

# 公开匿名投票系统

---

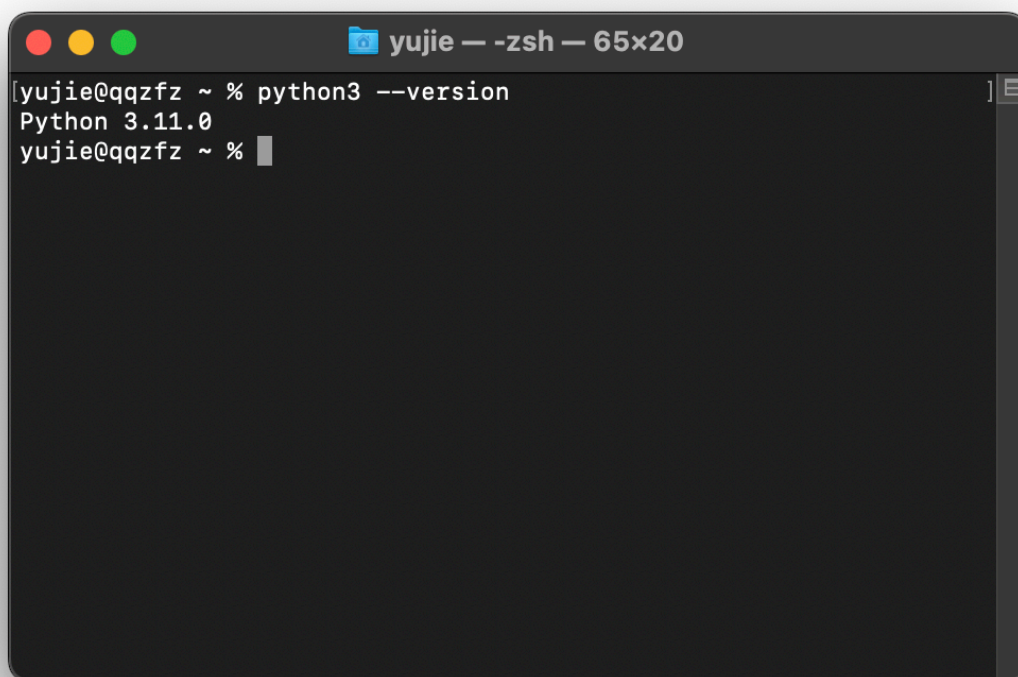
本系统基于python、django开发，在django官方教程的基础上实现了一个更为完善的公开匿名投票系统。

## LAB1.1

---

### 1 安装环境

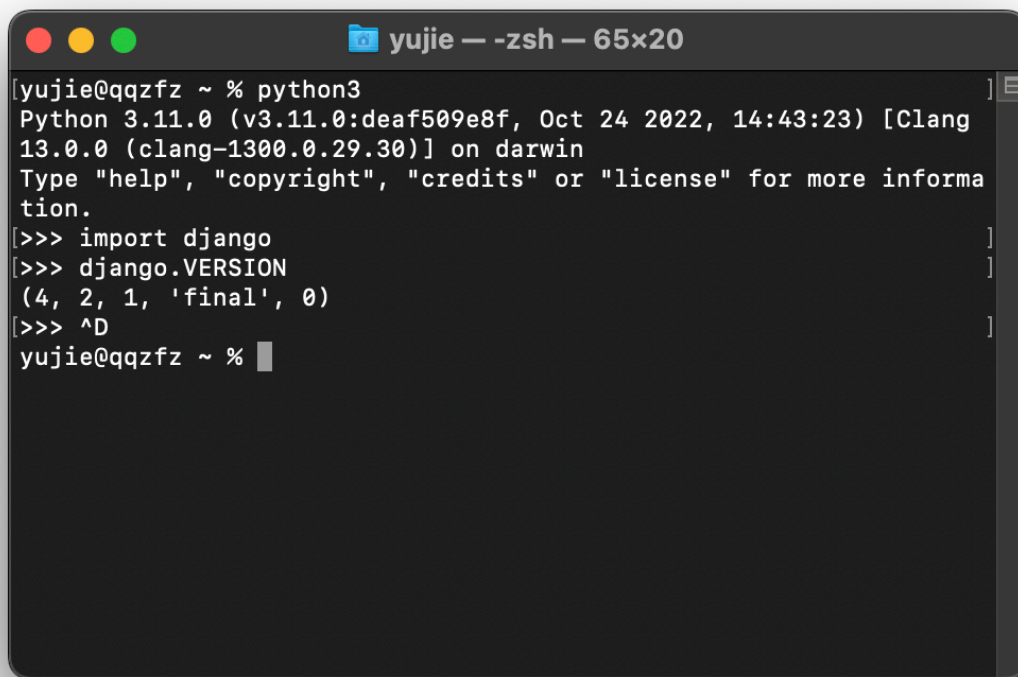
#### 1.1 python

A terminal window titled 'yujie - zsh - 65x20' with standard macOS window controls (red, yellow, green buttons). The terminal shows the command 'python3 --version' being executed, resulting in the output 'Python 3.11.0'. The prompt 'yujie@qqzfq ~ %' is visible at the bottom.

```
yujie@qqzfq ~ % python3 --version
Python 3.11.0
yujie@qqzfq ~ %
```

#### 1.2 django

```
pip install django
```

A terminal window titled 'yujie — -zsh — 65x20' with standard macOS window controls (red, yellow, green buttons). The terminal shows the execution of 'python3', displaying the Python 3.11.0 version and build information. It then shows 'import django' and 'django.VERSION' returning '(4, 2, 1, 'final', 0)'. The prompt returns to 'yujie@qqzfz ~ %' after a carriage return.

```
yujie@qqzfz ~ % python3
Python 3.11.0 (v3.11.0:deaf509e8f, Oct 24 2022, 14:43:23) [Clang
13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more informa
tion.
[>>> import django
[>>> django.VERSION
(4, 2, 1, 'final', 0)
[>>> ^D
yujie@qqzfz ~ %
```

