《Python程序设计基础》程序设计作品说明书

题目： Django开发Web应用程序

学院： 计算机科学与工程学院

班级：21计科01谭志峰

姓名： 谭志峰

学号：B20210102113

指导教师： 周景

起止日期：2023.11.10-2023.12.10

摘要

介绍本次设计完成的项目的概述，本文的主要内容，总结你主要完成的工作以及关键词。

本文介绍了使用Django框架进行Web应用程序开发的过程。通过该项目，我们实现了一个学习笔记应用，其中包括主题和条目的管理功能。文章主要涵盖以下内容：

1. **项目概述：** 介绍了学习笔记应用的设计目的和功能，以及采用Django框架的原因。
2. **关键特性：** 概述了学习笔记应用的主要功能，包括主页展示、主题管理、条目管理等。
3. **Django框架：** 对Django框架的基本概念进行了解释，包括模型、视图、模板和URL模式等。
4. **项目结构：** 说明了项目的文件结构和关键组件，包括模型定义、视图函数、URL模式和模板文件。
5. **前端集成：** 展示了如何使用Bootstrap库来美化应用的前端，并解决了一些与模板标签的问题。
6. **错误处理：** 分析了在开发过程中可能遇到的一些错误，并提供了相应的解决方案。
7. **部署和运行：** 提供了在本地运行项目的步骤，以及可能遇到的一些常见问题的解决方法。

通过完成这个项目，我们深入了解了Django框架的使用方法，学习了如何处理数据库、创建视图和模板，并通过实践解决了一些常见的Web开发问题。项目的源代码和文档可以作为学习Django框架的实践材料，帮助读者更好地理解Web应用程序的开发过程。

关键词： Django，Web开发，学习笔记，模型-视图-模板，Bootstrap。

第1章 需求分析

本章的内容主要包括系统的需求分析，系统主要需要实现的功能有哪些，可以帮助用户解决哪些问题等等。

1.1 引言

随着信息时代的发展，人们获取知识的途径变得更加多样化和便捷化。在学习过程中，合理管理和记录所学内容对于知识的消化和整理变得尤为重要。本项目旨在满足用户在学习过程中对知识整理的需求，提供一个方便、直观的学习笔记管理系统。

1.2 项目背景

传统的纸质笔记方式已经不能满足现代学习的要求，数字化学习笔记成为一种趋势。为了更好地帮助用户管理学习笔记，本项目将开发一个基于Web的学习笔记应用，提供主题和条目的管理功能。

1.3 功能需求

1.3.1 主页展示

- 用户能够通过系统主页查看最新的学习主题和相关条目。
- 主页将展示主题的摘要信息和最近添加的条目，方便用户快速了解学习进度。

1.3.2 主题管理

- 用户能够查看自己创建的所有主题，了解每个主题的详细信息。
- 用户可以点击主题，查看该主题下的所有条目。
- 用户可以添加新的主题，便于整理不同领域的学习内容。

1.3.3 条目管理

- 在主题详情页，用户能够查看该主题的所有条目。
- 用户可以点击条目，查看详细内容。
- 用户可以添加新的条目，记录学到的新知识或者心得体会。

1.3.4 用户认证和权限控制

- 用户需要进行注册，并通过登录验证后才能使用学习笔记应用。
- 每个用户只能查看、管理自己创建的主题和条目，确保数据的隐私性和安全性。

1.3.5 界面优化

- 添加用户注册页面，使新用户能够创建帐户。
- 添加用户登录页面，已注册用户可以登录并访问其学习笔记。
- 界面简洁明了，操作直观，提高用户体验。

1.4 非功能需求

1.4.1 用户友好性

- 界面简洁明了，操作直观，降低用户学习成本。
- 响应速度快，提高用户体验。

1.4.2 可扩展性

- 系统具备良好的可扩展性，方便后续根据用户需求进行功能的扩展和优化。

1.5 用户特征

本系统主要面向希望以数字方式管理学习笔记的用户，包括但不限于学生、自学者、研究人员等。用户具备一定的计算机应用基础，能够熟练使用Web浏览器。

1.6 系统环境

- 操作系统：支持常见的操作系统，包括Windows、Linux、MacOS等。
- 浏览器：支持现代Web浏览器，如Chrome、Firefox、Safari等。

1.7 总结

通过对项目需求的分析，我们明确了系统的主要功能和用户特征。在后续的设计和开发中，将围绕这些需求进行系统架构设计和具体功能实现。

第2章 分析与设计

Django 介绍

Django 详细介绍 Django 是一个用于构建 Web 应用程序的高级 Python Web 框架。它鼓励快速开发和干净、可重用的设计。以下是 Django 的一些主要特点和包含的文件：

