数据库实验报告

- 姓名：顾哲远
- 学号：162020214
- 班级：1620202
- 完成时间：2023.6.4
- 实验内容：使用前后端技术结合数据库实现一个抽卡模拟器
- 指导老师：周良老师

实验环境

我采用的是前后端实现的UI界面，前端是手写html，css，JavaScript；后端采用python语言和flask库，进行路由设置和前后端数据交互；数据库软件采用的是MySQL，但是我基本没有使用其图形化界面workbench，主要使用了python的flask_sqlalchemy库实现后端和数据库的交互，并且使用了cmd来手动添加记录，用于测试。

前端实现和部署

前端全部是手写html页面、css和js，前端的具体部署是通过flask实现的，这一点和传统的前后端分离方式不同（毕竟就我一个人做，前后端直接合体得了）。

后端实现和部署

后端使用的语言是python，使用了flask进行数据库连接、数据库操作以及和前端的数据交互。

后端所使用的头文件如下：

```
from flask import Flask, g, session
from flask import render_template
from flask_sqlalchemy import SQLAlchemy
from flask import request
from sqlalchemy import text
import random
```

random模块用于随机数生成，和前后端技术以及数据库没有直接联系。

相关软件

数据库——MySQL MySQL

后端+前端——PyCharm Download PyCharm: Python IDE for Professional Developers by JetBrains

python环境管理——Anaconda Free Download | Anaconda

页面显示和功能测试——Edge 了解 Microsoft Edge

项目文件组织

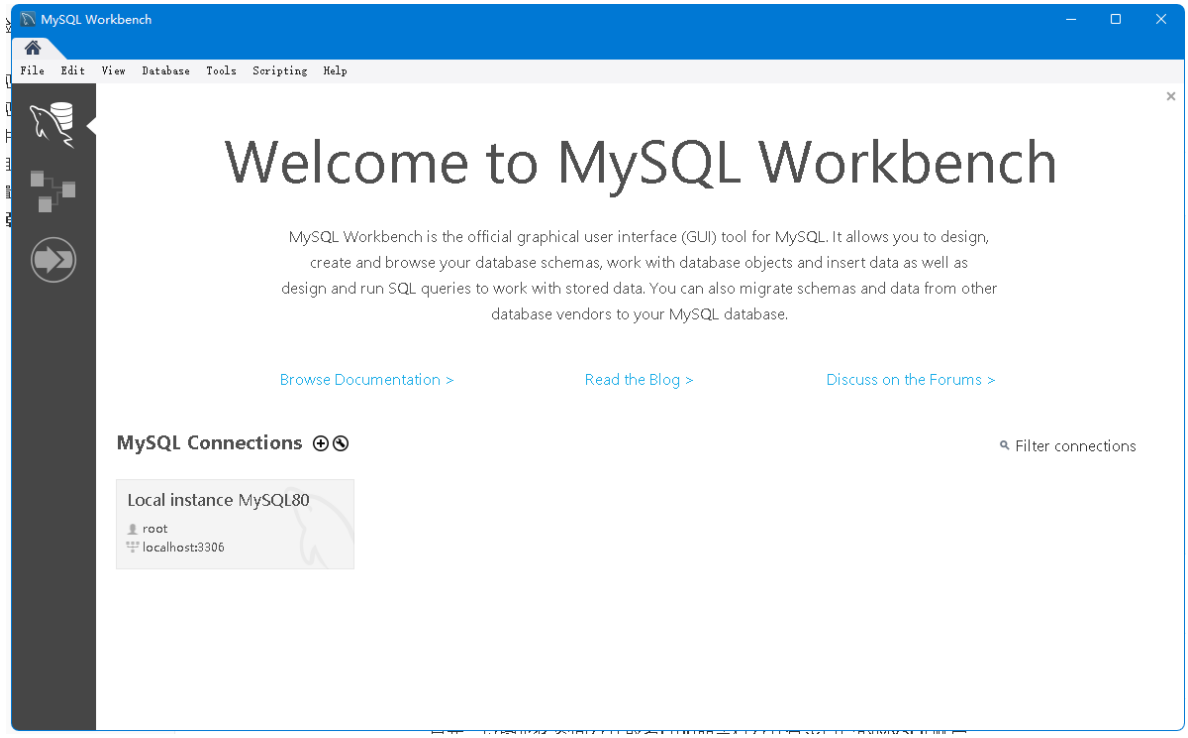
整个项目的文件组织架构如下：

```
flask # 项目根目录
-static # 用于存储css、js以及图片资源
--css # 用于存储各种css样式文件的目录
--Javascript # 用于存储js文件的目录
--others # 用于存储图片资源的目录
-templates # 用于存储html文件的目录
--gacha.html # 抽卡目录
--helpless1.html # 抽卡结果分析目录
--helpless2.html # 显示此报告的目录
--index.html # 登录界面
--my_gacha_result.html # 查看抽卡结果的目录
-app.config # 用于对整个项目进行配置，是flask生成的config，从结果上看没有用到这个文件
-test.py # 所有后端逻辑都在这个文件里面，也是整个flask项目的起点
```

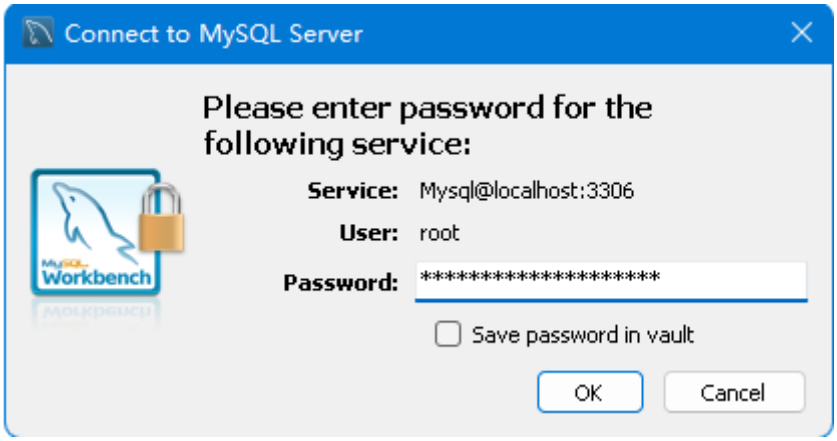
数据库配置

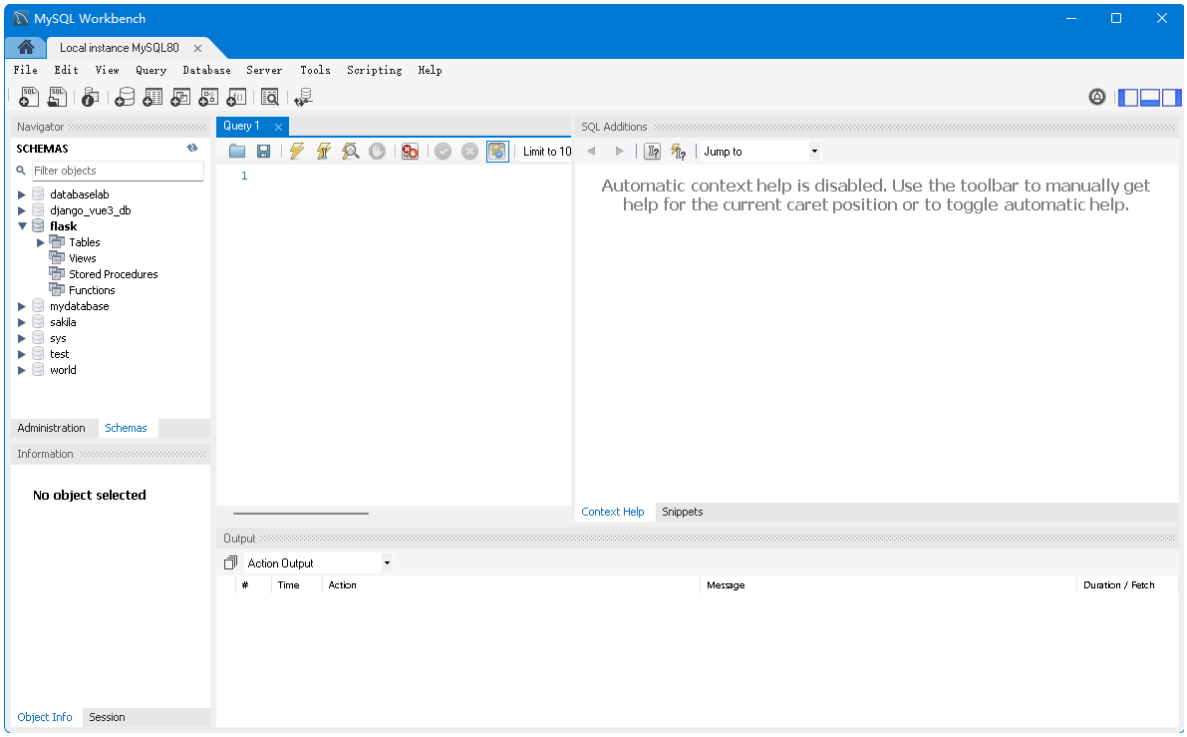
创建数据库和相关表格

首先，以图形化界面方式或者cmd命令行方式登录自己的MySQL账户：
图形化方式：直接单击打开workbench软件即可，一般而言直接下载mysql会自动安装一个，点开以后出现如下界面