## 2 创建Hello World

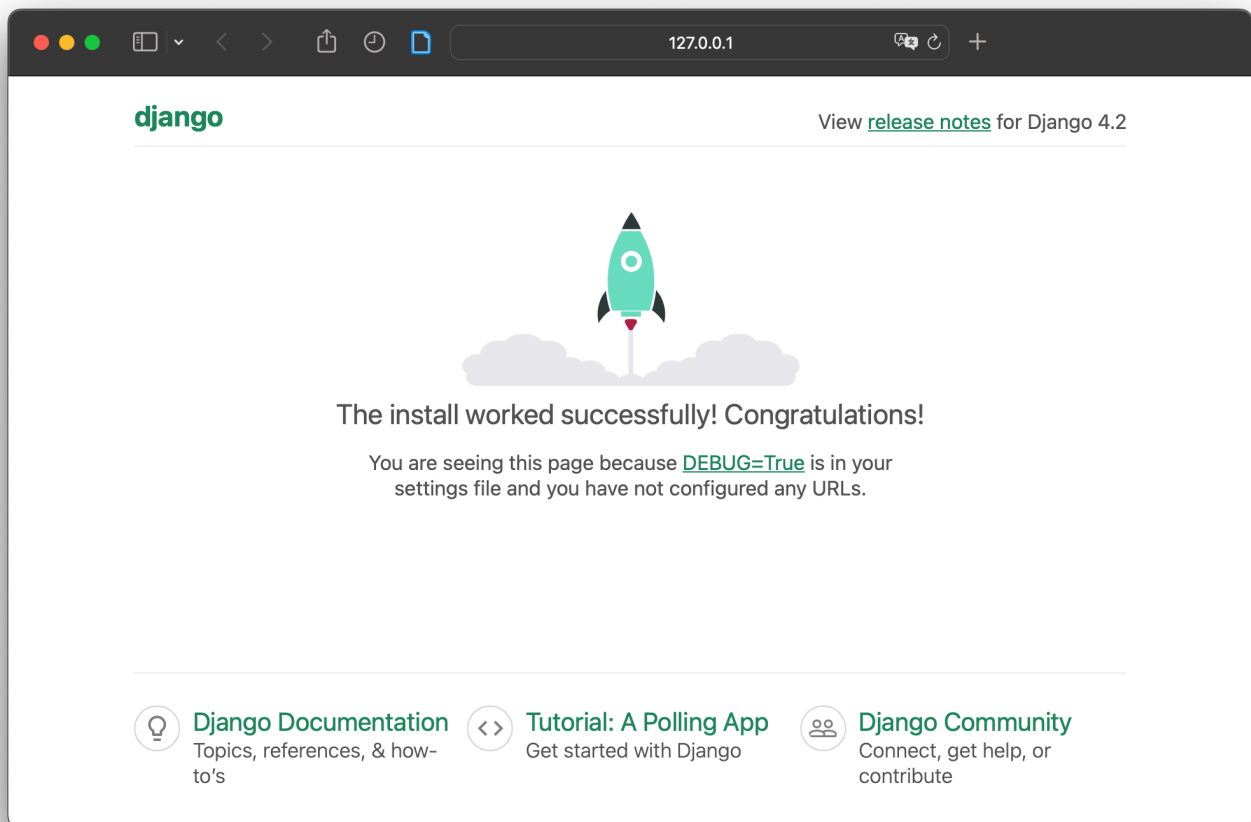
使用 django-admin 来创建 HelloWorld 项目：

```
django-admin startproject HelloWorld
```

进入 HelloWorld 目录输入以下命令，启动服务器：

```
python3 manage.py runserver
```

浏览器打开[初始](#)页面：



现在已经创建了一个空的django项目。

在先前创建的 HelloWorld 目录下的 HelloWorld 目录新建一个 views.py 文件，并输入代码：

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world ! ")
```

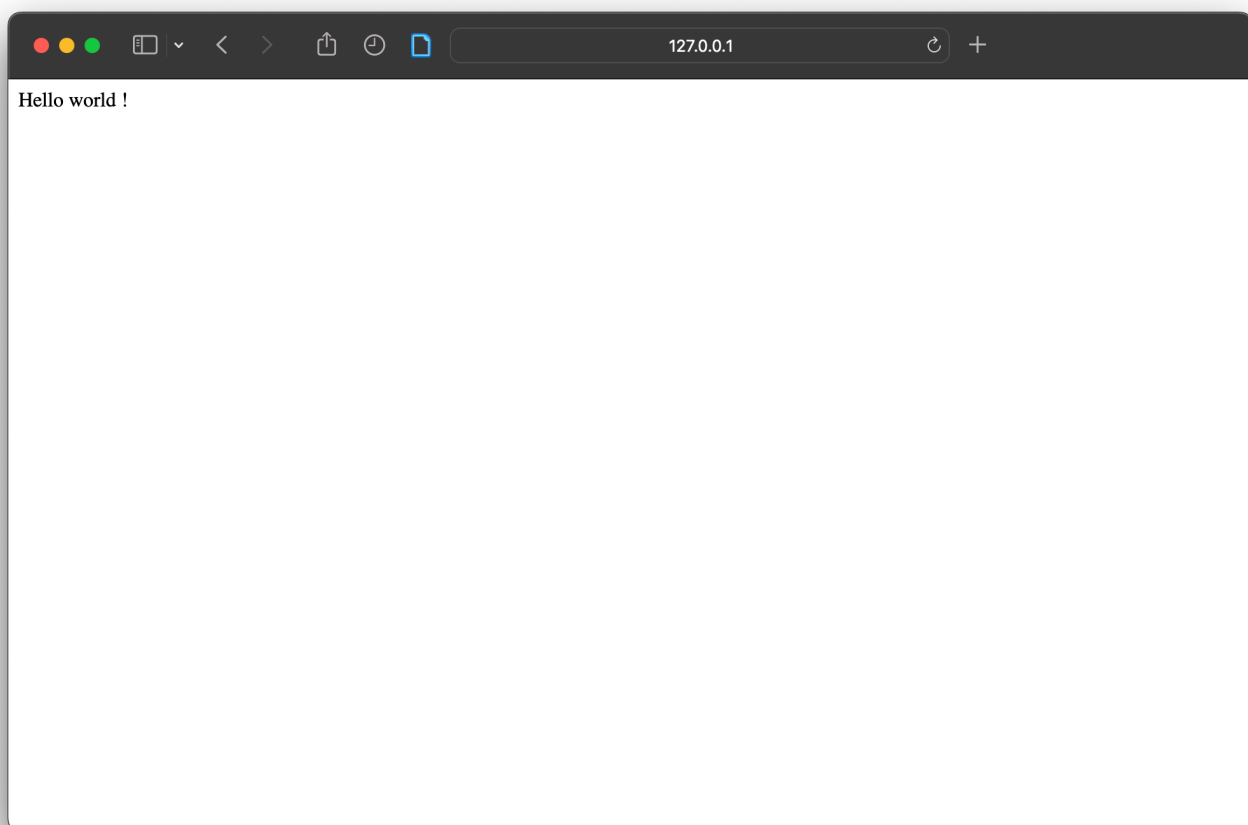
接着，绑定 URL 与视图函数。打开 urls.py 文件，修改为以下内容：

