大连理工大学计算机学院

程序设计训练实验报告

实 验 名 称 ： Web 实验室工具管理平台

学 生 姓 名 ： 李英平

学 生 学 号 ： 201685040

联 系 方 式 ： 13941475810

Web 实验室工具管理平台

实验简介

近年来，物联网技术快速发展。借着物联网的东风，很多原有的生活问题有了基于物联网技术的解决方案。智慧家居、智慧城市、智慧交通等解决方案层出不穷，物联网技术使得很多领域的人力物力得到解放，极大地便利了人们的生活。

我的作品灵感来自于这两学年上实验课的体会。实验技能是要靠动手练习的，实验课上的时间并不能够很好的锻炼同学的动手能力。这学期做物理实验时，旁边一名电信学部的同学连示波器都不会使用，而我们专业此前已经做过了很多电学实验，可见，仅凭课堂时间，有的同学并不能得到很好的实践锻炼。同时，有些实验在实验前，同学们并不能对实验仪器和操作有直观的印象，这使得预习效果并不能很好，我了解到我校化环生学部引入了虚拟现实和手机模拟等技术来方便同学预习，但是手机的模拟并不能起到很好的效果，譬如滴定实验，App 的仿真很难让学生感受出真实的滴定点附近的实验现象和操作。最好的办法就是让同学们能够自由时间进入实验室去操作设备体会实验。

然而，这样的“半开放”的实验室使得人员管理、设备管理等等有很多麻烦，所以我准备设计一套结合物联网技术的智慧实验室管理系统。我联系了创新创业学院物联网应用工坊的王老师，组建了一个团队，我负责设计这个系统的软件部分。

这个网站实现了管理系统的一个核心功能，对放在柜子里的物品的借还和预约功能。网站设计并实现了注册、登陆、借还、预约、开关控制的功能，并实现了与硬件端通信的 API 接口和二维码页面。

实验目的

1. 掌握 Web 框架 Flask 的基本用法。
2. 熟悉前端语言，以及 Bootstrap 框架。
3. 熟悉 Python 语法和工具 pip、virtualenv 的使用。
4. 熟悉 Python 集成开发环境 PyCharm 的使用。
5. 掌握在服务器上网站的部署和发布操作。

实验相关原理与技术

本实验中涉及到的技术栈如下：

服务器端：Web 服务器 NginX，中间件 Gunicorn，数据库 MySQL；后端：Web 框架 Flask 及其拓展，ORM 框架 SQLAlchemy，模板渲染引擎 Jinja2；前端：Bootstrap 框架，CSS 框架 Sass。