主要特点：

1. **MTV 架构：** Django 使用 MTV（Model-Template-View）架构，与传统的 MVC（Model-View-Controller）有所不同。在 MTV 中，模型（Model）表示数据结构，模板（Template）用于定义如何渲染数据，视图（View）处理用户请求并返回响应。

2. **ORM（对象关系映射）：** Django 提供了强大的 ORM，允许开发者使用 Python 代码而不是 SQL 查询来定义和操作数据库模型。

3. **自带管理后台：** Django 自带了一个强大的管理后台，通过简单的配置，开发者可以轻松管理应用中的数据。

4. **表单处理：** Django 提供了表单处理的框架，简化了用户输入的验证和处理。

5. **用户认证和权限控制：** 内置的用户认证系统可以轻松地处理用户注册、登录、注销等操作，并支持权限控制。

6. **URL 映射：** Django 使用 URL 映射机制将 URL 请求分发到对应的视图函数上。

7. **中间件：** Django 中间件是一个处理 HTTP 请求和响应的钩子机制，可以用于执行诸如身份验证、日志记录等功能。

包含的文件和目录：

1. **manage.py：** Django 项目的命令行工具，用于执行多种操作，如运行开发服务器、应用数据库迁移等。

2. **settings.py：** 包含项目的设置，如数据库配置、静态文件路径等。开发者可以在这里配置项目的各种参数。

3. **urls.py：** URL 映射配置文件，定义了 URL 与视图函数之间的映射关系。

4. **wsgi.py：** 用于将 Django 项目部署到 WSGI 服务器的入口文件。
5. **asgi.py：** 用于将 Django 项目部署到 ASGI 服务器的入口文件。
6. **init.py：** 表明该目录是一个 Python 包。
7. **apps 目录：** 包含 Django 项目中的应用程序，每个应用程序通常包含模型、视图、模板等。
8. **static 目录：** 存放静态文件，如样式表、JavaScript 文件等。
9. **templates 目录：** 存放 HTML 模板文件。
10. **migrations 目录：** 存放数据库迁移文件，用于管理数据库模型的变更。
11. **media 目录：** 存放用户上传的媒体文件。
12. **tests 目录：** 存放应用程序的测试文件。
13. **venv 或 virtualenv 目录：** 存放虚拟环境，用于隔离项目的依赖。
14. **db.sqlite3：** 默认的 SQLite 数据库文件，可以根据项目需求更改为其他数据库。

这些文件和目录共同构成了一个 Django 项目的基本结构。在项目的开发过程中，开发者会根据需要创建新的应用程序、模型、视图、模板等，从而构建出完整的 Web 应用。

第2章 系统设计

2.1 系统架构

本系统采用典型的三层架构，包括用户界面层、业务逻辑层和数据访问层。

1. **用户界面层（Presentation Layer）：** 用户通过Web浏览器与系统进行交互，实现用户注册、登录、主题和条目的管理等功能。
2. **业务逻辑层（Business Logic Layer）：** 处理用户请求，进行业务逻辑的处理，包括主题和条目的增删改查，用户认证和权限控制等。
3. **数据访问层（Data Access Layer）：** 负责与数据库进行交互，实现数据的持久化存储和检索。

2.2 系统流程

2.2.1 用户注册流程

1. 用户访问注册页面。
2. 用户填写注册表单，包括用户名和密码。
3. 用户提交表单。
4. 系统验证表单数据的有效性，包括用户名是否已经存在。
5. 如果验证通过，系统创建新用户并将用户信息存储到数据库。
6. 用户注册成功，系统自动登录用户，并跳转到主页。

2.2.2 用户登录流程

1. 用户访问登录页面。

2. 用户填写登录表单，包括用户名和密码。
3. 用户提交表单。
4. 系统验证用户身份。
5. 如果验证通过，系统登录用户，并跳转到主页。

2.2.3 主题和条目管理流程

1. 用户登录后，访问主页。
2. 主页展示最新的学习主题和相关条目。
3. 用户可以点击主题，查看该主题下的所有条目。
4. 用户可以添加新的主题，记录不同领域的学习内容。
5. 在主题详情页，用户可以查看该主题的所有条目。
6. 用户可以点击条目，查看详细内容。
7. 用户可以添加新的条目，记录学到的新知识或者心得体会。

2.3 系统模块

系统模块主要包括以下几个部分：

1. **用户管理模块：** 负责用户的注册、登录、注销等操作。
2. **主题管理模块：** 处理与主题相关的操作，包括主题的创建、查看、编辑和删除。
3. **条目管理模块：** 处理与条目相关的操作，包括条目的创建、查看、编辑和删除。
4. **认证和权限控制模块：** 负责用户认证，确保用户只能查看、管理自己创建的主题和条目。
5. **界面优化模块：** 提供用户友好的界面，包括注册、登录、主题和条目的展示和交互。