```
from django.contrib import admin
from django.urls import path

from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('hello/', views.hello),
]
```

访问[hello](#)页面：



## LAB1.2

在LAB1.2中，我基于django的官方教程，实现了一个美观、完善的问题投票系统，实现了添加问题、进行问题投票和票数统计等功能。运行方法已经写在项目的README.md中，可自行查阅。

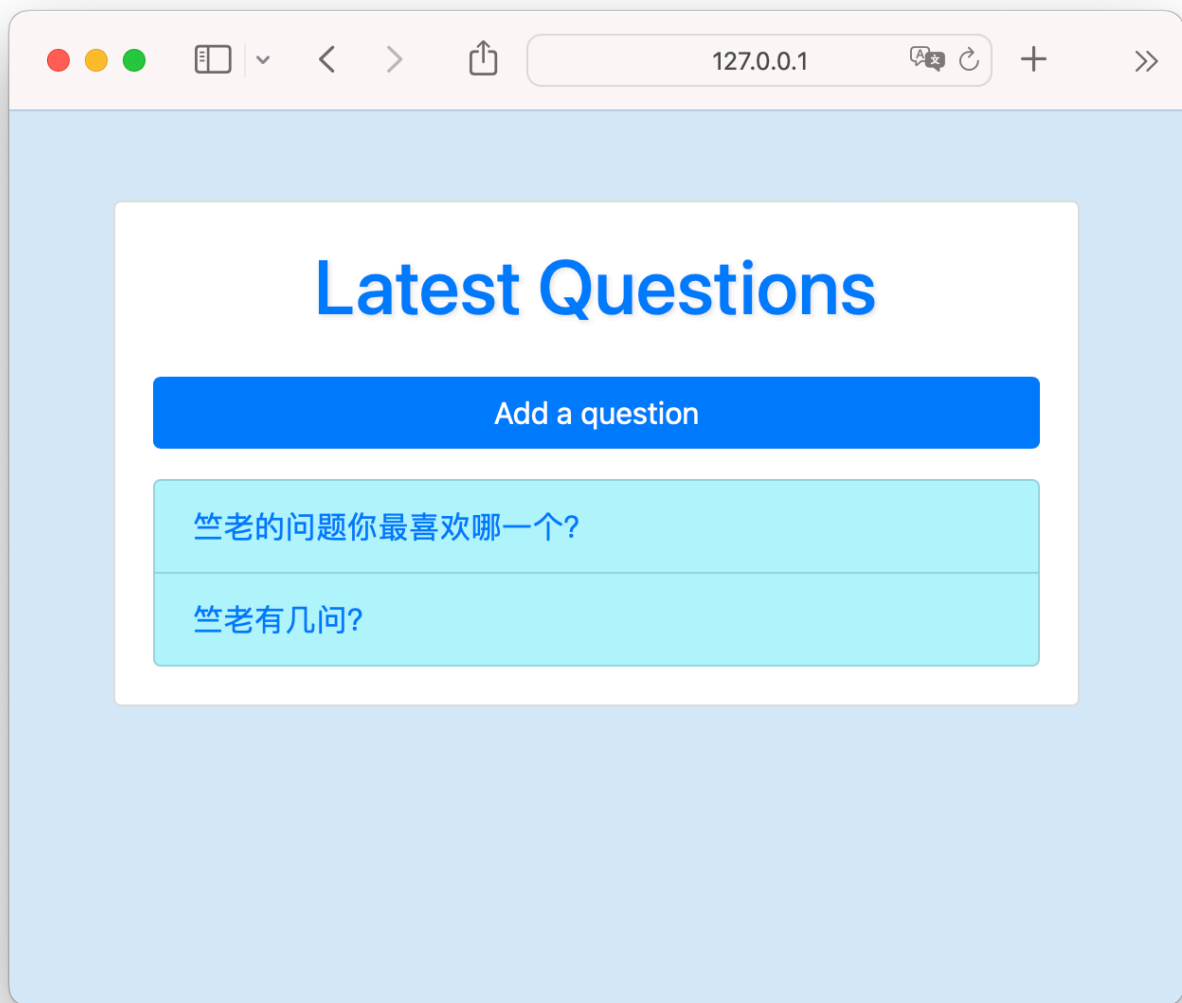
### 0 功能展示

在此web应用中，我实现了一个公开的匿名投票系统。

用户可以看到其他人发布的问题，并选择问题进行投票；投票结束后，可以看到此问题的投票结果，且可以反复多次投票；用户也可以发布自己的问题，设置发布时间和问题选项，让别人对自己的问题进行投票。

在主页面中，会按照发布时间显示已经发布过的问题，最新发布的问题位于最上方。

### 主页面



添加问题页面

127.0.0.1

+

>>

