# 健身房会员管理系统软件工程化文档

# 1. 项目概述

## 1.1 项目背景

随着健身行业的快速发展，健身房会员数量不断增加，传统的人工管理方式已无法满足现代化健身房的管理需求。为提高管理效率、优化会员体验，我们团队开发了这套健身房会员管理系统。

## 1.2 项目目标

1. **实现会员信息的数字化管理**： 将会员的各类信息，包括基本资料、消费记录、会员等级等进行数字化存储，方便快捷地进行查询、修改、统计等操作，提高信息管理的准确性和效率。
2. **提供便捷的课程预约功能**： 为会员打造简单易用的课程预约平台，会员可随时随地查看课程信息并进行预约，同时支持在规定时间内取消预约，提升会员的使用便利性。
3. **支持管理员对会员和课程的全面管理**： 赋予管理员强大的管理权限，使其能够对会员信息进行全方位管理，包括新增、编辑、删除等操作；对课程信息进行动态管理，如添加新课程、调整课程安排、修改课程价格等。
4. **提供友好的用户界面和交互体验**： 精心设计系统界面，采用简洁美观的布局和直观易懂的操作流程，降低用户的学习成本，使用户能够轻松上手，享受良好的交互体验。
5. **确保系统安全性和数据一致性**： 通过多种安全技术手段，保障系统数据的安全性，防止数据泄露和非法访问；采用合理的数据处理机制，确保数据在各种操作下的一致性和完整性。

# 2. 系统架构

本系统采用前后端分离的架构模式，这种架构有助于提高开发效率，便于系统的维护和扩展。

## 2.1 前端

使用 HTML5、Bootstrap 和 Jinja2 模板引擎构建前端页面。

HTML5 作为新一代的超文本标记语言，提供了丰富的语义化标签，用于构建页面结构；

Bootstrap 是一款流行的前端框架，能够快速创建响应式、移动设备优先的网站界面；

Jinja2 模板引擎则用于实现动态页面渲染，可根据后端传递的数据生成个性化的页面内容。

## 2.2 后端

后端采用 Python Flask 框架进行开发。

Flask 是一个轻量级的 Web 应用框架，具有简洁灵活的特点，便于快速搭建后端服务。通过该框架，实现业务逻辑处理、与数据库交互以及提供 API 接口等功能。

## 2.3 数据库

使用 PostgreSQL 和MySQL关系型数据库存储系统数据。

由于我们刚开始是在本地进行开发，用的是PostgreSQL以及连接云数据库 https://supabase.com/dashboard/project/gmevpselpkclylnjgity，Supabase提供了三种连接方式，分别是直接连接（仅允许IPv6地址，端口为5432）、事务池（兼容IPv4地址，端口为6543）、会话池（仅允许IPv4地址，端口为5432）。

后续我们转成PythonAnywhere替代本地开发环境，但由于是在平台的免费开发模式下进行，只允许出站连接到特定端口：80和443，而PythonAnywhere的服务器是IPv4网络，与Supabase云数据库的三种连接方式均不适配，因此我们的在线版本将数据库转成PythonAnywhere平台自带的免费MySQL数据库。本地版本仍可以使用Supabase云数据库，但由于逻辑完全相同，后续仅对在线版本进行说明即可。

# 3. 项目管理与协作

## 3.1 团队协作

1. **功能模块分工**：将整个项目按照功能模块划分为多个子任务，如会员管理模块、课程管理模块、预约管理模块等，团队成员各自负责不同的功能模块开发，明确分工，提高开发效率。

2. **定期代码审查和进度同步**：每周定期组织代码审查会议，团队成员相互检查代码，发现并解决代码中的问题，提高代码质量；同时进行进度同步，汇报各自任务的完成情况，及时协调解决开发过程中遇到的问题，确保项目按计划推进。

## 3.2 项目管理工具

1. **word 文档**：用于记录项目需求、制定详细的任务分配计划和编写技术文档等。通过 word 文档，将项目需求进行清晰明确的描述，制定每个阶段的任务清单和时间节点，方便团队成员了解项目目标和自己的工作任务。
2. **微信**：作为团队日常沟通的主要工具，方便团队成员随时交流问题、分享想法和沟通项目进展情况，确保信息及时传递和沟通的顺畅。

# 4. 自动化创建和部署

## 4.1 前端自动化创建

- 使用Bootstrap框架快速构建响应式界面
- 通过Jinja2模板引擎实现动态页面渲染

## 4.2 后端自动化创建

- 使用Flask框架快速搭建后端服务
- 通过virtualenv管理Python环境

## 4.3 数据库创建和连接

```
# 数据库连接配置
db_params = {
    "dbname": "gym_management",
    "user": "postgres",
    "password": "14789",
    "host": "localhost"
    #"port": "5432"
}

def get_db_connection():
    try:
        conn = psycopg2.connect(**db_params)
```

```
        return conn
    except psycopg2.OperationalError as e:
        print(f"An error occurred while connecting to the database: {e}")
        return None
```