此外，本实验还涉及了 TCP/IP 协议的一些知识点。譬如，HTTP 请求、端口监听等。

实验环境

实验编程环境：

PyCharm Professional 2018.1.4 版本

DataGrip 2018.1.5 版本

Pingendo 4.0 版本

Anaconda3 5.1.0 版本

实验服务器环境：

腾讯云服务器标准型 S2 主机

Ubuntu Server 16.04.1 LTS 64 位操作系统

服务器环境已部署服务：

MySQL、NginX、Gunicorn

运行环境：

Python 3.5+版本

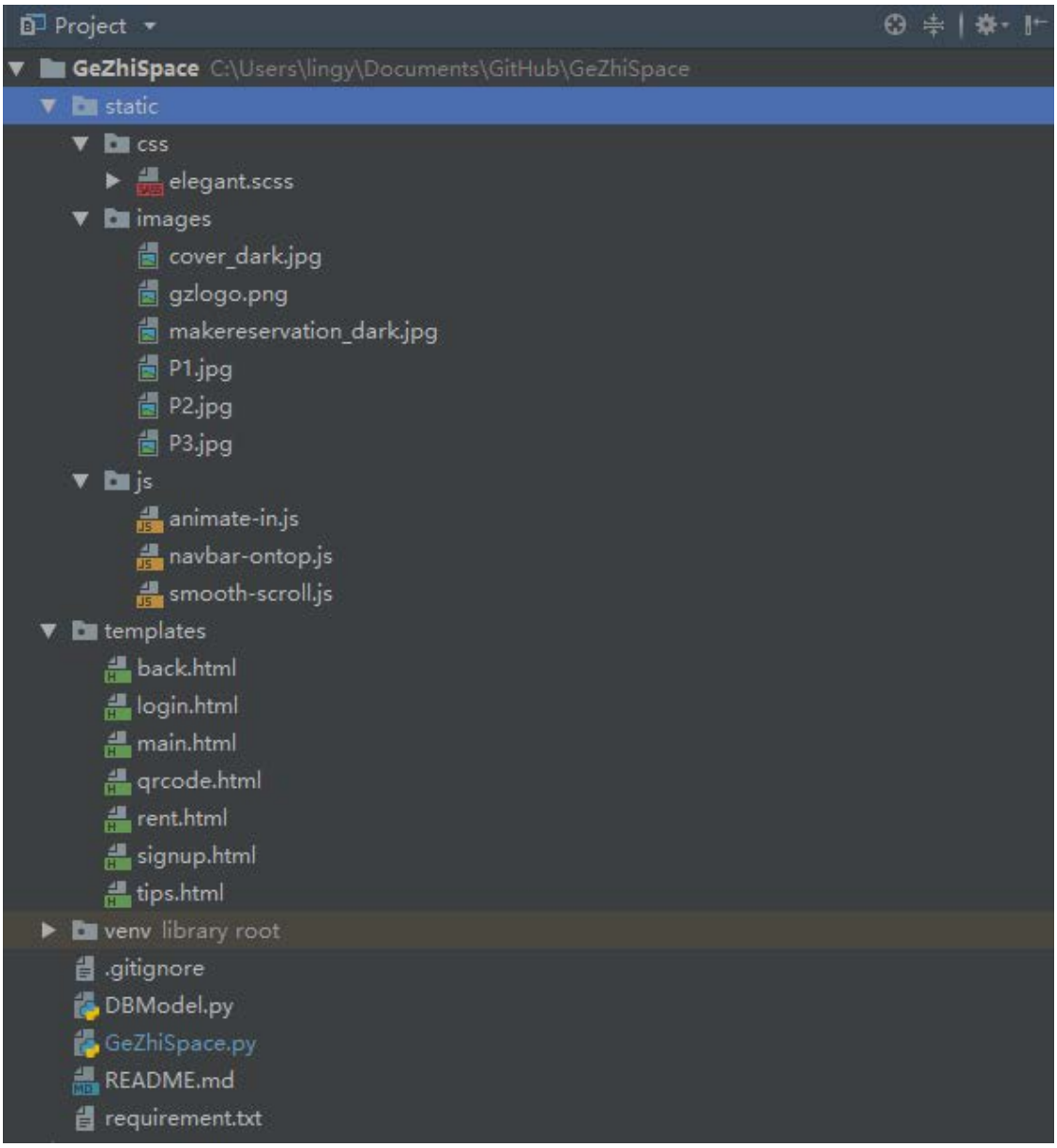
使用 pip install -r requirement.txt 命令安装必要依赖包。

代码中数据库等配置可以使用我的数据库不做修改。

登陆可以使用学号 201685040 密码 123456 登陆验证功能。

实验方案与过程

第一部分：目录结构



- static 文件夹存放网页所加载的 js、css 和图片文件。
- templates 文件夹存放网页模板文件，jinja2 默认在此目录下寻找文件。
- venv 文件夹是我的实验虚拟环境，.gitignore 和 README.md 是 Github 仓库的文件，上交代码里没有此三者。
- DBModel.py 定义数据类；GeZhiSpace.py 是应用入口，定义后端逻辑。
- requirement.txt 是项目必要依赖库的列表，在运行代码前，需要使用 pip install -r requuirement.txt 命令安装必要依赖，否则会产生 ImportError。

第二部分：后端逻辑的实现

1. 功能分析及解决方案简述

本实验项目后端主要需要实现三类需求，第一类是访问 URL 后的页面推送请求；第二类是对数据库操作的代理；第三类是提供给单片机硬件部分的交互接口。

对于第一类需求，Flask 框架提供了注册静态路由和动态路由的装饰器，可以处理 URL 请求。此外，Flask 提供了基于 Jinja2 模板引擎的网页渲染功能，可以通过 Flask 提供的 render_template()函数，将设计好的 HTML 页面传递浏览器显示。

对于第二类需求，可以使用 ORM 技术，将数据库映射为 Python 对象，绕过 SQL 的书写，使用 SQLAlchemy 提供的更高效的方法实现数据库的增删改查操作。