# Add a Question

Question text:

Date published:

## Choices

Choice text:

Choice text:

Choice text:

Choice text:

Add question

Back

投票页面



127.0.0.1



# 竺老的问题你最喜欢哪一个？

☐ 到浙大来做什么？

☐ 将来毕业后做什么样的人？

☐ 都喜欢

☐ 都不喜欢

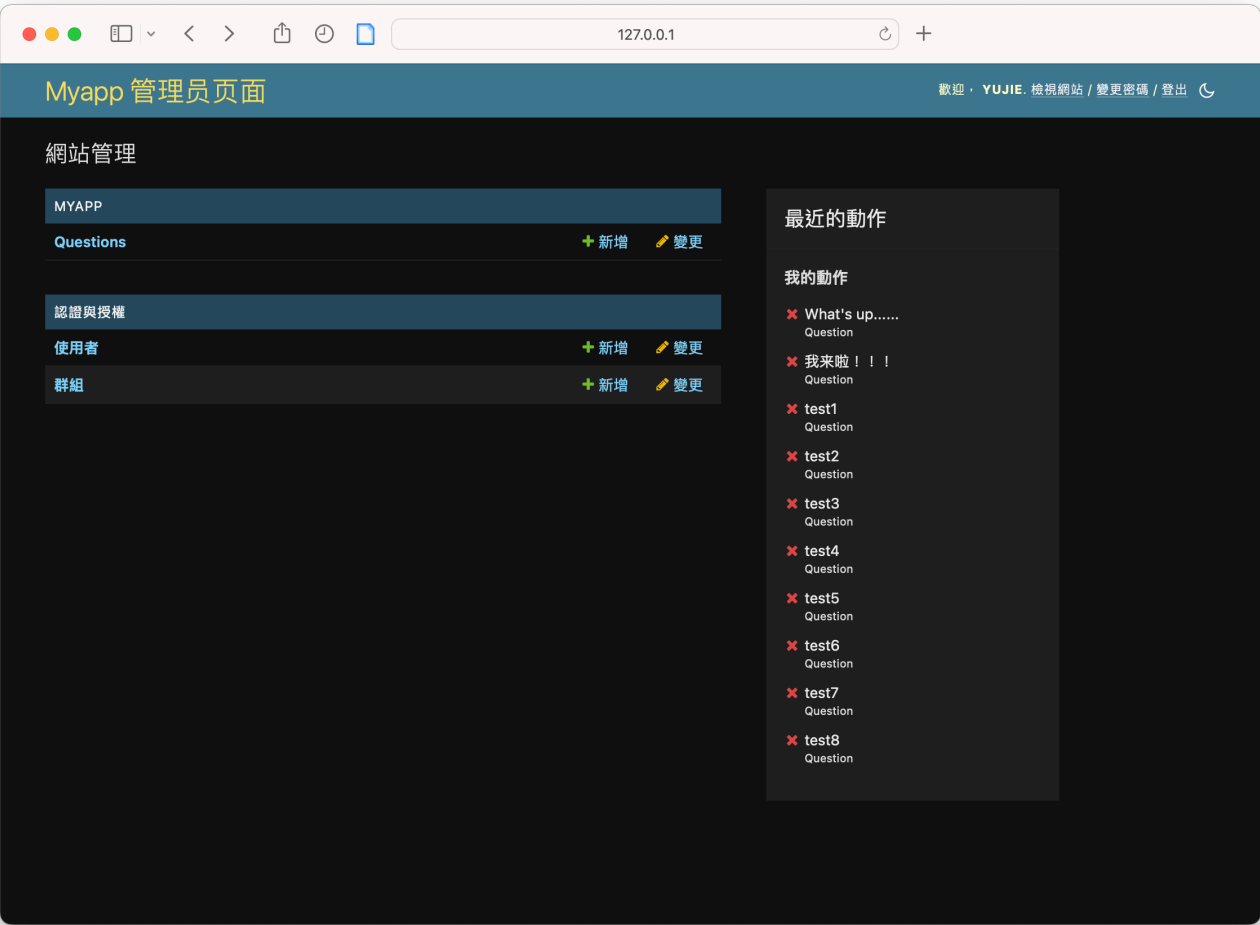
Vote

Back





# 管理员页面



## 1 技术栈

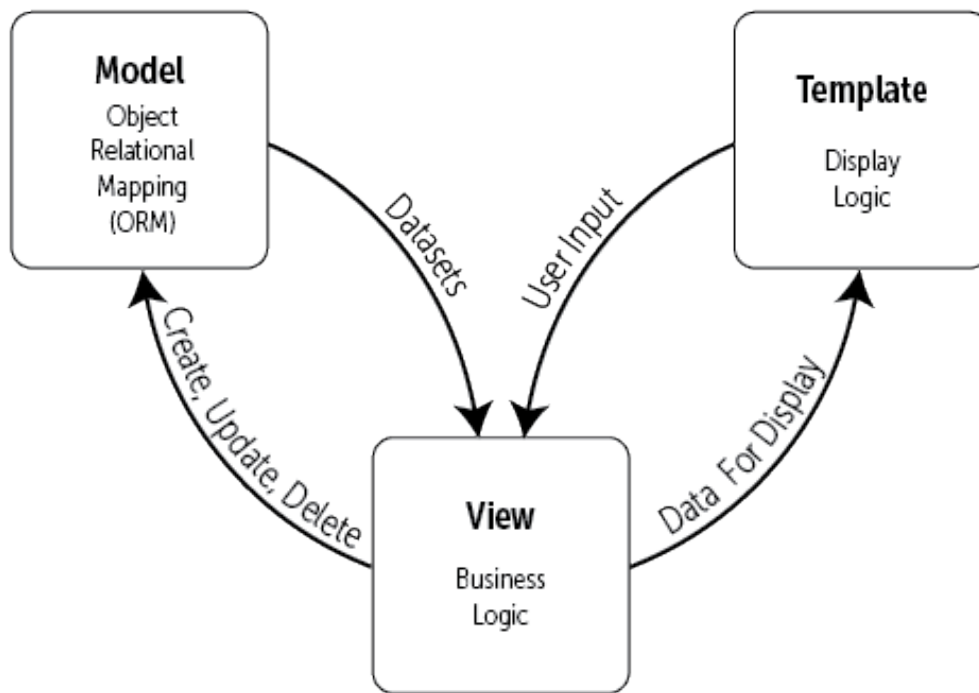
- 后端语言：python
- 前端语言：html、css、js
- web框架：django
- 数据库：sqlite3

## 3 后端设计

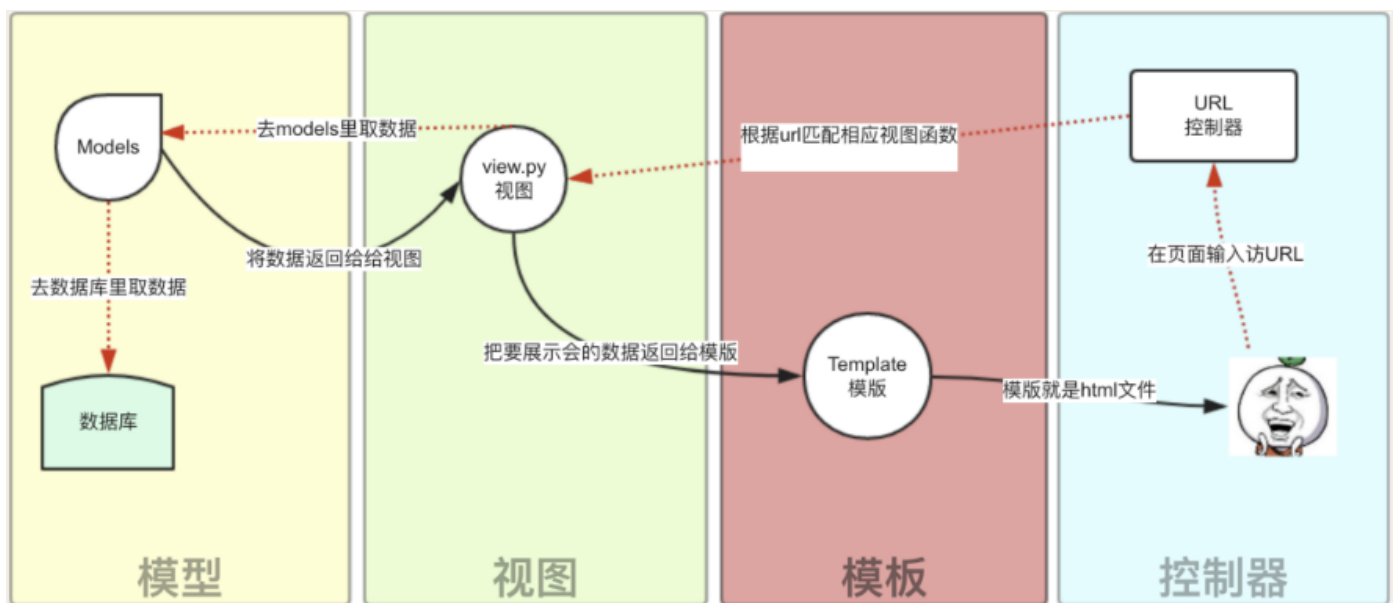
本程序基于django框架开发，采用**MTV**设计方法。django的**MTV**分别是指：

- M 表示模型（Model）：编写程序应有的功能，负责业务对象与数据库的映射(ORM)。