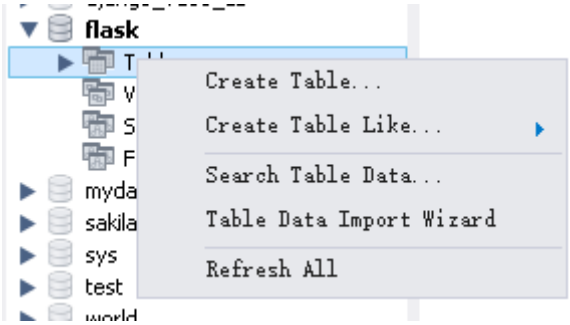
这个时候单击MySQL Connections里面的这个Local instance MySQL80，输入自己设定的密码，然后登录





登录以后，在左侧的SCHEMAS栏新建一个本次项目的数据库，我新建的叫做flask

可以右键数据库来创建新表



当然，也可以用命令行方式创建，个人觉得这种方式更加直接且方便，但是前提是已经将mysql.exe所在的路径加入了系统环境变量PATH中（系统高级设置->环境变量->PATH->添加），这样就可以直接打开cmd，

输入mysql -u root -p，然后输入密码即可登录

```
C:\Users\86180>mysql -u root -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 49
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

这个时候可以create database，也可以use我们之前的flask数据库，我这里直接use

```
mysql> use flask
Database changed
mysql>
```

然后用指令创建表格即可。

创建user表格

user表格总共有三列，第一列是user_id，也是主键，每次insert的时候都会自动递增，为int类型；

第二列是user_name，也是用户自己注册时使用的用户名，用户名可以重复，为字符类型；

第三列是user_pass，也就是用户的密码，为字符类型；

创建表格的代码如下：

```
CREATE TABLE user (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  user_name VARCHAR(255) NOT NULL,
  user_pass VARCHAR(255) NOT NULL
) AUTO_INCREMENT = 1;
```

名称	类型	约束	描述
user_id	INT	primary key	主键，用于识别不同用户的id
user_name	VARCHAR(255)	none	用户名，很自由的一个字符串
user_pass	VARCHAR(255)	none	用户密码，很自由的一个字符串

创建gacha_list表格

```
CREATE TABLE gacha_list (
    card_id INT PRIMARY KEY,
    card_name VARCHAR(255),
    card_status INT,
    card_describe VARCHAR(255)
);
```

名称	类型	约束	描述
card_id	INT	primary key	唯一标识，卡片id
card_name	VARCHAR(255)	none	卡名，很自由的一个字符串
card_status	INT	none	卡片稀有度，一个int
card_describe	VARCHAR(255)	none	卡片描述，字符串

创建gacha_history表格

```
-- 创建 gacha_history 表格
CREATE TABLE gacha_history (
  user_id INT,
  card_id INT,
  obtain_time TIMESTAMP,

  -- 添加外键约束
  CONSTRAINT fk_user_id
    FOREIGN KEY (user_id)
```

```
      REFERENCES user (user_id),
    CONSTRAINT fk_card_id
      FOREIGN KEY (card_id)
      REFERENCES gacha_list (card_id)
);
```

名称	类型	约束	描述
user_id	INT	外键约束	用户唯一id
card_id	INT	外键约束	卡片唯一id
obtain_time	TIMESTAMP	none	获取时间（添加记录的时间）

添加测试数据

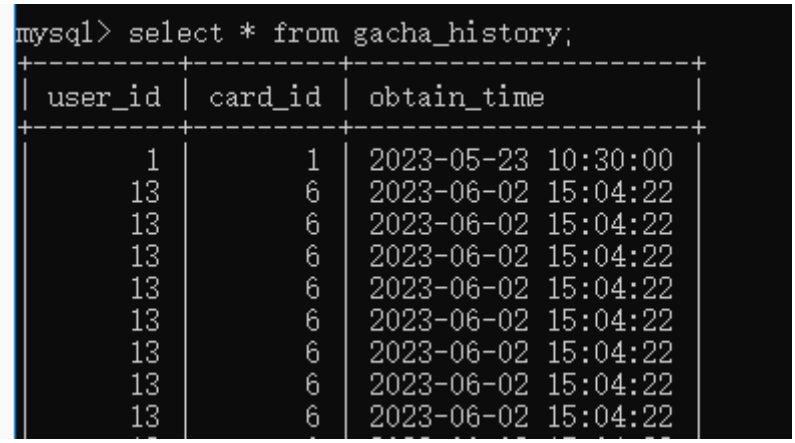
使用INSERT语句在几个数据库里添加一些数据，用于测试：

```
mysql> select * from user;
+---------+-------------+-------------+
| user_id | user_name   | user_pass   |
+---------+-------------+-------------+
|       1 | test        | test        |
|       2 | hanghang    | 214         |
|       3 | ct1         | ct1         |
|       4 | cgr         | cgr         |
|       9 | 111         | 111         |
|      10 | 1           | 1           |
|      11 | 44          | 44          |
|      12 | 12312312    | asdasgasdff |
|      13 | 666         | 666         |
|      14 | test244234  | 123122      |
|      15 | test4       | 64          |
|      16 | te          | test        |
|      17 | test111     | test        |
|      18 | test111111  | test        |
|      19 | chenwenjie  | chenwenjie  |
|      20 | hanghang214 | aaaaaa      |
+---------+-------------+-------------+
```

```
mysql> select * from gacha_list;
+---------+-------------------+-------------+------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------
--------------------------------+
| card_id | card_name         | card_status | card_describe

                                |
+---------+-------------------+-------------+------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------
--------------------------------+
|       1 | 果心居士          |           5 | 传说中的幻术（外术）使。妖术师。

                                |
|       2 | 瑟坦特            |           5 | 凯尔特神话、阿尔斯特神话中的大英雄
库·丘林--其修行时代的姿态。
肉体与精神都比以Lancer或Caster职阶现界的他更为年轻且稚嫩。然而同时，也蕴藏着作为一位为了成为什么样的
少年所具有的无限的可能性。                    |
|       3 | 大黑天            |           4 | 在日本，是作为七福神之一被大家所熟悉的福神。
但是，此次作为从者现界的，并不是那位神本身，而是神差遣的老鼠们
因此，灵格比原来的形象低很多，能做的事也很有限。
|       4 | 清少纳言          |           4 | 夏天是夜里最好。有月亮的时候，不必说了--但是，
夏天还有很多很多好玩的事情！
```



```
mysql> select * from gacha_history;
+---------+---------+---------------------+
| user_id | card_id | obtain_time         |
+---------+---------+---------------------+
|       1 |       1 | 2023-05-23 10:30:00 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
|      13 |       6 | 2023-06-02 15:04:22 |
```

前后端代码

各页面html和后端对应

index.html

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ε٩( ᵒ' 'ᵒ)ง3</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css')
}}">
    <link rel="stylesheet" href="//unpkg.com/element-plus/dist/index.css" />
    <link rel="stylesheet" href="../static/css/char-style.css">
    <link rel="stylesheet" href="../static/css/style.css">
    <!-- Import Vue 3 -->
    <script src="//unpkg.com/vue@3"></script>
    <!-- Import component library -->
    <script src="//unpkg.com/element-plus"></script>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css')
}}">
    <style>
        body {
            animation: fadeInAnimation ease 3s;
            animation-iteration-count: 1;
            animation-fill-mode: forwards;
        }
        @keyframes fadeInAnimation {
            0% {
                opacity: 0;
            }
            100% {
                opacity: 1;
            }
        }
        h2 {
            color: rgba(255,255,255,0.9);
            margin-bottom: 20px;
        }
```

```
    </style>
</head>
<body>
<div class="box">
    <form method="POST" action="/">
        <h2>FGO抽卡模拟器</h2>
        <div class="input-box">
                <label for="username">用户名</label>
                <input type="text" id="username" name="username" required>
        </div>
        <div class="input-box">
                <label for="password">密码</label>
                <input type="password" id="password" name="password" required>
        </div>
        <div class="btn-box">
            <a href="/gacha">游客登录</a>
            <div>
                <button type="submit">提交</button>
                <button type="reset">重置</button>
            </div>
            {% if data == "登录成功" %}
    <script>
        alert("登录成功！")
        window.location.href="./gacha";
    </script>
    {% endif %}
    {% if data == "新用户注册成功" %}
    <script>
        alert("新用户注册成功")
        window.location.href="./gacha";
    </script>
    {% endif %}
        {% if data == "登录失败，请检查密码重新输入" %}
    <script>
        alert("登录失败，请检查密码重新输入")
    </script>
    {% endif %}
        </div>
    </form>
    </div>
<script type="text/javascript"></script>

</body>
</html>
```