2.4 数据库设计

2.4.1 Topic（主题）表

- `id`: 主键，自增长的整数。
- `text`: 主题的文本内容，字符型，最大长度200。
- `date_added`: 记录主题创建的日期和时间。
- `owner`: 外键，关联到用户表，表示主题的创建者。

2.4.2 Entry（条目）表

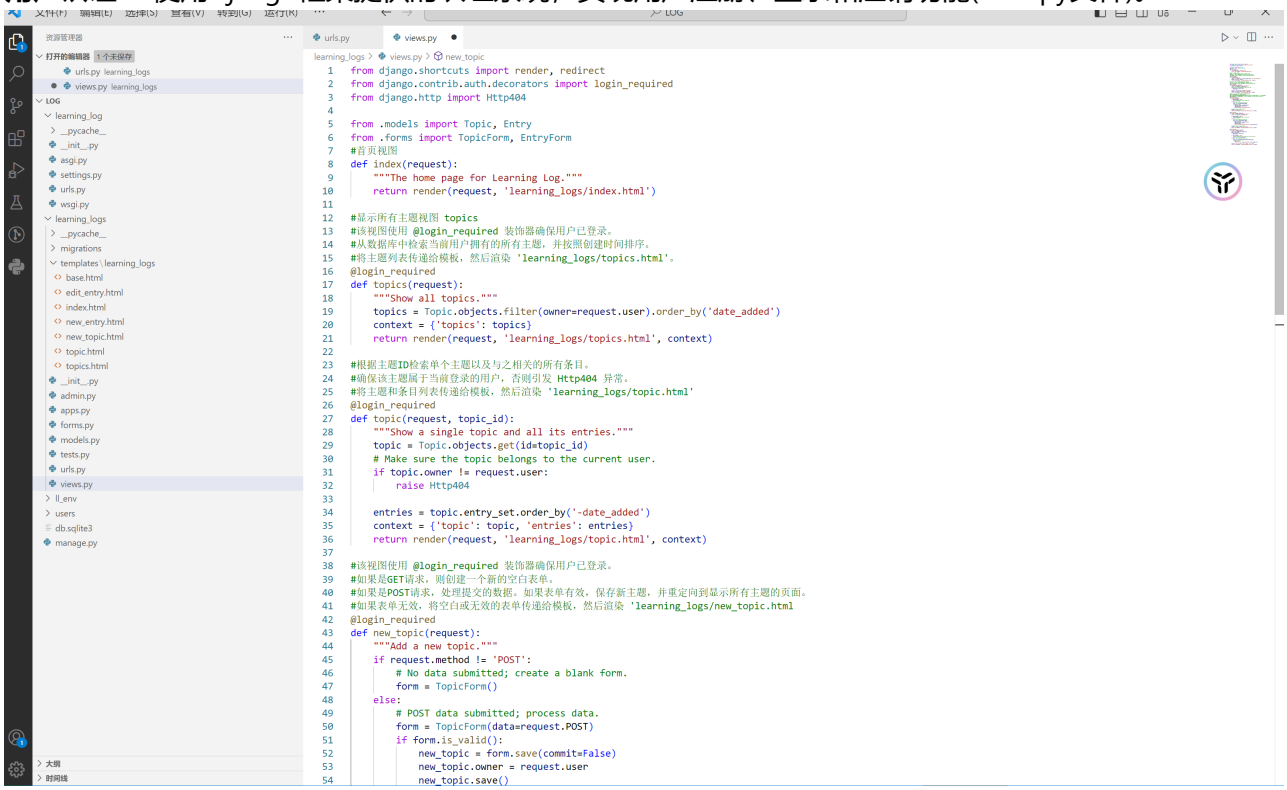
- `id`: 主键，自增长的整数。
- `topic`: 外键，关联到主题表，表示条目所属的主题。
- `text`: 条目的文本内容，文本型。
- `date_added`: 记录条目创建的日期和时间。

2.4.3 User（用户）表

- `id`: 主键，自增长的整数。
- `username`: 用户名，字符型，最大长度150。
- `password`: 密码，使用加密算法存储。
- 其他用户信息，根据实际需求添加。