- T 表示模板 (Template)：负责如何把页面(html)展示给用户。
- V 表示视图（View）：负责业务逻辑，并在适当时候调用 Model和 Template。

简易图：



用户操作流程图：



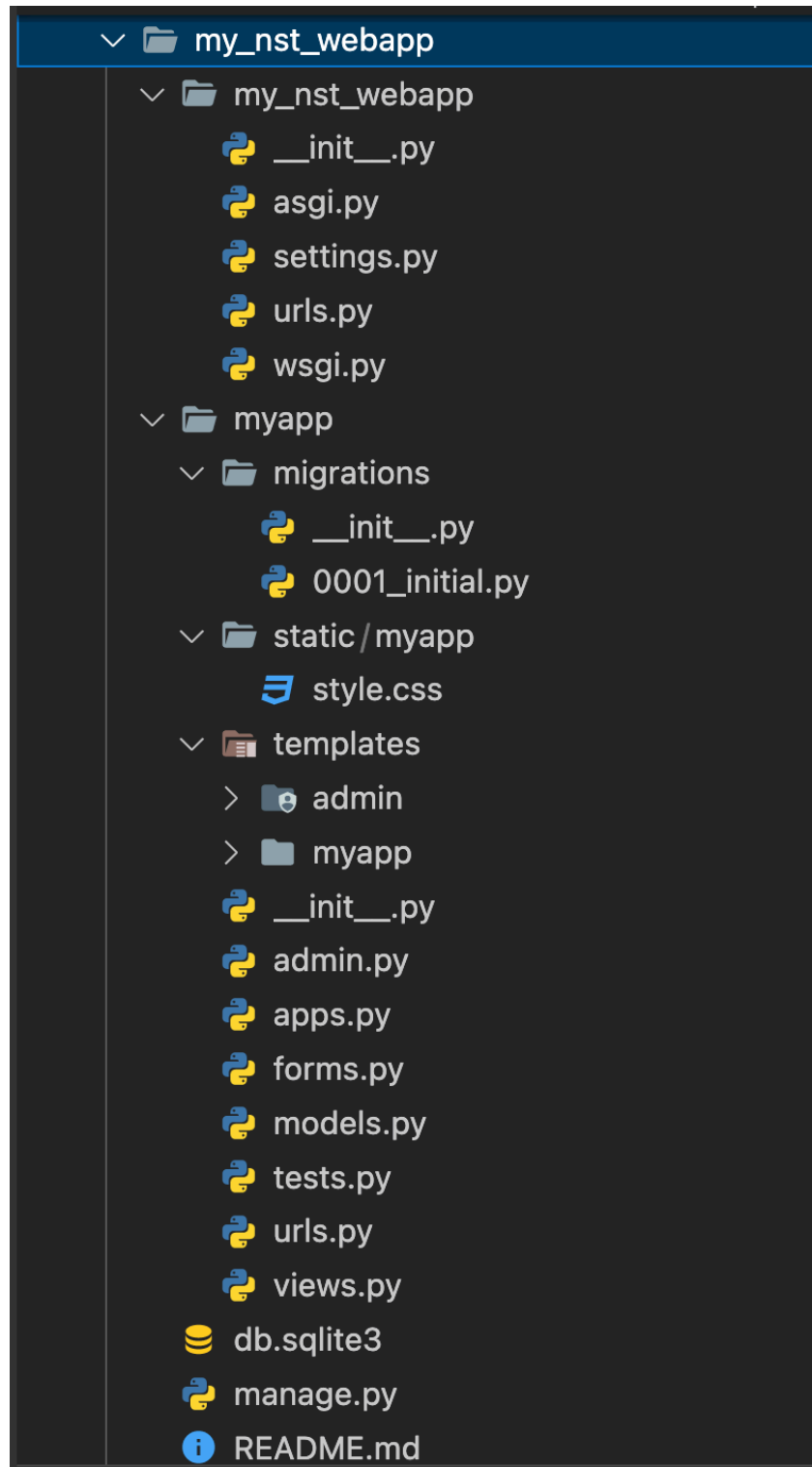
解析：

用户通过浏览器向我们的服务器发起一个请求(request)，这个请求会去访问视图函数：

- 如果不涉及到数据调用，视图函数直接返回一个模板给用户。
- 如果涉及到数据调用，那么视图函数调用模型，模型去数据库查找数据，然后逐级返回。

视图函数把返回的数据填充到模板中，最后返回网页给用户。

代码架构如下：



其中，`my_nst_webapp/my_nst_webapp` 存放整个web应用的基础设置；`my_nst_webapp/myapp` 存放我的代码，包括视图、模型、url、表单等；`my_nst_webapp/myapp/migrations` 存放数据库相关内容；`my_nst_webapp/myapp/static` 存放页面CSS；`my_nst_webapp/myapp/templates` 存放前端渲染模板。

核心是投票、发布问题两块内容，通过在视图中的处理实现：