前端有一个表单提交的方法，对于这个路由，后端的方法如下：

```
@app.route('/', methods=['GET', 'POST'])
def index():
   if request.method == 'GET':
      return render_template("index.html",data="default")
   else:
      login_result = 0 # 0->没有用户，进行注册，1->有用户，登录成功，2->有用户，登录失败
      login_result = login_request()
      if login_result == 0:
         msg = "新用户注册成功"
      elif login_result == 1:
```

```
            msg = "登录成功"
        else:
            msg = "登录失败，请检查密码重新输入"
        return render_template("index.html",data=msg)

def login_request():
    # 接收post请求的form表单参数
    username = request.form.get('username')
    userpass = request.form.get('password')
    print(str(username))
    print(str(userpass))
    with app.app_context():
        with db.engine.connect() as conn:
            stmt = text("SELECT IF(COUNT(*) > 0, 1, 0) AS result FROM user WHERE
user_name = :username")
            rs = conn.execute(stmt, {"username": username})
            whether_registed = rs.fetchone()[0]
            conn.commit()
            conn.closed
    if whether_registed > 0: # 这是用户已经注册过的情况，数据库内存储了用户的信息（主要是
用户名，用户注册也就看这个，然后用户id好像没啥用了）接下来一步应该是检验密码
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("SELECT COUNT(*) AS count FROM user WHERE user_name =
:username AND user_pass = :userpass;")
                rs = conn.execute(stmt, {"username": username,"userpass":userpass})
                pass_correct = rs.fetchone()[0]
                if pass_correct > 0: # 用户登录成功
                    login_result = 1
                    session['username'] = username
                    session['userpass'] = userpass
                    print("登录成功")
                else:
                    login_result = 2
                    print("输入密码错误，登录失败")
                conn.commit()
                conn.closed
    else: # 用户没有注册过的情况
        print("没注册过，帮他注册一个")
        login_result = 0
        print("账户名："+username+" "+"密码："+userpass)
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("INSERT INTO user (user_name, user_pass) VALUES
(:username, :userpass);")
                rs = conn.execute(stmt, {"username": username,"userpass":userpass})
                conn.commit()
                conn.closed
        session['username'] = username
        session['userpass'] = userpass
    return login_result
```

后端会返回页面以及一个msg，前端根据msg的信息来判断下一步的行为。

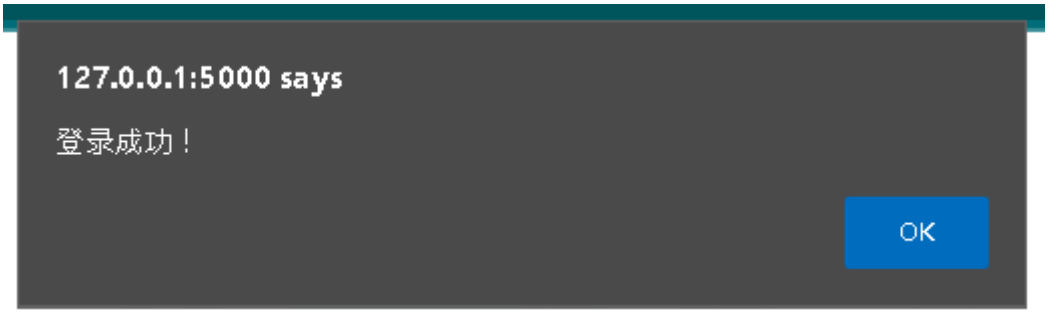
如果没有这个用户存在，就注册一个然后让其登录；如果有的话，就判断密码正确与否，决定是否放行。

session是用于存储用户名和密码的重要全局变量（生存期是整个会话）

这个页面的实际效果如下：

为了测试方便，提供了一个"游客登录"通道，可以直接进入下一个页面。

登录成功以后会弹窗，然后前往下一个页面（这个写在前端js里）



gacha.html

这就是登录后进入的第一个页面，可操作的地方有：底部的"抽个十连"按钮，以及中间图像展示框的左右切换按钮，此外，右下角的图标小人可以拖拽，其设置了回弹的阻尼系数等物理属性；左上方是切换页面的导航栏。

点击底部按钮，可以显示当前这次十连抽到了哪些：

gacha.html的代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ʕ,,,˙ꈊ˙ʔ</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
integrity="sha384-
xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N"
crossorigin="anonymous">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/gacha.css')
}}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css')
}}">
    <style>
        body {
            animation: fadeInAnimation ease 3s;
            animation-iteration-count: 1;
            animation-fill-mode: forwards;
        }
        @keyframes fadeInAnimation {
            0% {
                opacity: 0;
            }
            100% {
                opacity: 1;
            }
        }
        html .sakana-box{
        position: fixed;
        right: 0;
        bottom: 0;
        transform-origin: 100% 100%; /* 从右下开始变换 */
        }
    </style>
</head>
```

```
<body>
<div>
    <header class=sticky-header>
      <nav>
        <ul>
          <li><a href={{ url_for('index') }}>回到首页</a></li>
          <li><a href={{ url_for('gacha') }}>开始抽卡</a></li>
          <li><a href={{ url_for('my_gacha_result') }}>抽卡记录</a></li>
          <li><a href={{ url_for('helpless1') }}>抽卡结果分析</a></li>
          <li><a href={{ url_for('helpless2') }}>实验报告</a></li>
          <div class="slider"></div>
        </ul>
      </nav>
    </header>
</div>
<div><br><br></div>
<div class="container">
<h2 style="color: lightgray;">你说的对，但是</h2>
<p class="h4" style="color: lightgray;">《妖精圆桌领域》是由奈须蘑菇自主研发的一款全新开放世界冒险游戏。
    游戏发生在一个被称作「妖精国不列颠」的幻想世界，在这里被女王选中的人将被授予「妖精骑士着名」，引导妖精之力。
    你将扮演一位名为「摄像头」的神秘角色，在自由的旅行中邂逅性格各异、能力独特的芬恩、狮子头和C狗，和它们一起击败强敌，
    找回狮子夫人的同时，逐步发掘「蘑菇原案」的真相。</p>

<div id="carouselExampleCaptions" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carouselExampleCaptions" data-slide-to="0" class="active">
</li>
    <li data-target="#carouselExampleCaptions" data-slide-to="1"></li>
    <li data-target="#carouselExampleCaptions" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="../static/others/show_pic1.png" class="d-block w-100" alt="0">
      <div class="carousel-caption d-none d-md-block">
      </div>
    </div>
    <div class="carousel-item">
      <img src="../static/others/show_pic2.png" class="d-block w-100" alt="1">
      <div class="carousel-caption d-none d-md-block">
      </div>
    </div>
    <div class="carousel-item">
      <img src="../static/others/show_pic3.png" class="d-block w-100" alt="2">
      <div class="carousel-caption d-none d-md-block">
      </div>
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-
target="#carouselExampleCaptions" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-
target="#carouselExampleCaptions" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
```

```
    <span class="sr-only">Next</span>
  </button>
</div>

<blockquote class="blockquote text-center">
<p class="mb-1" style="color: white">Fate Grand Order GachaSimulator 2023</p>
<footer class="blockquote-footer" style="color: white">NUAA 数据库实验@<cite
title="https://github.com/hanghang214">hanghang214</cite></footer>
</blockquote>
</div>
<div class="container">
    <div class="item"><span id="data-placeholder"></span></div>
</div>
<div class="for_button">
  <button class="custom-btn btn-6" onclick="sendRequest()"><span>抽个十连</span>
</button>
</div>
<div class="sakana-box"></div>

