS： Django对此类需求提供了易用的实现方式，请参阅注释、<https://docs.djangoproject.com/zh-hans/4.0/ref/settings/#media-root> 和 <https://docs.djangoproject.com/zh-hans/4.0/topics/files/>。当然，我们也欢迎你采用任意的实现方法(例如直接读写文件)。注意，如果请求的路径不对应一个合法的用户名，应该拒绝这个请求。**绝对不能**暴露后端服务器上的其他文件。

7 上传用户头像

POST /api/avatar

上传一个用户的头像。

参数：

注意：该请求参数为form(表单)形式，见代码注释

user：用户名

pic：一个png图像文件。

响应内容：

上传成功时，返回码为200并能反映上传成功即可，同一用户重复上传应该能够覆盖之前的文件。上传成功后，调用接口6应该可以看到上传的头像，且应当持久化，重启服务后仍可访问。

无关本次作业的提示：

- 虽然本次作业不要求，但是实际开发中请务必做充分的校验(文件名，大小，格式，用户权限等)。
- 在大作业中，头像功能有很多种更加简单的实现方法，例如直接在前端调用Gravatar API(<http://en.gravatar.com/site/implement/images/django/>)

最终效果：当该功能成功添加后，你可以运行预先写好的单元测试（python3 manage.py test）来验证实现的正确性。此时你将通过所有的测试点。

```
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 10 tests in 0.022s

OK
Destroying test database for alias 'default'...
```

TIPS： 接口6、7相关的部分均有注释AVATAR，可以通过搜索快速查看。

评分标准

1. 所有功能点均正确完成，并顺利通过单元测试记满分
2. 完成部分功能点，单元测试部分通过酌情扣分