```
# 通过前端传入问题和选项内容，处理用户在投票应用中对某个问题的投票操作。
# 首先，通过 get_object_or_404 函数尝试获取指定 question_id 的 Question 对象。如果找不到该对象，
# 将返回一个 404 错误。
# 然后，尝试从 request.POST 中获取用户选择的选项（choice）。如果没有找到或者选项不存在，将返回一个包含
# 错误信息的页面，提示用户没有选择选项。
```

# 如果成功获取到用户选择的选项, 将该选项的 votes 属性值加 1, 表示增加一票, 然后保存该选项对象。  
# 最后, 使用 HttpResponseRedirect 函数重定向到投票结果页面 (myapp:results), 并将 question.id 作为参数传递。

```
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST["choice"])
    except (KeyError, Choice.DoesNotExist):
        return render(
            request,
            "myapp/detail.html",
            {
                "question": question,
                "error_message": "You didn't select a choice.",
            },
        )
    else:
        selected_choice.votes += 1
        selected_choice.save()
        return HttpResponseRedirect(reverse("myapp:results", args=(question.id,)))
```

# 使用自建表单进行前后端数据传递, 同时进行数据库更新, 处理用户在投票应用中添加新问题和相应选项的操作。  
# 首先, 检查 request 的方法是否为 'POST'。如果是 'POST', 说明用户已经提交了表单, 需要处理表单数据。  
# a. 使用 QuestionForm 和 ChoiceForm 分别处理问题和选项的表单数据。choice\_forms 是一个列表, 包含四个 ChoiceForm 实例, 用于处理四个选项。  
# b. 检查问题表单 (form) 和所有选项表单 (choice\_forms) 是否有效。如果都有效, 继续处理。  
# c. 保存问题表单, 创建一个新的 Question 对象。  
# d. 遍历选项表单, 为每个选项设置关联的问题 (choice.question = question), 然后保存选项。  
# e. 处理完成后, 重定向到投票应用的主页 (myapp:index)。  
# 如果 request 的方法不是 'POST', 说明用户还没有提交表单, 需要显示一个空白的表单供用户填写。  
# a. 创建一个空的 QuestionForm 实例和四个空的 ChoiceForm 实例。  
# 使用 render 函数返回一个包含问题表单和选项表单的页面 (myapp/add\_question.html), 供用户填写并提交。