<script src="https://cdn.jsdelivr.net/npm/sakana"></script>
<script>
Sakana.init({
  el:         '.sakana-box',     // 启动元素 node 或 选择器
  scale:      .5,                // 缩放倍数
  canSwitchCharacter: true,      // 允许换角色
});
</script>
{% if data == "test" %}
    <script>
        alert("test! ")
    </script>
{% endif %}
<script>
function sendRequest() {
  fetch('/gacha', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ data: '十连抽' }) // 向后台发送一个字符串类型的数据
  })
  .then(response => {
    // 处理响应结果
    if (response.ok) {
      // 请求成功
      return response.text(); // 将响应的数据作为字符串处理
    } else {
      // 请求失败
      throw new Error('Request failed');
    }
  })
  .then(data => {
    // 处理返回的字符串数据
    console.log(data);
    // 在这里可以根据返回的数据更新页面或执行其他操作
      document.getElementById('data-placeholder').innerText = data; // 更新页面上
的数据
      if(data==="test")
```

```
        {
            window.alert("后端数据传输成功");
        }
    })
    .catch(error => {
      // 处理错误
      console.error(error);
    });
}
</script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.min.js"
integrity="sha384-
7ymO4nGrkm372HoSbq1OY2DP4pEZnMiA+EOF3zPr+JQQtQ82gQ1HPY3QIVtztVua"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.min.js">
</script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js
"></script>
</body>
</html>
```

这个页面有一个向后端发送json包的行为，传输的是字符串"十连抽"，后端收到json后会进行十连抽，然后以一个长长的string的形式，把十连抽的结果返回给前端，显示在data-placeholder上。

后端处理的代码如下：

```
@app.route('/gacha', methods=['GET', 'POST'])
def gacha():
    if request.method == 'GET':
        print("get gacha")
        return render_template("gacha.html")
    else:
        data = request.get_json()  # 获取前端发送的JSON数据
        received_string = data['data']  # 获取字符串数据
        return_string = ""
        username = session.get('username')
        print("后端gacha收到：" + received_string)
        print("username:"+session.get('username'))
        # 这里根据当前用户的username，去用户表查询其user_id，然后再去卡池执行抽卡逻辑，拼接
获得新的记录，最后插入gacha_result中
        # 同时，这里需要把插入的card_name全部记录，以便于输出和显示
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("SELECT user_id AS result FROM user WHERE user_name =
:username")
                rs = conn.execute(stmt, {"username": username})
                userid= rs.fetchone()[0] # 这里获取到了当前用户的id
                for i in range(0,10):
                    gacha_result_once = generate_number()
                    if gacha_result_once == "3":
                        stmt = text("SELECT * FROM gacha_list WHERE card_status = 3
ORDER BY RAND() LIMIT 1;")
                        rs = conn.execute(stmt, {"username": username})
                        row = rs.fetchone()
```

```
                    cardid = row[0]  # 获取第一列 card_id
                    cardname = row[1]  # 获取第二列 card_name
                    print(cardid)
                    print(cardname)
                    return_string = return_string + cardname + " "
                elif gacha_result_once == "4":
                    stmt = text("SELECT * FROM gacha_list WHERE card_status = 4
ORDER BY RAND() LIMIT 1;")
                    rs = conn.execute(stmt, {"username": username})
                    row = rs.fetchone()
                    cardid = row[0]  # 获取第一列 card_id
                    cardname = row[1]  # 获取第二列 card_name
                    print(cardid)
                    print(cardname)
                    return_string = return_string + cardname + " "
                elif gacha_result_once == "5":
                    stmt = text("SELECT * FROM gacha_list WHERE card_status = 5
ORDER BY RAND() LIMIT 1;")
                    rs = conn.execute(stmt, {"username": username})
                    row = rs.fetchone()
                    cardid = row[0]  # 获取第一列 card_id
                    cardname = row[1]  # 获取第二列 card_name
                    print(cardid)
                    print(cardname)
                    return_string = return_string + cardname + " "
                stmt = text("INSERT INTO gacha_history
VALUES(:userid,:cardid,CURRENT_TIMESTAMP)")
                rs = conn.execute(stmt, {"userid": userid, "cardid": cardid})
                print("抽到了："+str(cardid))
            conn.commit()
            conn.closed
            print("userid："+str(userid))
        return session.get('username')+"上次十连的结果："+return_string
```

my_gacha_result.html

这个页面展示当前用户的抽卡记录，会**根据稀有度排序**，表格可以滑动展示

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/my_gacha_result.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css')
}}">
    <title>抽卡记录</title>
    <style>
        body {
            animation: fadeInAnimation ease 3s;
            animation-iteration-count: 1;
            animation-fill-mode: forwards;
        }
        @keyframes fadeInAnimation {
            0% {
                opacity: 0;
            }
            100% {
                opacity: 1;
            }
        }
        html .sakana-box{
        position: fixed;
        right: 0;
        bottom: 0;
        transform-origin: 100% 100%; /* 从右下开始变换 */
        }
    </style>
</head>
<body>
<div>
<header class=sticky-header>
      <nav>
        <ul>
          <li><a href={{ url_for('index') }}>回到首页</a></li>
          <li><a href={{ url_for('gacha') }}>开始抽卡</a></li>
          <li><a href={{ url_for('my_gacha_result') }}>抽卡记录</a></li>
          <li><a href={{ url_for('helpless1') }}>抽卡结果分析</a></li>
          <li><a href={{ url_for('helpless2') }}>实验报告</a></li>
          <div class="slider"></div>
        </ul>
      </nav>
    </header>
</div>
<div><br><br></div>
<div class="container">
    <div class="ytable">
    <table class="table table-hover center-table">
    <thead>
    <tr>

        {% for i in labels %}
            <td>{{ i }}</td>
        {% endfor %}

    </tr>
```

```
    </thead>
    <tbody>

    {% for i in content %}
        <tr>
            {% for j in i %}
                <td>{{ j }}</td>
            {% endfor %}
        </tr>
    {% endfor %}
    </tbody>
    </table>
    </div>
</div>
<div class="sakana-box"></div>

<script src="https://cdn.jsdelivr.net/npm/sakana"></script>
<script>
Sakana.init({
  el:         '.sakana-box',      // 启动元素 node 或 选择器
  scale:      .5,                 // 缩放倍数
  canSwitchCharacter: true,       // 允许换角色
});
</script>
</body>
</html>
```

后端把整个表头列表和表格内容列表传给前端，前端通过**JavaScript动态生成表格**进行展示。

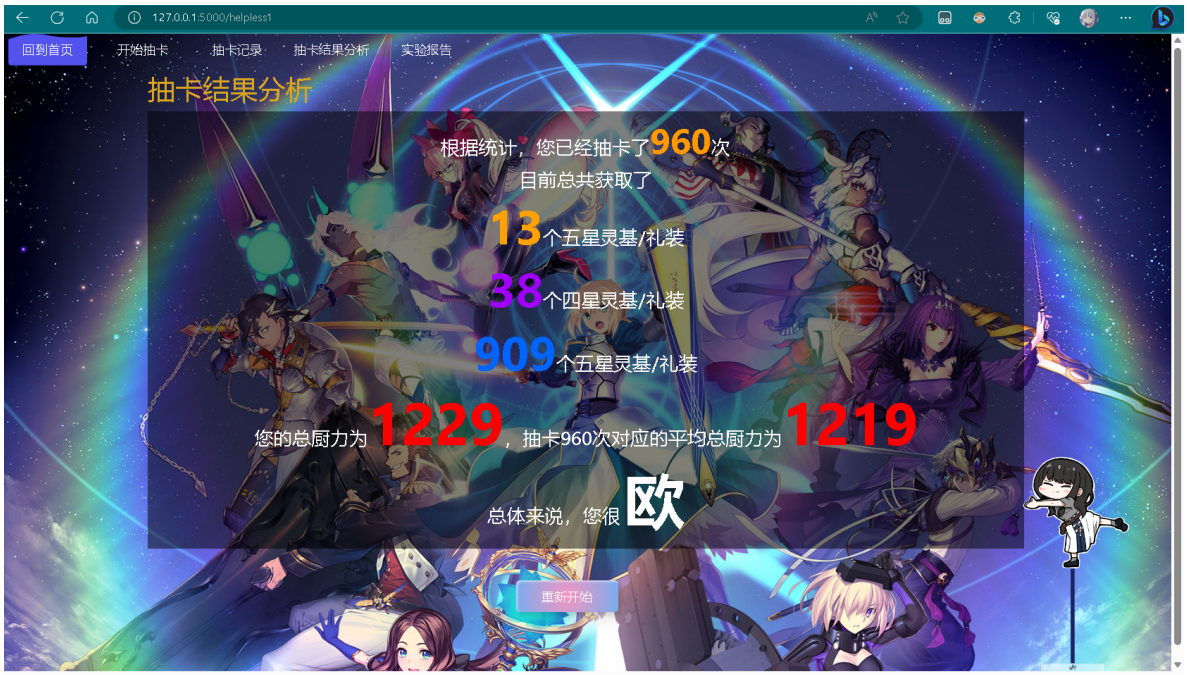
后端代码：

```
@app.route('/my_gacha_result', methods=['GET', 'POST'])
def my_gacha_result():
   # 这里首先把gacha_history里面所有属于userid的记录获取出来，然后用表格展示（数据全部传到前端）
   # 但是在这之前需要先把userid设置为session的全局变量()难蚌，感觉可以在前面登录和注册的功能那里加一个获取id的模块
   username = session.get('username')
   with app.app_context():
      with db.engine.connect() as conn:
         stmt = text("SELECT user_id FROM user WHERE user_name = :username;")
         rs = conn.execute(stmt, {"username": username})
         userid = rs.fetchone()[0]
         session['userid'] = userid # 此处获取了userid，去表格查询其所有的抽卡记录
         stmt = text("SELECT * FROM gacha_history WHERE user_id = :userid;")
         rs = conn.execute(stmt, {"userid": userid})
         history_list = rs.fetchall() # 此处获取一个list，其中每个元素都是一个元组，组成形式：(userid, cardid, obtain_time)，但是这个obtain_time比较特殊
         return_list = []
         # 接下来，通过对元组重新赋值，将list中每一个元组转化为(card_name, card_status, obtain_time),然后把list传给前端
         for record in history_list:
            cardid = record[1]
            obtain_time = record[2]
            stmt = text("SELECT card_name, card_status FROM gacha_list WHERE card_id =:cardid;")
            rs = conn.execute(stmt, {"cardid": cardid})
```

```
            card = rs.fetchone()
            cardname = card[0]
            cardstatus = card[1]
            return_list.append((cardname, cardstatus, obtain_time))
        stmt = text("SHOW FIELDS FROM gacha_history")
        rs = conn.execute(stmt)
        labels = rs.fetchall()
        labels = [l[0] for l in labels]
        # 这段代码获取表头，但是吧，应该可以在前端写一下，把表头单独展示，而不是在表格展示
        # 因为现在的滚动表格展示，只要一滚动，表头就看不见了，最好的方法其实是让表头固定，
然后表格内容滑动
        # 有空再做
        conn.commit()
        conn.closed
    labels[0] = "灵基/礼装名称"
    labels[1] = "稀有度"
    labels[2] = "获取时间"
    return_list.sort(key=takeSecond,reverse=True)
    return
render_template("my_gacha_result.html",labels=labels,content=return_list)
```