2.5 关键实现

1. **用户认证：** 使用Django框架提供的认证系统，实现用户注册、登录和注销功能(view.py文件)。



```python
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.http import Http404

from .models import Topic, Entry
from .forms import TopicForm, EntryForm
#首页视图
def index(request):
    """The home page for Learning Log."""
    return render(request, 'learning_logs/index.html')

#显示所有主题视图 topics
#该视图使用 @login_required 装饰器确保用户已登录。
#从数据库中检索当前用户拥有的所有主题，并按照创建时间排序。
#将主题列表传递给模板，然后渲染 'learning_logs/topics.html'。
@login_required
def topics(request):
    """Show all topics."""
    topics = Topic.objects.filter(owner=request.user).order_by('date_added')
    context = {'topics': topics}
    return render(request, 'learning_logs/topics.html', context)

#根据主题ID检索单个主题以及与之相关的所有条目。
#确保该主题属于当前登录的用户，否则引发 Http404 异常。
#将主题和条目列表传递给模板，然后渲染 'learning_logs/topic.html'
@login_required
def topic(request, topic_id):
    """Show a single topic and all its entries."""
    topic = Topic.objects.get(id=topic_id)
    # Make sure the topic belongs to the current user.
    if topic.owner != request.user:
        raise Http404

    entries = topic.entry_set.order_by('-date_added')
    context = {'topic': topic, 'entries': entries}
    return render(request, 'learning_logs/topic.html', context)

#该视图使用 @login_required 装饰器确保用户已登录。
#如果是GET请求，则创建一个新的空白表单。
#如果是POST请求，处理提交的数据。如果表单有效，保存新主题，并重定向到显示所有主题的页面。
#如果表单无效，将空白或无效的表单传递给模板，然后渲染 'learning_logs/new_topic.html
@login_required
def new_topic(request):
    """Add a new topic."""
    if request.method != 'POST':
        # No data submitted; create a blank form.
        form = TopicForm()
    else:
        # POST data submitted; process data.
        form = TopicForm(data=request.POST)
        if form.is_valid():
            new_topic = form.save(commit=False)
            new_topic.owner = request.user
            new_topic.save()
```

2. **权限控制：** 利用Django框架的装饰器或Mixin，确保用户只能查看、管理自己创建的主题和条目。（以显示单个主题及其相关的所有条目topic为例）

首先通过 Topic.objects.get(id=topic_id) 检索特定ID的主题。接着，通过检查 topic.owner 是否等于 request.user，确保该主题属于当前登录的用户。如果不属于当前用户，则引发 Http404 异常，表明请求的主题无效或不允许访问。

然后，通过 topic.entry_set.order_by('-date_added') 获取该主题的所有条目，按照日期逆序排序，以便在页面上以合适的顺序显示。最后，通过 render 函数将主题和条目列表传递给模板，并指定要渲染的模板文件为 'learning_logs/topic.html'。这样，用户请求该主题的页面时，将看到与之相关的所有条目以及其他主题信息。

```python
@login_required
def topic(request, topic_id):
    """Show a single topic and all its entries."""
    topic = Topic.objects.get(id=topic_id)
    # Make sure the topic belongs to the current user.
    if topic.owner != request.user:
        raise Http404

    entries = topic.entry_set.order_by('-date_added')
    context = {'topic': topic, 'entries': entries}
    return render(request, 'learning_logs/topic.html', context)
```

3.**数据库操作：** 使用Django的ORM(Object-Relational Mapping)进行数据库操作，实现对主题和条目的增删改查（model.py文件）。 代码定义了两个Django模型：`Topic`和`Entry`，分别用于表示学习笔记应用中的主题和