```
def add_question(request):
    if request.method == 'POST':
        form = QuestionForm(request.POST)
        choice_forms = [ChoiceForm(request.POST, prefix=str(
            x), instance=Choice()) for x in range(0, 4)]

        if form.is_valid() and all([cf.is_valid() for cf in choice_forms]):
            question = form.save()
            for cf in choice_forms:
                choice = cf.save(commit=False)
                choice.question = question
                choice.save()
            return redirect('myapp:index')
    else:
        form = QuestionForm()
        choice_forms = [ChoiceForm(prefix=str(x)) for x in range(0, 4)]
```

```
return render(request, 'myapp/add_question.html', {'form': form, 'choice_forms':
choice_forms})
```

其他内容则是一些简单的页面渲染，如有需要可以去源码中找。

## 4 数据库设计

本系统出于方便考虑，使用了python内建的sqlite数据库。django提供了非常方便的数据库管理功能，简单来说，对一个数据库进行修改管理只需要三步：

- 编辑 `models.py` 文件，改变模型。
- 运行 `python manage.py makemigrations` 为模型的改变生成迁移文件。
- 运行 `python manage.py migrate` 来应用数据库迁移。

我的 `models.py` 如下：

```
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    def __str__(self):
        return self.question_text

    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)

@admin.display(
    boolean=True,
    ordering="pub_date",
    description="Published recently?",
)
def was_published_recently(self):
    now = timezone.now()
    return now - datetime.timedelta(days=1) <= self.pub_date <= now


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __str__(self):
        return self.choice_text
```

## 1. Question模型：

- 包含两个字段：question\_text和pub\_date。
- question\_text字段是一个CharField，最大长度为200，用于存储问题的文本。
- pub\_date字段是一个DateTimeField，用于存储问题的发布日期。
- 定义了一个\_\_str\_\_方法，返回问题的文本。
- 定义了一个was\_published\_recently方法，用于检查问题是否在最近一天内发布。使用了Django的timezone模块来处理时区问题。
- 使用@admin.display装饰器为was\_published\_recently方法添加了一些额外的属性，如boolean、ordering和description，用于在Django管理界面中显示该方法的结果。

## 2. Choice模型：

- 包含三个字段：question、choice\_text和votes。
- question字段是一个ForeignKey，它将Choice模型与Question模型关联起来。当一个问题被删除时，on\_delete=models.CASCADE确保与之关联的所有选项也被删除。
- choice\_text字段是一个CharField，最大长度为200，用于存储选项的文本。
- votes字段是一个IntegerField，默认值为0，用于存储该选项获得的票数。
- 定义了一个\_\_str\_\_方法，返回选项的文本。

基于以上内容，一个合适的数据库便构建完成了。

# 5 前端设计

前端一共有四个页面，分别为主页面、添加问题页面、投票页面、投票结果页面。

## 5.1 主页面

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MyApp</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js">
</script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
</script>
    <link rel="stylesheet" href="{% static 'myapp/style.css' %}">
</head>
<body>
    <div class="container">
        <div class="row justify-content-center">
```

```

<div class="col-md-6">
  <div class="card my-5">
    <div class="card-body">
      <h1 class="text-center mb-4">Latest Questions</h1>
      <a href="{% url 'myapp:add_question' %}" class="btn btn-primary
btn-block mb-3">Add a question</a>
      {% if latest_question_list %}
        <div class="scrollable">
          <ul class="list-group">
            {% for question in latest_question_list %}
              <li class="list-group-item"><a href="{% url
'myapp:detail' question.id %}">{{ question.question_text }}</a></li>
            {% endfor %}
          </ul>
        </div>
      {% else %}
        <div class="alert alert-warning" role="alert">
          No queations are available.
        </div>
      {% endif %}
    </div>
  </div>
</div>
</div>
</div>
</body>
</html>

```

页面显示了一个包含最新问题的列表，以及一个用于添加新问题的按钮。

1. 引入所需的库和样式表：

- 引入Bootstrap CSS库，用于页面的基本样式。
- 引入jQuery和Popper.js库，这是Bootstrap的依赖项。
- 引入Bootstrap JavaScript库，用于页面的交互功能。
- 引入自定义的CSS样式表（位于 `myapp/style.css`）。

2. 创建一个包含Bootstrap容器、行和列的基本页面结构。这将使页面内容居中显示。

3. 在列中创建一个卡片组件，用于显示最新问题列表和添加问题按钮。

4. 在卡片的主体部分：

- 显示一个居中的标题，显示“Latest Questions”。
- 创建一个链接到 `myapp:add_question` URL的按钮，用于添加新问题。
- 使用Django模板标签检查是否有最新问题列表。如果有，显示一个带有滚动条的列表。否则，显示一个警告消息，表示没有可用的问题。

5. 使用Django模板标签遍历最新问题列表，并为每个问题创建一个列表项。列表项包含一个链接到 `myapp:detail` URL的问题文本，该URL将显示问题的详细信息。

## 5.2 添加问题页面

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add a Question</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js">
</script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
</script>
    <link rel="stylesheet" href="{% static 'myapp/style.css' %}">
</head>
<body>
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-6">
                <div class="card my-5">
                    <div class="card-body">
                        <h1 class="text-center mb-4">Add a Question</h1>
                        <form method="post">
                            {% csrf_token %}
                            {{ form.as_p }}
                            <h3>Choices</h3>
                            {% for choice_form in choice_forms %}
                                <div class="form-group">
                                    {{ choice_form.as_table }}
                                </div>
                            {% endfor %}
                            <button type="submit" class="btn btn-primary btn-block">Add
question</button>
                        </form>
                        <div class="mt-3">
                            <a href="{% url 'myapp:index' %}" class="btn btn-secondary
btn-block">Back</a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
</div>
```



