# 员工管理系统 - 项目实现说明

## 1. 项目简介

本项目是一个基于 Python 的 Django 框架实现的简单员工管理系统。它提供了对部门和员工信息进行管理的基本功能，包括增、删、改、查（CRUD），并包含一个简单的登录认证机制。

## 2. 技术栈

* **后端**: Python 3, Django
* **前端**: HTML, Bootstrap 3
* **数据库**: SQLite (Django 默认)

## 3. 项目结构

项目主要由两个 Django App 组成：

* departmentManage: 用于管理部门信息。
* staffManage: 用于管理员工信息、用户登录、注销等。

employeeManagementSystem/  
|-- departmentManage/ # 部门管理App  
| |-- models.py # 部门数据模型  
| |-- views.py # 部门管理的视图函数  
| `-- ...  
|-- staffManage/ # 员工管理App  
| |-- models.py # 员工数据模型  
| |-- views.py # 员工管理的视图函数  
| |-- forms.py # 登录表单  
| |-- templates/ # HTML模板文件  
| |-- static/ # 静态文件 (CSS, JS)  
| `-- ...  
|-- employeeManagementSystem/ # 项目主配置  
| |-- settings.py # 项目配置  
| |-- urls.py # 主URL路由  
| `-- ...  
`-- manage.py # Django管理脚本

## 4. 数据模型

项目定义了两个核心的数据模型：

* **Department (部门)**: 在 departmentManage/models.py 中定义，只包含一个 name 字段用于存储部门名称。
* class Department(models.Model):  
   name = models.CharField(verbose\_name='部门名称', max\_length=64)
* **UserInfo (员工信息)**: 在 staffManage/models.py 中定义，包含了员工的详细信息，并通过外键 depart 与 Department 模型关联。
* class UserInfo(models.Model):  
   username = models.CharField(verbose\_name='用户名', max\_length=16)  
   password = models.CharField(verbose\_name='密码', max\_length=32)  
   age = models.IntegerField(verbose\_name='年龄')  
   account = models.DecimalField(...)  
   create\_time = models.DateTimeField(verbose\_name='入职时间')  
   # 外键关联到部门表  
   depart = models.ForeignKey(to='departmentManage.Department', ...)  
   gender = models.SmallIntegerField(...)

## 5. 核心功能实现

### 页面与数据的交互流程

本项目的核心是典型的 **MVT (Model-View-Template)** 设计模式，页面与数据的交互流程如下：

1. **用户请求**: 用户在浏览器中输入URL（例如 /user/list/）或点击页面上的链接/按钮。
2. **URL路由**: Django 根据 employeeManagementSystem/urls.py 中的配置，将请求转发给对应的视图函数（View）。
   * 例如，/user/list/ 请求会由 staffManage.views.user\_list 函数处理。
3. **视图处理 (View)**:
   * 视图函数接收到请求后，会执行相应的业务逻辑。
   * 如果需要操作数据（查询、创建、更新、删除），视图会通过 Django 的 ORM (Object-Relational Mapping) 与数据库中的模型（Model）进行交互。例如，models.UserInfo.objects.all() 会获取所有员工的信息。
   * 视图函数处理完数据后，会将数据打包成一个 context 字典。
4. **模板渲染 (Template)**:
   * 视图函数调用 render() 方法，将 context 数据传递给指定的 HTML 模板（Template）。
   * 在模板中，使用 Django 模板语言（如 {{ obj.username }} 和 {% for obj in queryset %}）将后端传递过来的数据显示在 HTML 页面上。
5. **响应返回**: 最终渲染好的 HTML 页面作为 HTTP 响应返回给用户的浏览器，用户即可看到最终的页面效果。

**以”员工列表”页面为例：**

* **请求**: 用户访问 /user/list/。
* **路由**: urls.py 匹配到 path('user/list/', staff\_views.user\_list)。
* **视图**: staff\_views.user\_list 函数执行 queryset = models.UserInfo.objects.all() 从数据库获取所有员工数据，然后调用 render(request, 'user\_list.html', {'queryset': queryset})。
* **模板**: user\_list.html 接收到 queryset，通过 {% for obj in queryset %} 循环遍历，将每个员工的信息填入表格中。
* **响应**: 生成的包含员工列表的完整 HTML 页面被发送回浏览器。

### 表单数据提交（以”添加员工”为例）

1. **GET 请求**: 用户访问 /user/add/，视图函数 user\_add 响应 GET 请求，渲染一个包含输入表单的 user\_add.html 页面。
2. **用户输入**: 用户在表单中填写信息，然后点击”提交”按钮。表单会以 POST 方法将数据提交回 /user/add/。
3. **POST 请求**: user\_add 视图函数接收到 POST 请求。
4. **数据处理**:
   * 通过 request.POST.get("username") 等方法获取用户提交的数据。
   * 调用 models.UserInfo.objects.create(...) 将新数据存入数据库。
5. **重定向**: 创建成功后，使用 redirect("/user/list/") 将用户重定向到员工列表页面，用户即可看到新添加的员工。

## 6. 如何运行项目

1. **激活虚拟环境**:

* D:\python\_venv\djano\Scripts\activate

1. **安装依赖**: (如果需要)

* pip install django

1. **初始化数据库**: 项目首次运行时，需要创建数据库表结构。

* python manage.py makemigrations  
  python manage.py migrate

1. **（可选）初始化示例数据**: 项目提供了一个 init\_db.py 脚本用于填充初始数据。

* python scripts/init\_db.py

1. **启动开发服务器**:

* python manage.py runserver

1. **访问项目**: 在浏览器中打开 http://127.0.0.1:8000/login/ 即可开始使用。