条目。

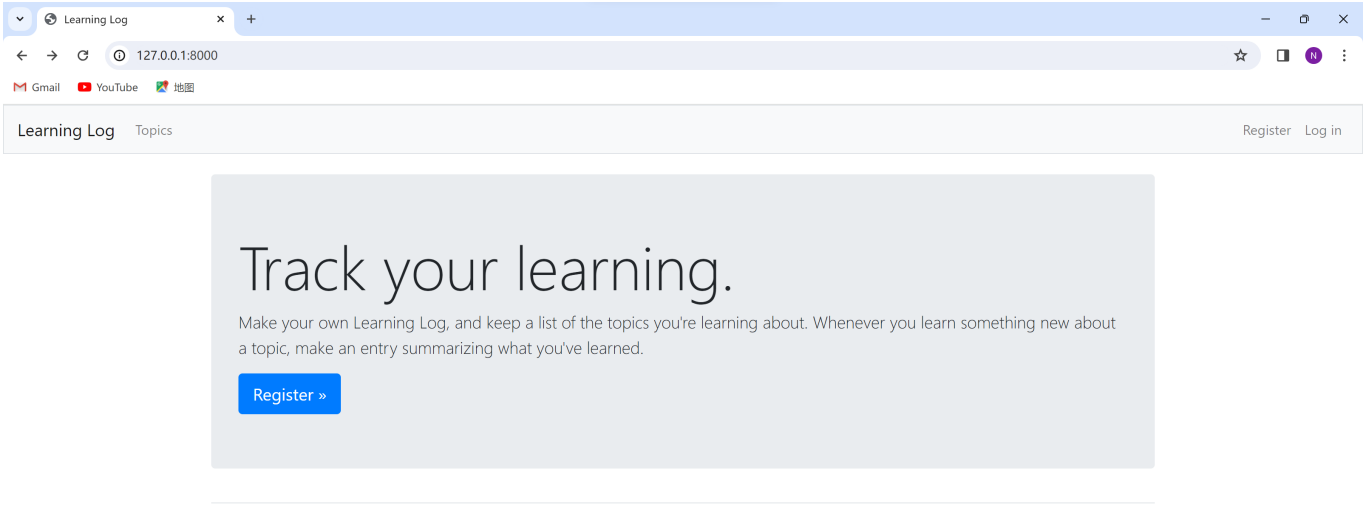
```python
from django.db import models
from django.contrib.auth.models import User

class Topic(models.Model):
    """A topic the user is learning about."""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(auto_now_add=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        """Return a string representation of the model."""
        return self.text

class Entry(models.Model):
    """Something specific learned about a topic."""
    topic = models.ForeignKey(Topic, on_delete=models.CASCADE)
    text = models.TextField()
    date_added = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name_plural = 'entries'

    def __str__(self):
        """Return a string representation of the model."""
        return f"{self.text[:50]}..."

```

4. **界面优化：** 使用Django模板语言和Bootstrap等前端框架，设计用户友好的界面，提高用户体验。

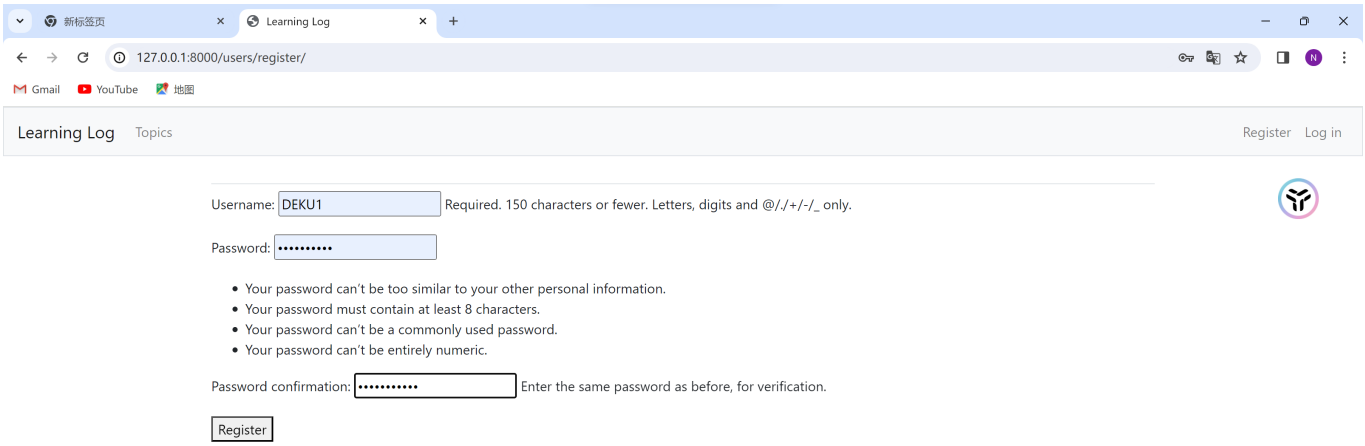
通过以上设计，系统具备良好的可扩展性和维护性，便于后续根据需求进行功能扩展和优化。在实现过程中，充分利用Django框架提供的现成功能，提高了开发效率和代码质量。