对于第三类需求，我设计了两种交互方式，第一种是通过 API 接口，作为单片机对数据库查询的代理。第二种是进行借还操作的时候，通过生成二维码，来把借用人和借用的物品所在柜子号传递给单片机。

2. 路由列表

URI	HTTP 方法	必要参数	路由作用
/	GET	无	返回登陆页面
/main	GET	Stedent id	返回功能界面
/signUpPage	GET	无	返回注册界面
/CreateQRCode	GET	Student ID Box ID	将两个 ID 打包成 json 生成二维码并写入数据库
/powerControl	GET	Student ID PowerBar ID Connect	改变数据库中开关通断的值
/orderControl	GET	Student ID Box ID Control	改变柜子（工具）的预约状态
/back	GET	Student ID	返回归还界面
/backQRCode	GET	Student ID Box ID	将两个 ID 打包成 json 生成二维码并写入数据库
/signUp	POST	Student ID name password Phone Number	将注册信息写入数据库
/login	POST	Student ID	判断用户
/rent	GET	无	返回可借用工具列表
/thisPersonIsAllowed/<int:student_id>	GET	无（参数通过 URI 传递）	供单片机查询
/thisPowerbarIsNeededToUsed/<int:power_bar_id>	GET	无（参数通过 URI 传递）	供单片机查询
/thisPersonOpenThisBoxIsAllowed		无（参数通过 URI 传递）	供单片机查询

3. 代表性路由实现说明

- **普通页面推送请求路由**

```
@app.route('/')
def sign_in():
    return render_template("login.html", judgeLoginPath=url + '/login', registerPath=url + '/signUpPage')
```

首先通过 Flask 的装饰器@app.route()将函数注册到 URI'/'下，这样，当根路径被访问时，会调用下面的函数。函数调用 Flask 的 render_template()方法，将网页模板渲染之后作为请求的响应返回给浏览器。

- **数据库查询代理类路由**

```
@app.route('/orderControl')
def order_box_control():
    id = int(request.values.get('student_id'))
    box_id = int(request.values.get('box_id'))
    control = int(request.values.get('control'))
    box = Boxes.query.get(box_id)
    box.isWanted = control
    if control:
        box.wantedPerson = id
    else:
        box.wantedPerson = 0
    db.session.commit()
    if control:
        return render_template('tips.html', content='操作成功! ', backUrl=url + '/rent?student_id=' + str(id))
    else:
        return render_template('tips.html', content='操作成功! ', backUrl=url + '/back?student_id=' + str(id))


@app.route('/back')
def back_tool_page():
    id = int(request.values.get('student_id'))
    user = Users.query.get(id)
    boxes = Boxes.query.filter(Boxes.usedPerson == id).all()
    boxes_od = Boxes.query.filter(Boxes.wantedPerson == id).all()
    return render_template('back.html', user=user, boxes=boxes, boxes_od=boxes_od, backQRCodePath=url + '/backQRCode',
                           cancelOrderPath=url + '/orderControl', mainUrl=url + '/main?id=' + str(id))
```

这类路由会先通过 Flask 的 request 方法，获取请求头里的参数信息，然后通过数据类（例如 Users 类）的 query()方法执行查询，通过 query.filter()方法可以实现对查询信息的筛选。通过 db.session.commit()方法提交对数据库的修改到数据库。并返回提示信息。

orderControl 路由先获取 HTTP GET 请求中的学号、目标工具号、操作方式（control 为 1 表示预约为 0 表示取消预约），然后从数据库获取此工具的所有信息，将 isWanted 字段设为 control 的值，若请求操作是预约，将学号写入 wantdePerson 字段，否则，将该字段置零，并提交数据库。再根据参数 control 的值返回借用页或归还页。

back 路由在获取请求参数后，查询 box 表所有信息，过滤出借用者和预约者为传入的学号的条目，传递给归还页面。

- **二维码生成路由**

```
@app.route('/CreateQRCode')
def code():
    id = str(request.values.get('student_id'))
    box_id = str(request.values.get('box_id'))
    box = Boxes.query.get(int(box_id))
    box.isEmpty = 1
    box.usedPerson = int(id)
    if box.isWanted and box.wantedPerson == id:
        box.isWanted = 0
        box.wantedPerson = 0
    db.session.commit()
    content = json.dumps([{'id': id, 'box_id': box_id, 'action': 'rent'}])
    img_url = 'http://qr.liantu.com/api.php?text=' + content
    return render_template("qrcode.html", imgpath=img_url, backUrl=url + '/rent?student_id=' + id)
```