思路都七七八八写在代码里啦

helpless1.html



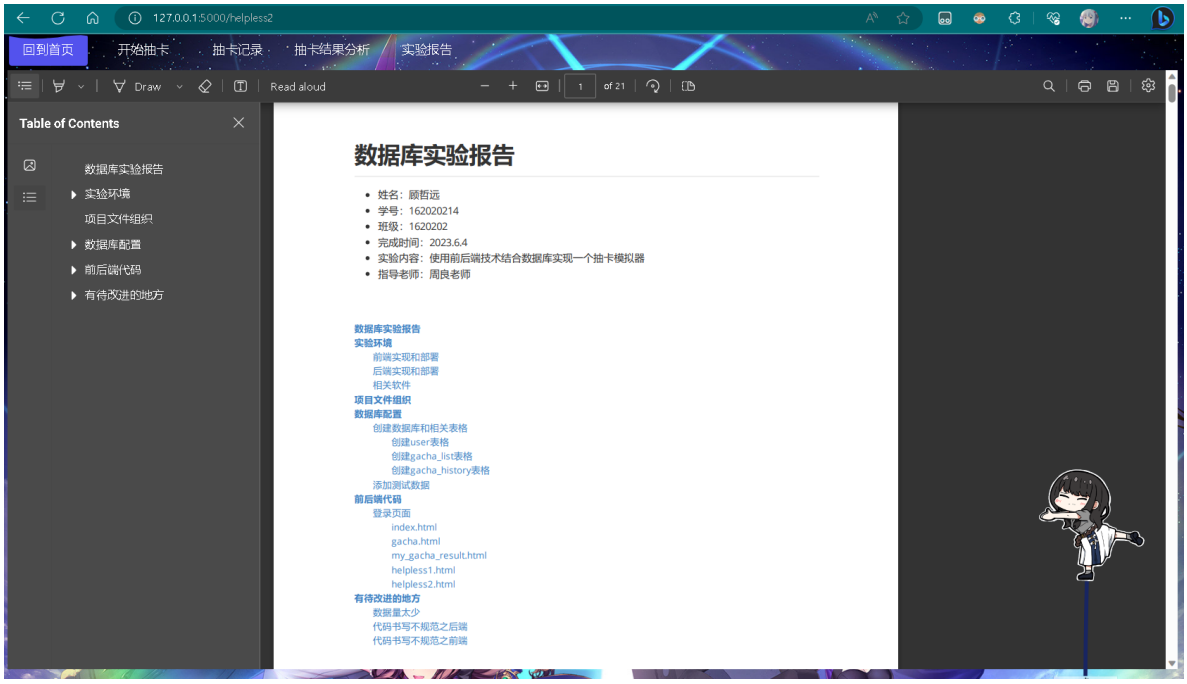
这个页面对当前用户的抽卡情况做统计和展示，提供一个按钮，可以让用户删除所有自己的抽卡记录，从头再来。

后端逻辑就是接收POST的时候就去删，接收GET就展示。

```
@app.route('/helpless1', methods=['GET', 'POST'])
def helpless1():
    if request.method == 'GET':
        gacha_times = 0
        userid = session['userid']
        user_five = 0
        user_four = 0
        user_three = 0
```

```
        user_total_score = 0
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("SELECT COUNT(*) AS count FROM gacha_history WHERE
user_id = :userid;")
                rs = conn.execute(stmt, {"userid": userid})
                gacha_times = rs.fetchone()[0]  # 获取该用户总的抽卡次数
                stmt = text("SELECT * FROM gacha_history WHERE user_id = :userid;")
                rs = conn.execute(stmt, {"userid": userid})
                user_gacha_history = rs.fetchall()  # 获取该用户所有的抽卡记录，做分析和展
示
                for item in user_gacha_history:
                    cardid = item[1]
                    stmt = text("SELECT card_status FROM gacha_list WHERE card_id
=:cardid;")
                    rs = conn.execute(stmt, {"cardid": cardid})
                    card = rs.fetchone()
                    cardstatus = card[0]
                    if cardstatus == 5:
                        user_five = user_five + 1
                    elif cardstatus == 4:
                        user_four = user_four + 1
                    elif cardstatus == 3:
                        user_three = user_three + 1
        user_total_score = user_three * 1 + user_four * 5 + user_five * 10
        user_average_score = gacha_times * 1 * 0.95 + gacha_times * 5 * 0.036 +
gacha_times * 10 * 0.014
        user_average_score = int(user_average_score)
        if user_average_score < user_total_score:
            result = "欧"
        else:
            result = "非"
        conn.commit()
        conn.closed
        return render_template("helpless1.html", result=result,
gacha_times=gacha_times, user_five=user_five,
                               user_four=user_four, user_three=user_three,
user_average_score=user_average_score,
                               user_total_score=user_total_score)
    else: # 查询数据库，把该用户的所有抽卡记录删除
        userid = session['userid']
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("DELETE FROM gacha_history WHERE user_id=:userid")
                rs = conn.execute(stmt, {"userid": userid})
                conn.commit()
                conn.closed
    return render_template("helpless1.html")
```

helpless2.html



展示报告的页面

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>更是没用</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css')
}}">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/helpless.css') }}">

    <style>
        body {
            animation: fadeInAnimation ease 3s;
            animation-iteration-count: 1;
            animation-fill-mode: forwards;
        }
        @keyframes fadeInAnimation {
            0% {
                opacity: 0;
            }
            100% {
                opacity: 1;
            }
        }
        html .sakana-box{
        position: fixed;
        right: 0;
        bottom: 0;
        transform-origin: 100% 100%; /* 从右下开始变换 */
        }
    </style>
</head>
<body>
<div>
<header class=sticky-header>
      <nav>
```

```
        <ul>
          <li><a href={{ url_for('index') }}>回到首页</a></li>
          <li><a href={{ url_for('gacha') }}>开始抽卡</a></li>
          <li><a href={{ url_for('my_gacha_result') }}>抽卡记录</a></li>
          <li><a href={{ url_for('helpless1') }}>抽卡结果分析</a></li>
          <li><a href={{ url_for('helpless2') }}>实验报告</a></li>
          <div class="slider"></div>
        </ul>
      </nav>
    </header>
</div>
<div><br><br></div>
<embed src="../static/others/report.pdf" type="application/pdf" width="100%"
height="760px" />