第3章 软件测试

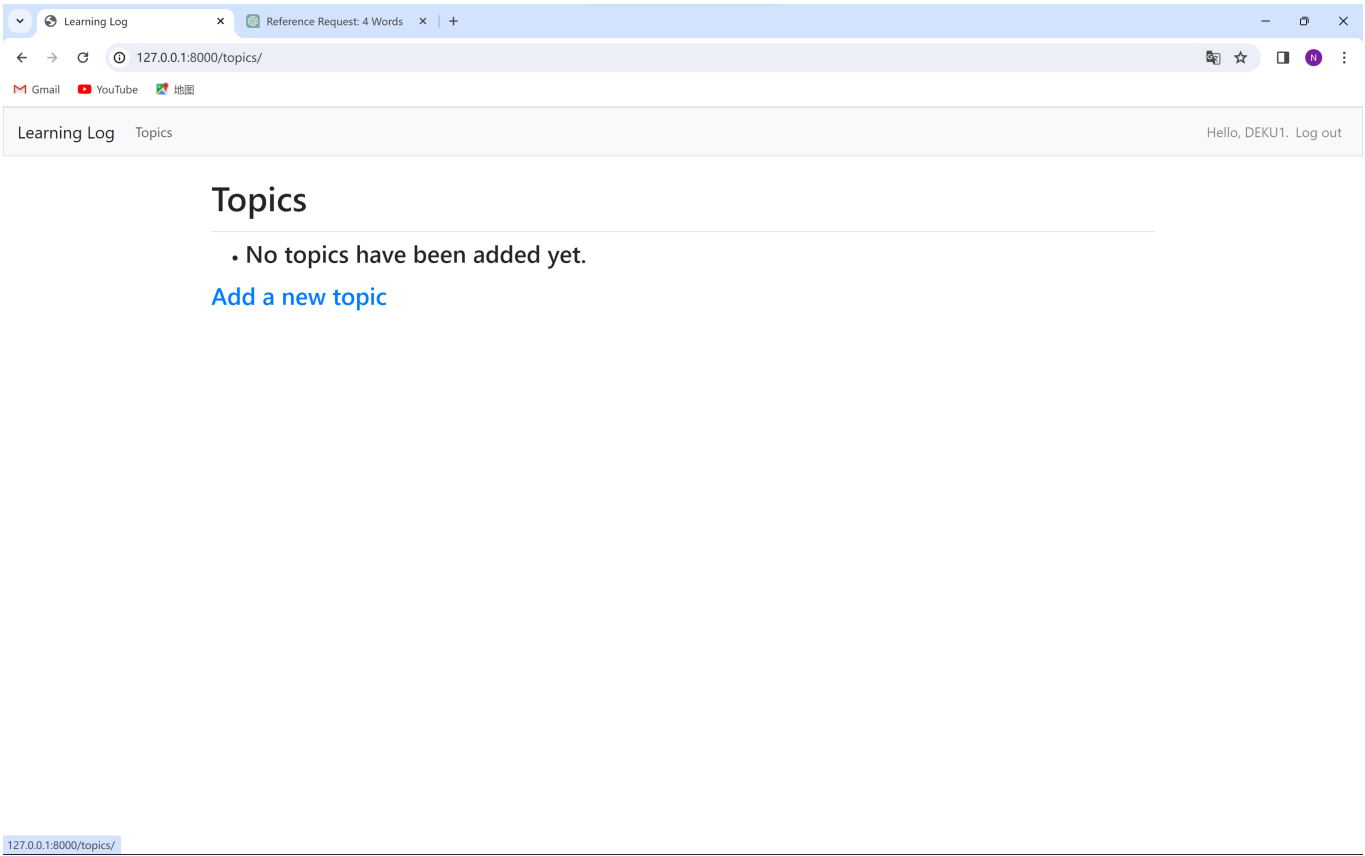
本章的内容主要是对项目的总结，项目主要实现了哪些功能，达到了哪些目标，哪些不足之处，可以如何改进。 功能实现和目标达成情况： 项目成功实现了一个基于Django框架的学习笔记应用，包括主题和条目的管理功能。主页展示、主题管理、条目管理等功能得到了有效实现。 实现的web界面如下:



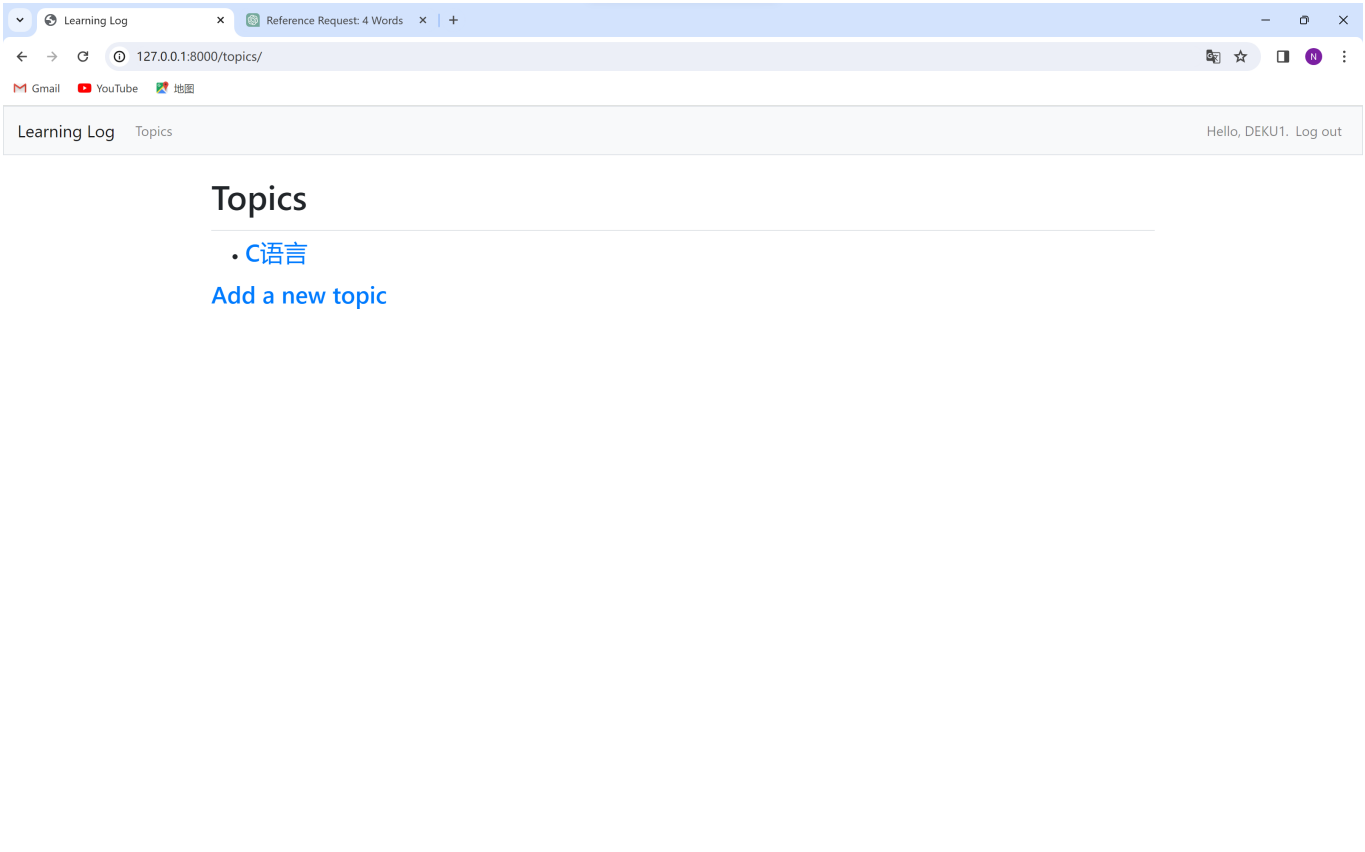
(1)通过点击Register,用户可实现账号的注册 注册账户名为DEKU1 密码为Mm20218308



(2)点击topics,进入到笔记页面，提示：No topics have benn added yet

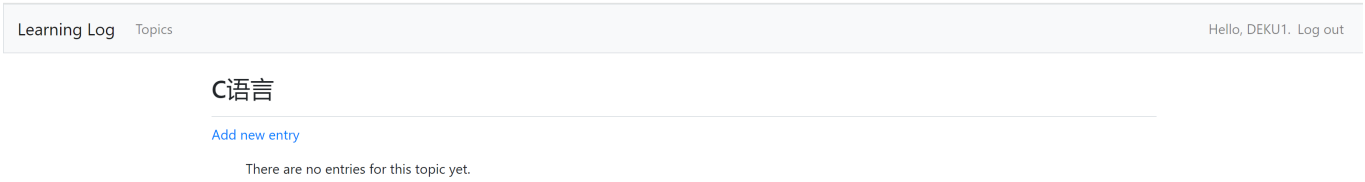


(3)点击Add a new topic,实现笔记的添加 添加topics为C语言



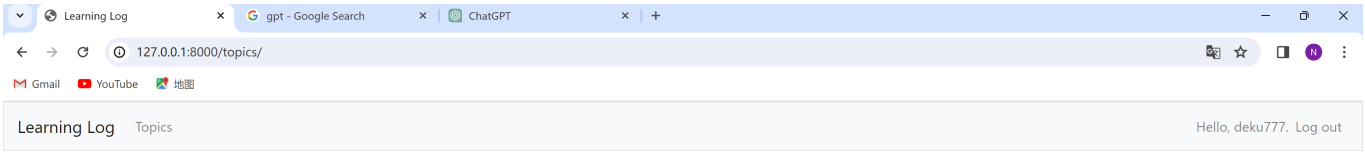
---

(4)点击C语言，显示未添加entry 可以点击Add entry 添加笔记内容



显示效果如下

(5)重复上述流程，创建新用户为deku777,登录，发现并不能看到之前DEKU1用户创建的C语言topics

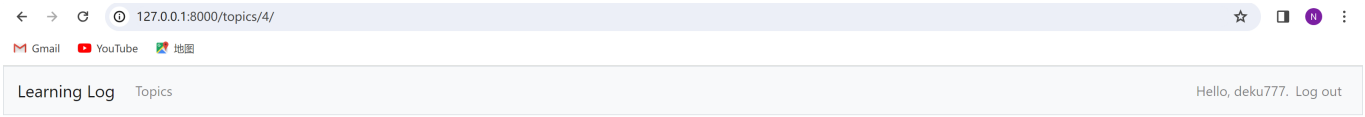


Topics

- No topics have been added yet.