二维码生成路由接收数据后，先查询数据库，将已被借用和借用者学号写入对应字段里，并判断借用者是否是预约后借用，若是有预约，那么将预约字段重置为 0，提交数据库保存修改。再将学号和柜子号打包成 json 数据包，调用了一个免费的二维码生成接口生成二维码。

- **单片机查询接口**

```
@app.route('/thisPowerbarIsNeededToUsed/<int:power_bar_id>', methods=["GET"])
def power_bar_query_proxy(power_bar_id):
    power_bar = Powerbars.query.get(power_bar_id)
    if power_bar.isUsed:
        return '1'
    else:
        return '0'
```

单片机查询接口在注册路由是使用了 Flask 提供的动态路由属性，可以将学号这一数据直接写在 URI 上。函数得到这一数据，查询这一个电源是否正在被使用，并返回 1 或 0。单片机可以在每个周期里轮询每一个接口，通过是否有跳变控制继电器来控制电源。

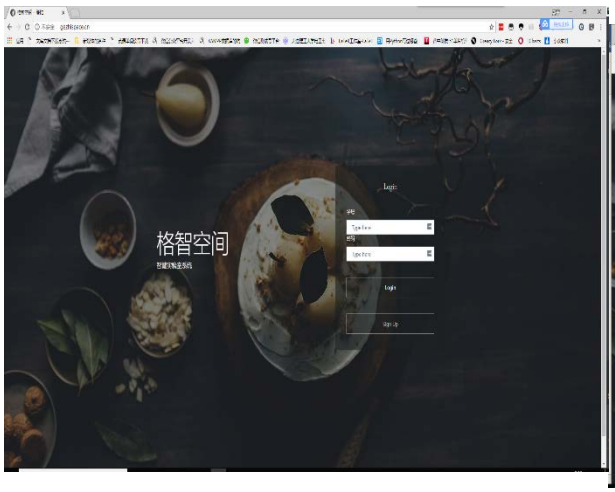
第三部分：前端页面的实现

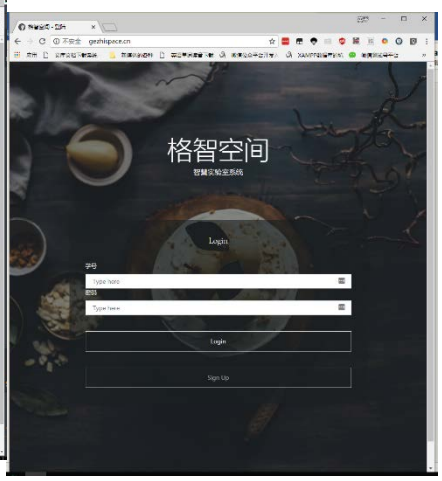
1. 前端页面功能和实现简述

本实验应用共定义了前端页面七个模板网页，其作用和所需参数如下表格：

模板名	参数	功能
back.html	user boxes boxes_od backQRCodePath cancelOrderPath mainUrl	根据用户借用和预约的工具记录生成归还工具和取消预约页面
login.html	judgeLoginPath registerPath	登陆页面
main.html	user rentPath backPath	主页面，选择功能借用或归还
qrcode.html	imgpath backUrl	显示二维码
rent.html	user boxes powerbars QRCodePath orderPath powerPath mainUrl	列出所有工具和电源信息，提供借用、预约、用电、断电功能
signup.html	judgeSignUpPath loginPath	注册页面
tips.html	content backUrl	显示提示信息

我不善于设计界面，所以前端页面使用 Bootstrap 框架的可视化编辑工具 Pingendo 做了最简单的设计，并依照 Jinja2 的语法格式对页面进行修改，完成与后端的信息交互。利用 Bootstrap 优秀的响应式开发属性，网站都采用响应式布局，可以在不同尺寸的终端上，呈现不同的界面。登陆页的不同呈现效果如图：