<div class="sakana-box"></div>

<script src="https://cdn.jsdelivr.net/npm/sakana"></script>
<script>
Sakana.init({
  el:         '.sakana-box',     // 启动元素 node 或 选择器
  scale:      .5,                // 缩放倍数
  canSwitchCharacter: true,      // 允许换角色
});
</script>
</body>
</html>
```

后端没啥好说的这里

```
@app.route('/helpless2', methods=['GET', 'POST'])
def helpless2():
    return render_template("helpless2.html")
```

CSS样式

main.css

```
.header {
  color: #ffa402;
  padding: 30px;
  font-size: 30px;
  left: 25px;
  top: 25px;
  background-position: center;
  position: absolute;
  z-index: 1000;
}
.sub-header {
  color: #000000;
  padding: 0px;
  font-size: 100px;
  background-position: center;
  position: relative;
  animation: ease-in;
  z-index: 10;
```

```
}
.cover {
  position: absolute;
  top: -20px;
  left: 50px;
  transform: scale(0.7);
  z-index: 1;
  opacity: 1.0;
}
.background {
  background-image: url('../others/705836.jpg');
  background-size: cover;  /* 调整背景图片的尺寸以覆盖整个元素 */
  background-position: center;  /* 将背景图片居中 */
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  opacity: 0.5;
  z-index: 0;
}
.login_background {
  background-image: url('../others/login_background2.png');
  background-size: cover;  /* 调整背景图片的尺寸以覆盖整个元素 */
  background-position: center;  /* 将背景图片居中 */
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  opacity: 0.5;
  z-index: 0;
}
.enter{
  color: #ffa402;
  padding: 10px;
  z-index: 100;
}
.top_bar {
    background: antiquewhite;
    position: relative;
    left: 0%;
    top: 30%;
}
.link_style {
    font-size: 30px;
    color: #ffa402;
}




ul,
li {
  margin: 0;
  padding: 0;
}
```

```css
nav {
  display: unset;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

ul {
  position: relative;
  display: flex;
}

ul li {
  /* 如果设置为inline-block，会有空隙 */
  /* https://stackoverflow.com/questions/19038799/why-is-there-an-unexplainable-gap-between-these-inline-block-div-elements */
  list-style: none;
  width: 120px;
  line-height: 40px;
  text-align: center;
}

ul li a {
  color: white;
  text-decoration: none;
}

.slider {
  width: 100px;
  height: 40px;
  background-color: #5352ed;
  border-radius: 4px;
  position: absolute;
  left: 10px;
  bottom: 0;
  z-index: -1;
  transition: all ease 0.4s;
  animation: 2s ease-in-out waves infinite;
}

li:nth-child(1):hover ~ .slider {
  left: 10px;
}

li:nth-child(2):hover ~ .slider {
  left: 128px;
}

li:nth-child(3):hover ~ .slider {
  left: 248px;
}

li:nth-child(4):hover ~ .slider {
  left: 368px;
}

li:nth-child(5):hover ~ .slider {
  left: 488px;
```

```
}

@keyframes waves {
  from {
    clip-path: polygon(
      0% 17%,
      9% 10%,
      18% 4%,
      30% 0%,
      43% 1%,
      49% 4%,
      57% 7%,
      66% 10%,
      78% 11%,
      89% 11%,
      96% 9%,
      100% 7%,
      100% 100%,
      0% 100%
    );
  }
  50% {
    clip-path: polygon(
      0% 4%,
      6% 9%,
      13% 13%,
      23% 15%,
      31% 16%,
      42% 15%,
      49% 13%,
      61% 10%,
      71% 5%,
      81% 3%,
      90% 2%,
      100% 5%,
      100% 100%,
      0% 100%
    );
  }
  to {
    clip-path: polygon(
      0% 17%,
      9% 10%,
      18% 4%,
      30% 0%,
      43% 1%,
      49% 4%,
      57% 7%,
      66% 10%,
      78% 11%,
      89% 11%,
      96% 9%,
      100% 7%,
      100% 100%,
      0% 100%
    );
  }
}
```

```
.sticky-header {
  position: absolute;
  top: 0px;
  left: 0px;
  width: 100%;
}

button {
  margin: 20px;
}
.custom-btn {
  width: 130px;
  height: 40px;
  color: #fff;
  border-radius: 5px;
  padding: 10px 25px;
  font-family: 'Lato', sans-serif;
  font-weight: 500;
  background: transparent;
  cursor: pointer;
  transition: all 0.3s ease;
  position: relative;
  display: inline-block;
   box-shadow:inset 2px 2px 2px 0px rgba(255,255,255,.5),
   7px 7px 20px 0px rgba(0,0,0,.1),
   4px 4px 5px 0px rgba(0,0,0,.1);
  outline: none;
}

.btn-6 {
  background: rgb(247,150,192);
background: radial-gradient(circle, rgba(247,150,192,1) 0%, rgba(118,174,241,1)
100%);
  line-height: 42px;
  padding: 0;
  border: none;
}
.btn-6 span {
  position: relative;
  display: block;
  width: 100%;
  height: 100%;
}
.btn-6:before,
.btn-6:after {
  position: absolute;
  content: "";
  height: 0%;
  width: 1px;
 box-shadow:
   -1px -1px 20px 0px rgba(255,255,255,1),
   -4px -4px 5px 0px rgba(255,255,255,1),
   7px 7px 20px 0px rgba(0,0,0,.4),
   4px 4px 5px 0px rgba(0,0,0,.3);
}
.btn-6:before {
  right: 0;
  top: 0;
```

```
  transition: all 500ms ease;
}
.btn-6:after {
  left: 0;
  bottom: 0;
  transition: all 500ms ease;
}
.btn-6:hover{
  background: transparent;
  color: #76aef1;
  box-shadow: none;
}
.btn-6:hover:before {
  transition: all 500ms ease;
  height: 100%;
}
.btn-6:hover:after {
  transition: all 500ms ease;
  height: 100%;
}
.btn-6 span:before,
.btn-6 span:after {
  position: absolute;
  content: "";
  box-shadow:
   -1px -1px 20px 0px rgba(255,255,255,1),
   -4px -4px 5px 0px rgba(255,255,255,1),
   7px 7px 20px 0px rgba(0,0,0,.4),
   4px 4px 5px 0px rgba(0,0,0,.3);
}
.btn-6 span:before {
  left: 0;
  top: 0;
  width: 0%;
  height: .5px;
  transition: all 500ms ease;
}
.btn-6 span:after {
  right: 0;
  bottom: 0;
  width: 0%;
  height: .5px;
  transition: all 500ms ease;
}
.btn-6 span:hover:before {
  width: 100%;
}
.btn-6 span:hover:after {
  width: 100%;
}
table th {
  width: 200px; /* 设置宽度为100像素 */
}
table {
  width: 600px;
  background-color: rgba(0, 0, 0, 0.5); /* 设置背景颜色为半透明的灰色 */
}
.ytable{
```

```
  height: 650px;
  overflow-y: scroll;
}
table td{
  text-align: center;
  color: white;
  font-size: 20px;
  animation: ease-in;
}

.container {
    align-items: center;
    justify-content: center;
  position: center;
  display: block;
}

.item {
  position: center;
  font-size: 20px;
  color: white;
}

.center-table {
  margin: 0 auto;
}

.text-container{
  background-color: rgba(0, 0, 0, 0.5); /* 使用 RGBA 颜色值，设置半透明的黑色背景 */
  padding: 10px; /* 添加一些内边距，使文字与背景之间有间隔 */
}

.gacha-times {
  font-weight: bold;
  font-size: 45px;
  color: #ff9900;
}
.gacha-five {
  font-weight: 700;
  font-size: 60px;
  color: #ffa100;
}
.gacha-four {
  font-weight: 700;
  font-size: 60px;
  color: #a100ff;
}
.gacha-three {
  font-weight: 700;
  font-size: 60px;
  color: #0065ff;
}
.total-power {
  font-weight: 700;
  font-size: 75px;
  color: #ff0000;
}
.total-result {
```

```
  font-weight: 700;
  font-size: 75px;
  color: #ffffff;
}
```

my_gacha_result.css

```
body {
    background-image: url('../others/fate.png');
    background-size: cover;
}
```

style.css

```
@charset "utf-8";

* {
    margin: 0;
    padding: 0;
}

body {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background: url("../others/index.jpg") no-repeat;
    background-size: cover;
}

.box {
    border-radius: 20px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 350px;
    height: 380px;
    border-top: 1px solid rgba(255,255,255,0.5);
    border-left: 1px solid rgba(255,255,255,0.5);
    border-bottom: 1px solid rgba(255,255,255,0.2);
    border-right: 1px solid rgba(255,255,255,0.2);
    backdrop-filter: blur(10px);
    background: rgba(50,50,50,0.2);
}