Add a new topic

创建自己的笔记名为python



Python

Add new entry

Nov 22, 2023 09:05 edit entry

Python（Python编程语言）是一种高级、通用、解释型的编程语言。由荷兰程序员Guido van Rossum于1989年在Centrum Wiskunde & Informatica (CWI)创建，并于1991年发布了第一个公开版本（Python 1.0）。Python的设计哲学强调代码的可读性和简洁的语法，使其成为初学者学习编程的优秀选择，同时也被广泛用于各种领域的软件开发。

以下是Python的一些重要特点：

1. **易读性强：** Python的语法设计简单直观，使用空白符（缩进）而非花括号，使得代码更具可读性。

2. **高级语言：** Python是一种高级编程语言，具有自动内存管理和动态类型等特性，使得编程更加灵活，减少了程序员的工作负担。

3. **面向对象：** Python支持面向对象编程，允许定义类和对象，使用面向对象的方法进行编程。

4. **跨平台：** Python可以在多个操作系统上运行，包括Windows、Linux、和Mac OS等。

5. **强大的标准库：** Python内置了大量的库和模块，涵盖了文件处理、网络通信、数据库接口、图形用户界面等多个领域。

6. **广泛应用领域：** Python被广泛用于Web开发、数据科学、人工智能、自动化测试、游戏开发等多个领域。

7. **社区支持：** Python拥有强大的社区支持，有大量的第三方库和框架可供开发者使用，同时有活跃的社区论坛和文档支持。

Python的版本更新较快，目前主要有Python 2和Python 3两个主要的版本分支。由于Python 2于2020年停止官方支持，新项目和维护项目都应该使用Python 3。 Python 3引入了一些语法改进和新特性，同时解决了Python 2中的一些设计缺陷。

总体而言，Python是一门功能强大、易学易用的编程语言，适用于各种规模的项目和应用。

第4章 总结

《Python入门到实践》第三版，此书所用到的工具版本分别是：

- python3.5 (我用的是python3.13)
- django4.1 (我用的是dijango4.2.7)
- bootstrap5 (我用的是bootstrap4)

以下是一些常见问题及解决方案：

1. 外键定义时缺少 'on_delete' 参数

问题描述：

```
topic = models.ForeignKey(Topic)
# 导致错误：TypeError: __init__() missing 1 required positional argument:
'on_delete'
```

原因： 在 Django 2.0 之后，定义外键时需要明确指定 'on_delete' 参数，以避免数据不一致问题。

解决方案：

```
topic = models.ForeignKey(Topic, on_delete=models.CASCADE)
```

2. URL 配置中的 'namespace' 参数问题

问题描述：

```
url(r'', include('learning_logs.urls'), namespace='learning_logs'))
# 导致错误：TypeError: url() got an unexpected keyword argument 'namespace'
```

原因： 在 Django 2.0 之后，URL 配置的 'namespace' 参数不再使用。

解决方案： 修改 URL 配置，去掉 'namespace' 参数：

```
path('', include(('learning_logs.urls', 'learning_logs'))),
```

3. 导入 'login' 函数的问题

问题描述：

```
from django.contrib.auth.views import login
# 导致错误：ImportError: cannot import name 'login'
```

原因： 模块导入的写法发生变化。

解决方案： 使用新的导入方式：

```
from django.contrib.auth.views import LoginView
```

4. 导入 'reverse' 函数的问题

问题描述：

```
from django.core.urlresolvers import reverse
# 导致错误：ModuleNotFoundError: No module named 'django.core.urlresolvers'
```

原因： 在 Django 2.0 之后，导入路径发生变化。

解决方案： 使用新的导入方式：

```
from django.urls import reverse
```

6. URL 配置中的 'url' 函数问题

问题描述：

```
url(r'^topics/$', views.topics, name='topics'),
# 导致警告：WARNINGS:?: (2_0.W001) Your URL pattern '^topics/$' ...
```

原因： Django 2.0 后，URL 配置的写法发生了改变。

解决方案： 修改为新的写法：

```
re_path(r'^topics/$', views.topics, name='topics'),
```

第5章 参考文献

1. Django Documentation
2. Bootstrap Documentation
3. 《Python 从入门到实践》第二版