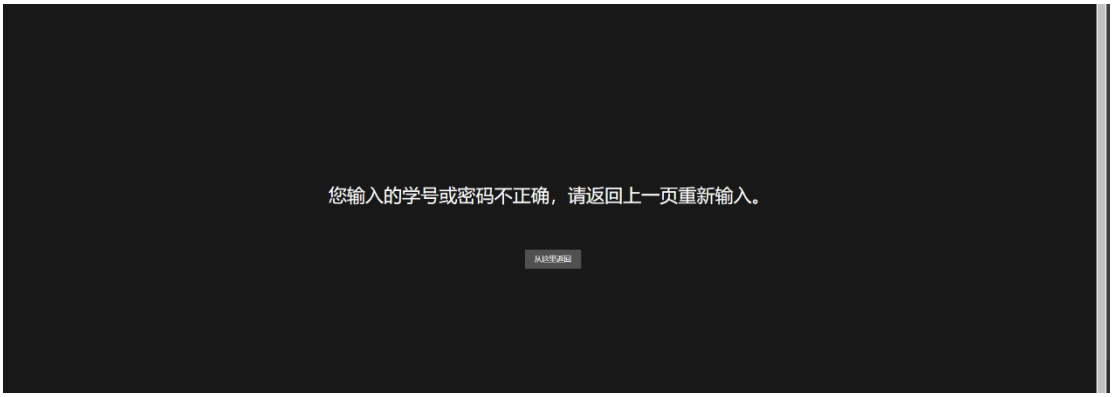


2. 代表性前端代码实现

- **普通 HTML**

```
<div class="py-3 cover align-items-center d-flex photo-overlay" id="venue">
    <div class="container">
        <div class="row">
            <div class="col-md-12 text-dark">
                <h1 class="mb-4">{{ content }}</h1>
            </div>
        </div>
        <div class="row">
            <br>
            <br>
        </div>
        <div class="row">
            <div class="col-md-12">
                <a href="{{ backUrl }}" class="btn navbar-btn ml-2 btn-secondary">从这里返回</a>
            </div>
        </div>
    </div>
</div>
```

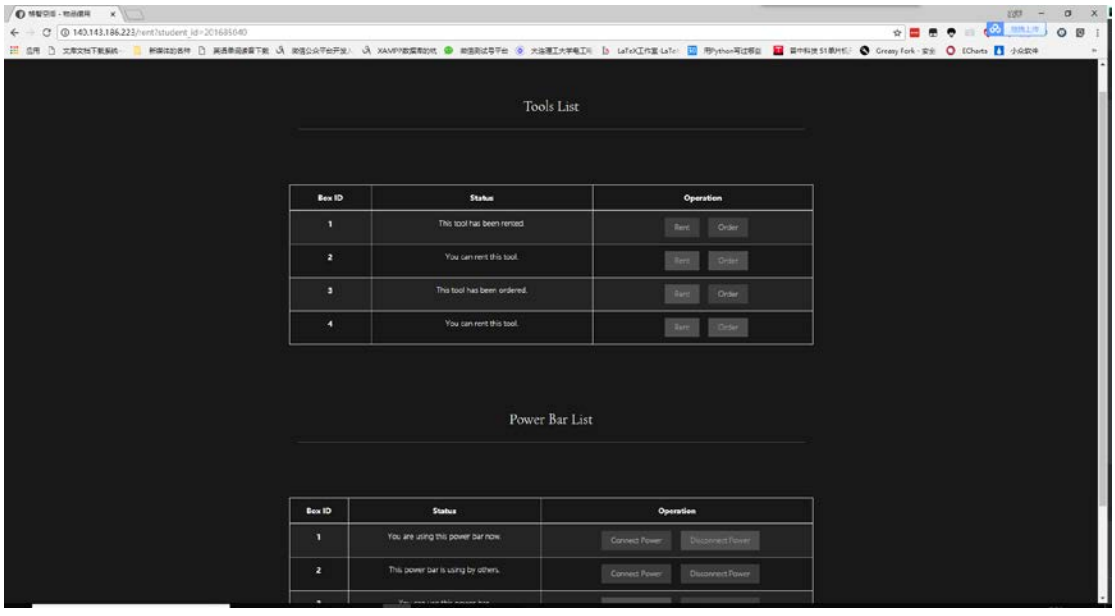
图为 tips.html 的一段代码。HTML 的主题使用前端框架 Bootstrap 响应式构建，Bootstrap 所提供的 container 类、col-md-12 等类可以实现基于屏幕大小按比例显示内容，方面交互。{{var name}}是 Jinja2 传递参数的格式，后端通过 render_template()的方法传递来的参数会被渲染到指定的位置。使用模板渲染引擎可以实现一个模板页面在不同场景下的复用。tips.html 接受两个参数，一个是文字提示内容，一个是返回按钮返回的路径。这个页面被我通过不同的参数传递，充当了所有的文字提示页面。例如：下图为密码错误提示页。