.box > h2 {
    color: rgba(255,255,255,0.9);
    margin-bottom: 20px;
}
/*莫名其妙h2没法生效，所以把这个样式手动加到index的header里面去了*/
.box .input-box {
    display: flex;
    flex-direction: column;
    justify-content: center;
```

```
    align-items: start;
    margin-bottom: 10px;
}
.box .input-box > label {
    margin-bottom: 5px;
    color: rgba(255,255,255,0.9);
    font-size: 13px;
}

.box .input-box > input {
    box-sizing: border-box;
    color: rgba(255,255,255,0.9);
    font-size: 14px;
    height: 35px;
    width: 250px;
    background: rgba(255,255,255,0.3);
    border: 1px solid rgba(255,255,255,0.5);
    border-radius: 5px;
    transition: 0.2s;
    outline: none;
    padding: 0 10px;
    letter-spacing: 1px;
}

.box .input-box > input:focus {
    border: 1px solid rgba(255,255,255,0.8);
}

.box .btn-box {
    width: 250px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: start;
}

.box .btn-box > a {
    font-size: 14px;
    text-decoration: none;
    color: rgba(255,255,255,0.9);
    transition: 0.2s;
    width: 250px;
    text-align: end;
}

.box .btn-box > a:hover {
    color: rgba(255,255,255,1);
}

.box .btn-box > div {
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: start;
    margin-top: 20px;
}

.box .btn-box > div > button {
```

```
    width: 95px;
    height: 35px;
    border: 1px solid rgba(75, 128, 231, 0.8);
    background: rgba(58, 90, 197, 0.5);
    color: rgba(255,255,255,0.9);
    border-radius: 5px;
    transition: 0.2s;
    left: auto;
}

.box .btn-box > div > button:nth-of-type(2) {
    margin-left: -8px;
}

.box .btn-box > div > button:hover {
    border: 1px solid rgba(248, 108, 76,0.8);
    background: rgba(248, 108, 76,0.5);
}
```

helpless.css

```
body {
    background-image: url('../others/background.jpg');
    background-size: cover;
}
```

gacha.css

```
body {
    background-image: url('../others/705836.jpg');
    background-size: cover;
    position: relative;
}

.for_button{
    position: relative;
    left: 42%;
    top: auto;
}

.image-slider {
  width: 50%;
  height: 200px;
  overflow: hidden;
  position: relative;
    top: 40px;
}

.image-slider img {
  width: 100%;
  height: 100%;
  object-fit: cover;
    display: inline-block;
  transition: transform 0.3s ease;
}
```

```
.image-slider .prev-btn,
.image-slider .next-btn {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  font-size: 24px;
  color: #fff;
  background-color: rgba(0, 0, 0, 0.5);
  padding: 8px;
  cursor: pointer;
}

.image-slider .prev-btn {
  left: 16px;
}

.image-slider .next-btn {
  right: 16px;
}
```

后端代码完整版

test.py

```
from flask import Flask, g, session
from flask import render_template
from flask_sqlalchemy import SQLAlchemy
from flask import request
from sqlalchemy import text
import random

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'

# Connection credentials
HOSTNAME = "127.0.0.1"
PORT = 3306
USERNAME = "root"
PASSWORD = "1a8a8a7415157aaaBbb"
DATABASE = "flask"

# configuring our database uri

app.config['SQLALCHEMY_DATABASE_URI'] = f"mysql+pymysql://{USERNAME}:
{PASSWORD}@{HOSTNAME}:{PORT}/{DATABASE}?charset=utf8mb4"

db = SQLAlchemy(app)

# with app.app_context():
#     with db.engine.connect() as conn:
#         rs = conn.execute(text("select 1"))

def generate_number():
   choices = ["3", "4", "5"]
   probabilities = [0.95, 0.036, 0.014]
   return random.choices(choices, probabilities)[0]
```

```
def takeSecond(elem):
    return elem[1]

@app.route('/gacha', methods=['GET', 'POST'])
def gacha():
    if request.method == 'GET':
        print("get gacha")
        return render_template("gacha.html")
    else:
        data = request.get_json()  # 获取前端发送的JSON数据
        received_string = data['data']  # 获取字符串数据
        return_string = ""
        username = session.get('username')
        print("后端gacha收到：" + received_string)
        print("username:"+session.get('username'))
        # 这里根据当前用户的username，去用户表查询其user_id，然后再去卡池执行抽卡逻辑，拼接
获得新的记录，最后插入gacha_result中
        # 同时，这里需要把插入的card_name全部记录，以便于输出和显示
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("SELECT user_id AS result FROM user WHERE user_name =
:username")
                rs = conn.execute(stmt, {"username": username})
                userid= rs.fetchone()[0] # 这里获取到了当前用户的id
                for i in range(0,10):
                    gacha_result_once = generate_number()
                    if gacha_result_once == "3":
                        stmt = text("SELECT * FROM gacha_list WHERE card_status = 3
ORDER BY RAND() LIMIT 1;")
                        rs = conn.execute(stmt, {"username": username})
                        row = rs.fetchone()
                        cardid = row[0]  # 获取第一列 card_id
                        cardname = row[1]  # 获取第二列 card_name
                        print(cardid)
                        print(cardname)
                        return_string = return_string + cardname + " "
                    elif gacha_result_once == "4":
                        stmt = text("SELECT * FROM gacha_list WHERE card_status = 4
ORDER BY RAND() LIMIT 1;")
                        rs = conn.execute(stmt, {"username": username})
                        row = rs.fetchone()
                        cardid = row[0]  # 获取第一列 card_id
                        cardname = row[1]  # 获取第二列 card_name
                        print(cardid)
                        print(cardname)
                        return_string = return_string + cardname + " "
                    elif gacha_result_once == "5":
                        stmt = text("SELECT * FROM gacha_list WHERE card_status = 5
ORDER BY RAND() LIMIT 1;")
                        rs = conn.execute(stmt, {"username": username})
                        row = rs.fetchone()
                        cardid = row[0]  # 获取第一列 card_id
                        cardname = row[1]  # 获取第二列 card_name
                        print(cardid)
                        print(cardname)
                        return_string = return_string + cardname + " "
```

```
                    stmt = text("INSERT INTO gacha_history
VALUES(:userid,:cardid,CURRENT_TIMESTAMP)")
                    rs = conn.execute(stmt, {"userid": userid, "cardid": cardid})
                    print("抽到了："+str(cardid))
                conn.commit()
                conn.closed
                print("userid："+str(userid))
        return session.get('username')+"上次十连的结果："+return_string

@app.route('/', methods=['GET', 'POST'])
def index():
    print("????")
    if request.method == 'GET':
        return render_template("index.html",data="default")
    else:
        print("<<<")
        login_result = 0 # 0->没有用户，进行注册，1->有用户，登录成功，2->有用户，登录失败
        login_result = login_request()
        print("<!<")
        if login_result == 0:
            msg = "新用户注册成功"
        elif login_result == 1:
            msg = "登录成功"
        else:
            msg = "登录失败，请检查密码重新输入"
        return render_template("index.html",data=msg)

def login_request():
    # 接收post请求的form表单参数
    username = request.form.get('username')
    userpass = request.form.get('password')
    print(str(username))
    print(str(userpass))
    with app.app_context():
        with db.engine.connect() as conn:
            stmt = text("SELECT IF(COUNT(*) > 0, 1, 0) AS result FROM user WHERE
user_name = :username")
            rs = conn.execute(stmt, {"username": username})
            whether_registed = rs.fetchone()[0]
            conn.commit()
            conn.closed
    if whether_registed > 0: # 这是用户已经注册过的情况，数据库内存储了用户的信息（主要是
用户名，用户注册也就看这个，然后用户id好像没啥用了）接下来一步应该是检验密码
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("SELECT COUNT(*) AS count FROM user WHERE user_name =
:username AND user_pass = :userpass;")
                rs = conn.execute(stmt, {"username": username,"userpass":userpass})
                pass_correct = rs.fetchone()[0]
                if pass_correct > 0: # 用户登录成功
                    login_result = 1
                    session['username'] = username
                    session['userpass'] = userpass
                    print("登录成功")
                else:
                    login_result = 2
                    print("输入密码错误，登录失败")
                conn.commit()
```

```
            conn.closed
    else: # 用户没有注册过的情况
        print("没注册过，帮他注册一个")
        login_result = 0
        print("账户名："+username+" "+"密码："+userpass)
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("INSERT INTO user (user_name, user_pass) VALUES
(:username, :userpass);")
                rs = conn.execute(stmt, {"username": username,"userpass":userpass})
                conn.commit()
                conn.closed
        session['username'] = username
        session['userpass'] = userpass
    return login_result