## 4.4 前后端自动化运行

- 前端：通过Flask模板渲染自动运行
- 后端：使用Flask开发服务器运行

## 4.5 部署平台

PythonAnywhere是一个基于云端的Python开发和托管平台，提供完整的在线编程环境与Web应用部署服务。

生产环境配置：

```
# Flask应用关键配置
app.secret_key = os.environ.get('SECRET_KEY') # 从环境变量获取密钥
app.config['DEBUG'] = False # 生产环境关闭调试模式
```

# 5. 数据库设计

## 5.1 表格设计

```
-- 用户表
CREATE TABLE Users (
    username VARCHAR(100) PRIMARY KEY,
    password VARCHAR(100),
    role VARCHAR(10) -- 'admin'或'member'
);

-- 会员表
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    gender CHAR(1),
    age INT,
    contact VARCHAR(100),
    membership_level VARCHAR(50)
```

```
);

-- 课程表
CREATE TABLE Courses (
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    duration VARCHAR(50),
    coach VARCHAR(100),
    price DECIMAL(10, 2)
);

-- 预约表
CREATE TABLE Reservations (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,
    member_id INT,
    course_id INT,
    reservation_time DATETIME,
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

## 5.2 数据关系

- 用户与会员：一对一关系
- 会员与课程：多对多关系（通过预约表关联）

# 6. 前端设计

## 6.1 前端整体设计

- 使用Bootstrap构建响应式界面
- 采用导航栏+内容区布局
- 根据用户角色动态显示界面元素

## 6.2 主要功能模块实现

### 6.2.1 登录界面

```
<!DOCTYPE html>
<html>
<head>
    <title>登录</title>
```

```
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <h1>登录</h1>
    <form action="{{ url_for('login') }}" method="post">
        <div class="form-group">
            <label for="username">用户名</label>
            <input type="text" class="form-control" id="username" name="username"
required>
        </div>
        <div class="form-group">
            <label for="password">密码</label>
            <input type="password" class="form-control" id="password"
name="password" required>
        </div>
        <button type="submit" class="btn btn-primary">登录</button>
    </form>
</body>
</html>
```

### 6.2.2 课程预约功能

```
<!-- 课程列表中的预约按钮 -->
{% if session.role == 'user' %}
    <td>
        {% if is_reserved_by_current_user(course.course_id) %}
            <form action="{{ url_for('cancel_reservation',
course_id=course.course_id) }}" method="post">
                <button type="submit" class="btn btn-sm btn-warning">取消预约
</button>
            </form>
        {% else %}
            <form action="{{ url_for('make_reservation',
course_id=course.course_id) }}" method="post">
                <button type="submit" class="btn btn-sm btn-success">预约</button>
            </form>
        {% endif %}
    </td>
{% endif %}
```

# 7. 后端设计

## 7.1 主要功能实现

### 7.1.1 登录功能

```
@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute('SELECT role FROM Users WHERE username = %s AND password =
%s', (username, password))
        user = cursor.fetchone()
        if user:
            session['role'] = user[0]
            if user[0] == 'admin':
                return redirect(url_for('manage_members'))
            else:
                cursor.execute('SELECT member_id FROM Members WHERE name = %s',
(username,))
                member_info = cursor.fetchone()
                if member_info:
                    session['member_id'] = member_info[0]
                    return redirect(url_for('view_courses'))
                else:
                    return "会员未找到", 404
        else:
            return "用户名或密码错误", 401
    except psycopg2.Error as e:
        print(f"登录错误: {e}")
        return "登录失败", 500
    finally:
        cursor.close()
        conn.close()
```

### 7.1.2 课程预约功能

```
@app.route('/make_reservation/<int:course_id>', methods=['POST'])
def make_reservation(course_id):
    if 'role' not in session or session['role'] != 'user':
        return redirect(url_for('login'))
    member_id = session.get('member_id')
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute('''
            INSERT INTO Reservations (member_id, course_id, reservation_time)
            VALUES (%s, %s, NOW())
        ''', (member_id, course_id))
        conn.commit()
        return redirect(url_for('view_courses'))
    except psycopg2.Error as e:
        print(f"预约错误: {e}")
        conn.rollback()
        return "预约失败", 500
```

```
    finally:
        cursor.close()
        conn.close()
```

# 8. 测试与部署

## 8.1 测试方案

- 单元测试：测试各功能模块
- 集成测试：测试系统整体功能
- 用户验收测试：模拟用户操作流程

## 8.2 部署方案

1. 安装Python环境和依赖库
2. 配置PostgreSQL数据库及MySQL数据库
3. 运行Flask应用
4. 通过Nginx反向代理提供Web服务