- **使用 Jinja2 特性的列表**

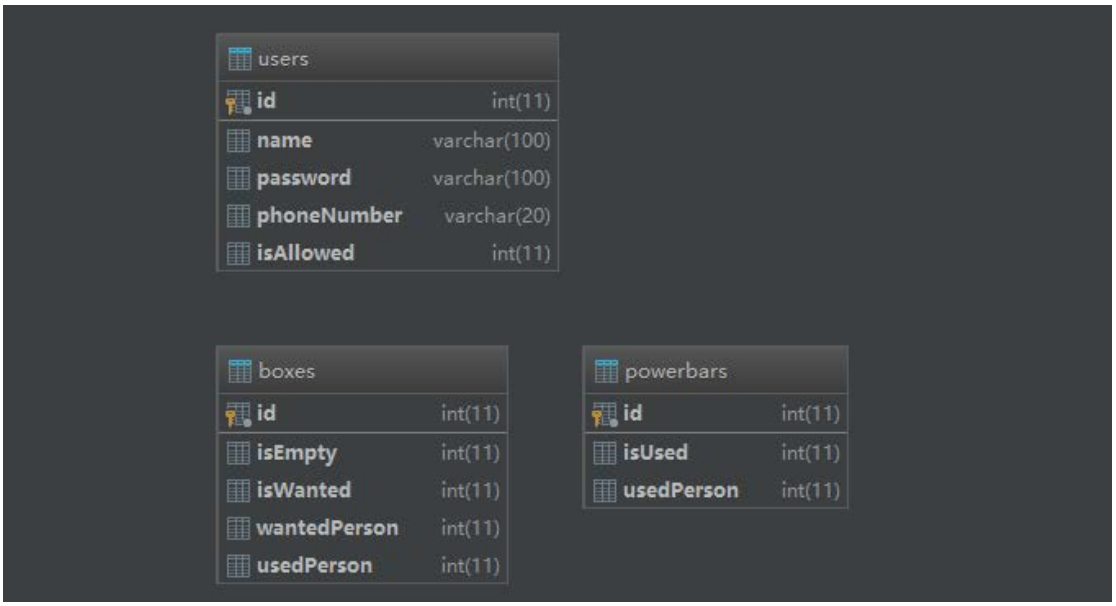
```
<table class="table table-hover table-striped table-bordered">
    <thead class="thead-inverse">
    <tr>
        <th scope="col">Box ID</th>
        <th scope="col">Status</th>
        <th scope="col">Operation</th>
    </tr>
    </thead>
    {% if powerbars %}
        <tbody class="text-center">
        {% for powerbar in powerbars %}
            <tr>
                <th scope="row">{{ powerbar.id }}</th>
                {% if powerbar.isUsed and powerbar.usedPerson != user.id %}
                    <td>This power bar is using by others.</td>
                {% elif powerbar.isUsed and powerbar.usedPerson == user.id %}
                    <td>You are using this power bar now.</td>
                {% else %}
                    <td>You can use this power bar.</td>
                {% endif %}
                <td>
                    {% if powerbar.isUsed %}
                        <button class="btn navbar-btn ml-2 btn-secondary disabled">Connect Power
                        </button>
                    {% else %}
                        <button class="btn navbar-btn ml-2 btn-secondary"><a
                                href="{{ "%s?student_id=%s&power_bar_id=%s&connect=1"|format(powerPath, user.id, powerbar.id) }}">Connect
                            Power</a></button>
                    {% endif %}
                    {% if powerbar.isUsed and powerbar.usedPerson == user.id %}
                        <button class="btn navbar-btn ml-2 btn-secondary"><a
                                href="{{ "%s?student_id=%s&power_bar_id=%s&connect=0"|format(powerPath, user.id, powerbar.id) }}">Disconnect
                            Power</a></button>
                    {% else %}
                        <button class="btn navbar-btn ml-2 btn-secondary disabled">Disconnect Power
                        </button>
                    {% endif %}
                </td>
            </tr>
        {% endfor %}
        </tbody>
    {% endif %}
</table>
```