@app.route('/my_gacha_result', methods=['GET', 'POST'])
def my_gacha_result():
    # 这里首先把gacha_history里面所有属于userid的记录获取出来，然后用表格展示（数据全部传到
前端）
    # 但是在这之前需要先把userid设置为session的全局变量()难蚌，感觉可以在前面登录和注册的功
能那里加一个获取id的模块
    username = session.get('username')
    with app.app_context():
        with db.engine.connect() as conn:
            stmt = text("SELECT user_id FROM user WHERE user_name = :username;")
            rs = conn.execute(stmt, {"username": username})
            userid = rs.fetchone()[0]
            session['userid'] = userid # 此处获取了userid，去表格查询其所有的抽卡记录
            stmt = text("SELECT * FROM gacha_history WHERE user_id = :userid;")
            rs = conn.execute(stmt, {"userid": userid})
            history_list = rs.fetchall() # 此处获取一个list，其中每个元素都是一个元组，组
成形式：(userid, cardid, obtain_time)，但是这个obtain_time比较特殊
            return_list = []
            # 接下来，通过对元组重新赋值，将list中每一个元组转化为(card_name, card_status,
obtain_time),然后把list传给前端
            for record in history_list:
                cardid = record[1]
                obtain_time = record[2]
                stmt = text("SELECT card_name, card_status FROM gacha_list WHERE
card_id =:cardid;")
                rs = conn.execute(stmt, {"cardid": cardid})
                card = rs.fetchone()
                cardname = card[0]
                cardstatus = card[1]
                return_list.append((cardname, cardstatus, obtain_time))
            stmt = text("SHOW FIELDS FROM gacha_history")
            rs = conn.execute(stmt)
            labels = rs.fetchall()
            labels = [l[0] for l in labels]
            # 这段代码获取表头，但是吧，应该可以在前端写一下，把表头单独展示，而不是在表格展示
            # 因为现在的滚动表格展示，只要一滚动，表头就看不见了，最好的方法其实是让表头固定，
然后表格内容滑动
            # 有空再做
            conn.commit()
            conn.closed
    labels[0] = "灵基/礼装名称"
    labels[1] = "稀有度"
```

```
    labels[2] = "获取时间"
    return_list.sort(key=takeSecond,reverse=True)
    return
render_template("my_gacha_result.html",labels=labels,content=return_list)

@app.route('/helpless1', methods=['GET', 'POST'])
def helpless1():
    if request.method == 'GET':
        gacha_times = 0
        userid = session['userid']
        user_five = 0
        user_four = 0
        user_three = 0
        user_total_score = 0
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("SELECT COUNT(*) AS count FROM gacha_history WHERE
user_id = :userid;")
                rs = conn.execute(stmt, {"userid": userid})
                gacha_times = rs.fetchone()[0]  # 获取该用户总的抽卡次数
                stmt = text("SELECT * FROM gacha_history WHERE user_id = :userid;")
                rs = conn.execute(stmt, {"userid": userid})
                user_gacha_history = rs.fetchall()  # 获取该用户所有的抽卡记录，做分析和展
示
                for item in user_gacha_history:
                    cardid = item[1]
                    stmt = text("SELECT card_status FROM gacha_list WHERE card_id
=:cardid;")
                    rs = conn.execute(stmt, {"cardid": cardid})
                    card = rs.fetchone()
                    cardstatus = card[0]
                    if cardstatus == 5:
                        user_five = user_five + 1
                    elif cardstatus == 4:
                        user_four = user_four + 1
                    elif cardstatus == 3:
                        user_three = user_three + 1
        user_total_score = user_three * 1 + user_four * 5 + user_five * 10
        user_average_score = gacha_times * 1 * 0.95 + gacha_times * 5 * 0.036 +
gacha_times * 10 * 0.014
        user_average_score = int(user_average_score)
        if user_average_score < user_total_score:
            result = "欧"
        else:
            result = "非"
        conn.commit()
        conn.closed
        return render_template("helpless1.html", result=result,
gacha_times=gacha_times, user_five=user_five,
                               user_four=user_four, user_three=user_three,
user_average_score=user_average_score,
                               user_total_score=user_total_score)
    else: # 查询数据库，把该用户的所有抽卡记录删除
        userid = session['userid']
        with app.app_context():
            with db.engine.connect() as conn:
                stmt = text("DELETE FROM gacha_history WHERE user_id=:userid")
                rs = conn.execute(stmt, {"userid": userid})
```

```
            conn.commit()
            conn.closed
    return render_template("helpless1.html")

@app.route('/helpless2', methods=['GET', 'POST'])
def helpless2():
    return render_template("helpless2.html")

if __name__ == '__main__':
    app.run()
```

有待改进的地方

数据量太少

原因是目前的数据都是手动输入的，所以卡池的数据量很少，显得很单调

解决方法是写个python脚本，自动化爬，边爬边执行INSERT，把数据库塞满

代码书写不规范之后端

有很多常量，在代码中应当是共享的，然而我在书写的时候单独赋值了，这会导致后期想修改这个全局里的常量时，需要在整个项目代码里找到用到这个常量的地方然后全部修改，非常麻烦

举例：
在test.py中，我写了这样一个函数：

```
def generate_number():
    choices = ["3", "4", "5"]
    probabilities = [0.95, 0.036, 0.014]
    return random.choices(choices, probabilities)[0]
```

这个函数的本意是根据设定的概率，随机生成3，4或5，然后返回给后端让其去执行抽卡逻辑，也就是说这个probabilities的list里面的实际上是**整个项目的卡池概率。**

再看这里计算user_average_score的代码，也就是计算平均运气下用户得分，以判断用户是运气好还是运气坏的代码：

```
user_total_score = user_three * 1 + user_four * 5 + user_five * 10
      user_average_score = gacha_times * 1 * 0.95 + gacha_times * 5 * 0.036 +
gacha_times * 10 * 0.014
      user_average_score = int(user_average_score)
```

这里计算user_average_score的时候，明显**使用了之前设定的卡池概率**，然而，我这里是手写的概率，也就是说如果卡池概率改变，想要再去计算这个user_average_score的话，还需要跑到这里来，把代码手动改一下才行，这一点感觉上很奇怪，一是自己改的时候容易忘记，另一方面如果别人来看这个代码，会很难理解

我想到的解决方法是，每一个卡池的概率设置为一个list，然后设定一个全局的总的字典来存放这些list（key对应卡池名，value对应卡池概率的列表），然后把generate_number()这个函数修改一下，改为：

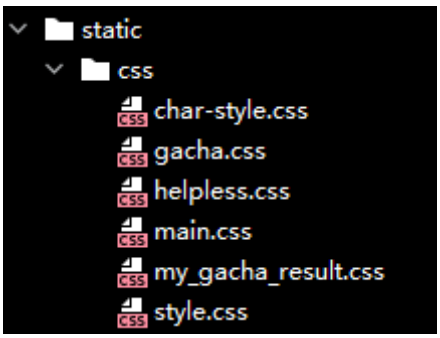
```
def generate_number(problist):
    choices = ["3", "4", "5"]
    probabilities = problist
    return random.choices(choices, probabilities)[0]
```

这样的话，直接从外界把概率列表传给函数就可以了，不同卡池设定不同概率也很方便。

计算user_average_score的函数也进行修改，其入口参数要放概率列表，然后内部直接进行一个score的计算就可以了。

代码书写不规范之前端

前端最不规范的地方是在CSS部分



乍一看我的项目组织井井有条，除了文件命名有些地方奇奇怪怪以外，还是做了文件分类的。

但是我在实际使用css样式的时候，有很多不合规范的写法，比如我单独开了一个char-style.css文件，本意是用于存储字体相关的css样式，但是实际上绝大多数的字体样式我都写在了main.css里面。诸如此类还有很多，因此css样式的管理还是挺混乱的。

实验总结

本次实验实现了一个基于前后端技术的抽卡模拟器，让我对于前后端开发技术、对于数据库操作有了进一步的认识。

最初，我尝试使用vue+django实现前后端分离的开发，然而，我缺乏前后端开发的一些基本的知识和认识，导致我使用vue写前端的时候非常痛苦，花了一段时间才弄懂前端的html，css，javascript以及他们在vue文件中的使用，以及webpack和cdn也是，最开始一头雾水；后端django一样让我一头雾水，一窍不通，零零碎碎做了一周多，直接选择开始重做，使用python flask写了四天不到，就写的七七八八了，主要是flask对于前后端耦合度要求不高的情况下，对于不熟悉前后端开发，且对python编程有经验的同学很友好。