```
</body>
</html>
```

显示一个添加问题的表单，用户可以在表单中输入问题和选项，然后提交表单以添加问题。

1. 引入Bootstrap和jQuery库：在 `<head>` 标签内，通过CDN链接引入了Bootstrap的CSS和JS文件，以及jQuery库。这些库用于实现页面的布局、样式和交互功能。
2. 引入自定义样式表：使用 `{% static 'myapp/style.css' %}` 引入了一个名为 `style.css` 的自定义样式表，它位于 `myapp` 应用的 `static` 文件夹内。
3. 页面结构：在 `<body>` 标签内，使用Bootstrap的 `container`、`row` 和 `col` 类创建了一个居中的表单区域。表单区域包含在一个 `card` 组件内，用于显示添加问题的表单。
4. 表单内容：使用 `<form>` 标签创建了一个表单，表单的提交方式为 `POST`。在表单内，使用 `{% csrf_token %}` 添加了一个CSRF令牌，用于防止跨站请求伪造攻击。
5. 渲染问题表单：使用 `{{ form.as_p }}` 将问题表单渲染为一组 `<p>` 标签。这将显示问题的输入字段。
6. 渲染选项表单：使用 `{% for choice_form in choice_forms %}` 循环遍历 `choice_forms` 列表，这个列表包含了多个选项表单。在循环内，使用 `{{ choice_form.as_table }}` 将每个选项表单渲染为一个表格。这将显示选项的输入字段。
7. 提交按钮：添加了一个类型为 `submit` 的按钮，用于提交表单。当用户点击此按钮时，表单将被提交，添加问题和选项。
8. 返回按钮：在表单下方添加了一个返回按钮，用于返回到应用的主页。使用 `{% url 'myapp:index' %}` 生成了主页的URL。

## 5.3 投票页面

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vote</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js">
</script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
</script>
    <link rel="stylesheet" href="{% static 'myapp/style.css' %}">
</head>
<body>
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-6">
                <div class="card my-5">
```

```

<div class="card-body">
    <h1 class="text-center mb-4">{{ question.question_text }}</h1>
    <form action="{% url 'myapp:vote' question.id %}"
method="post">

        {% csrf_token %}
        {% if error_message %}
            <div class="alert alert-danger" role="alert">
                {{ error_message }}
            </div>
        {% endif %}
        <div class="row">
            {% for choice in question.choice_set.all %}
                <div class="col-md-6 col-lg-4 mb-4">
                    <div class="card">
                        <div class="card-body">
                            <div class="form-check">
                                <input class="form-check-input"
type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
                                <label class="form-check-label"
for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label>
                            </div>
                        </div>
                    </div>
                </div>
            {% endfor %}
        </div>
        <div class="mt-3">
            <input type="submit" class="btn btn-primary btn-block"
value="Vote">
        </div>
    </form>
    <div class="mt-3">
        <a href="{% url 'myapp:index' %}" class="btn btn-secondary
btn-block">Back</a>
    </div>
</div>
</div>
</div>
</div>
</div>
</body>
</html>

```

在这个页面中，用户可以选择一个选项并提交投票。

1. 引入所需的CSS和JavaScript库，如Bootstrap和jQuery。
2. 使用Django的 `{% load static %}` 标签加载静态文件，如CSS样式表。
3. 设置HTML文档的基本结构，包括 `<head>` 和 `<body>` 标签。
4. 在 `<body>` 标签内，创建一个包含页面内容的 `<div>` 容器。

5. 使用Bootstrap的栅格系统创建一个居中的列，包含一个卡片组件，用于显示问题和选项。
6. 在卡片的 `<div class="card-body">` 中，显示问题文本，并创建一个表单，用于提交用户的投票。
7. 使用Django的 `{% csrf_token %}` 标签添加CSRF令牌，以保护表单免受跨站请求伪造攻击。
8. 如果有错误消息，使用Bootstrap的警告组件显示错误消息。
9. 使用Django的 `{% for %}` 循环遍历问题的所有选项，并为每个选项创建一个单选按钮。
10. 在表单底部，添加一个提交按钮，用于提交用户的投票。
11. 添加一个返回按钮，允许用户返回到投票应用的主页。

## 5.4 投票结果页面

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Results</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js">
</script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
</script>
    <link rel="stylesheet" href="{% static 'myapp/style.css' %}">
</head>
<body>
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-6">
                <div class="card my-5">
                    <div class="card-body">
                        <h1 class="text-center mb-4">{{ question.question_text }}</h1>
                        <ul class="list-unstyled">
                            {% for choice in question.choice_set.all %}
                                <li class="mb-2">
                                    <div class="card">
                                        <div class="card-body">
                                            {{ choice.choice_text }} -- {{ choice.votes }}
                                        </div>
                                    </div>
                                </li>
                            {% endfor %}
                        </ul>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="mt-3">
```

```

        <a href="{% url 'myapp:detail' question.id %}" class="btn
btn-primary btn-block">Vote again</a>
    </div>
    <div class="mt-3">
        <a href="{% url 'myapp:index' %}" class="btn btn-secondary
btn-block">Back</a>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>
</body>
</html>

```

显示投票结果页面。

1. 引入所需的CSS和JavaScript库，包括Bootstrap和jQuery。
2. 使用Django的 `{% load static %}` 标签加载静态文件，以便在模板中使用静态文件（如CSS）。
3. 在 `<head>` 标签中，设置页面的元数据，如字符集、视口设置和标题。
4. 在 `<body>` 标签中，创建一个包含页面内容的 `<div>` 容器。
5. 使用Bootstrap的 `row` 和 `col` 类创建一个居中的列，用于放置投票结果卡片。
6. 创建一个卡片，包含问题文本和投票结果列表。
7. 使用Django的 `{% for %}` 标签遍历问题的所有选项，并为每个选项创建一个列表项，显示选项文本和投票数。
8. 使用Django的 `{% url %}` 标签创建一个链接，允许用户重新投票或返回主页。

## 6 部署和测试

部署流程已经详细写在源码的README.md文件中

测试过程通过前端手动测试，和后端的断点，实现测试流程。