Jinja2 可以使用形如{% code block %}的格式运行类似 Python 的代码。这段代码出自 rent.html 其作用是根据传来的参数，生成一个表格。这里面利用 for 循环遍历列表，通过 if 判断应该显示的结果。这段程序里，通过 for 循环逐行生成列表，列表每一行有三列，第一列是序号直接通过双括号的语法显示。第二列通过判断给出“该电源正在被别人使用。”、“该电源正在被你使用”、“该电源可以被使用”三种提示。第三列通过判断属性来给 button 按钮的 class 属性添加“disabled”的禁用字段。可以实现当电源不能被使用时，connect 按钮不能被带点击这样的交互。其运行效果如下图:



第四部分：数据库设计与实现

1. 数据库的图建模



2. 基于 ORM 的数据库模型的实现

```
class Powerbars(db.Model):
    __tablename__ = 'powerbars'
    id = db.Column(db.Integer, primary_key=True)
    isUsed = db.Column(db.Integer, unique=False)
    usedPerson = db.Column(db.Integer, unique=False)

    def __init__(self, id):
        self.id = id
        self.isUsed = 0
        self.usedPerson = 0

    def __repr__(self):
        return '<PowerBar %r>' % self.id
```

数据库类的实现是通过 Flask 的 ORM 拓展 Flask-SQLAlchemy 实现。数据类继承自 SQLAlchemy.db.Model 类，第二行通过 __tablename__指定表名，后面的三行定义了数据库的字段。__init__()函数定义了数据类的创建方法，只需一个 id 来初始化一个类，isUsed 字段和 usedPerson 字段默认为 0。

DBModel.py 这一模块同时还可以实现在本地端对数据库进行 CREATE 和 DROP 操作，这是 bd 类所提供的方法。在 Python 交互式命令行里执行 from DBModel import db 之后，可以通过 db 类的 create_all()方法完成数据库的建表操作。

第五部分：服务器的配置与部署

1. NginX 配置和服务的启动

NginX 是个高性能的 Web 和反向代理服务器，使用它可以实现对 HTTP 的 80 端口的监听。其配置如下：

```
ubuntu@VM-0-196-ubuntu:/etc/nginx/conf.d$ cat shengya_nginx.conf
server {
    listen 80;
    server_name 140.143.186.223;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

}
```

在工作目录下使用指令 gunicorn -w 4 –b 127.0.0.1：8080 GeZhiSpace:app 可以通过中间件 gunicorn 沟通服务器 NginX 和 Flask 应用。

2. 网站部署和域名解析

网站文件从本地到服务器的同步使用 git 完成，在本地完成功能后，上传 github 上的仓库，在服务器上 clone 后使用 gunicorn 启动。通过腾讯云提供的域名解析服务，将其解析在 http://gezhispace.cn 上。

实验总结

1. **目标完成情况**

   完成了预期的功能，我在这一实验的过程中也熟悉了 Flask 框架的基本用法，对 Python 工程开发中的包管理工具、虚拟环境工具、集成开发环境 PyCharm、版本控制工具 git 以及 Ubuntu Server 下 NginX、MySQL 等服务的配置等有了基本的了解。

   在图书馆借阅了《Python 高效开发实战 Django Tornado Flask Twisted》一书。阅读了其中部分章节，了解到了一些关于数据库原理和 TCP/IP 协议的知识。

2. **现存的问题和不足**

   一个很重要的问题是我暂时没能搞清楚如何修改配置使得 MySQL 存储中文，目前 Users 用户类中 name 字段我是使用汉语拼音存储的，理想的应该是可以直接存汉字。

   不足之处包括我对前端语言不是很了解，所以界面也很简单，js 的属性也没有好好利用，现在的网页只是把后台传来的数据列出来，并没有很好的交互。在读图书馆的书时，发现 Flask 还有很多好用的拓展，譬如用来实现登陆的 Flask-Admin，用来生成表单的 Flask-WTF 等等。时间仓促，不能够把这些拓展发挥用处，所以后台的实现并不很有 Flask 的风格。包括 Flask 本身的上下文等功能我也没有利用起来，并没有很好的发挥 Flask 的作用。这些功能的使用我将慢慢学习并应用于这个 Web 系统平台的实践上。