智慧上海电动自行车共享换电柜管理平台

数据库原理课程设计报告

学　　号　2050254

姓　　名　王钧涛

专　　业　计算机科学与技术

授课老师　李文根

目录

摘 要 上海2035规划[1]提出：“要建设更可持续的韧性生态之城，全面推动绿色低碳发展，降低碳排放”。在绿色低碳发展的思想指导下，电动自行车作为过去二轮轻便摩托车的替代产品受到了城市居民的广泛认可，截止2022年底，2022年中国电动自行车保有量将达到3.5亿辆，且市场需求仍将保持增长态势。同时，随着共享经济概念的火热，共享电瓶作为传统充电痛点的解决方案受到了城市群众尤其是外卖骑手的认可。在这种背景下，我希望能够为上海这座城市设计一款电动自行车共享换电柜管理平台，实现如下这些功能：为管理者提供集成的可视化的站位信息总览，包括地理分布情况、利用率情况及一些趋势图表，为使用者提供取还电一体的基于电瓶型号的详细共享电柜查询，为突发的电柜或电瓶故障提供工单处理服务；本次实验报告为最终报告设计，包含需求分析以及可行性分析、概念设计、逻辑设计、具体实现、成果展示、项目总结。

关键词：数据库；信息可视化设计；电动自行车换电服务：共享电柜

一.概述

1.1 课题背景

上海 2035 规划[2]提出：“要建设更可持续的韧性生态之城，全面推动绿色低碳发展，降低碳排放”。在绿色低碳发展的思想指导下，电动自行车作为过去二轮轻便摩托车的替代产品受到了城市居民的广泛认可。截止 2022 年底，2022 年中国电动自行车保有量将达到 3.5 亿辆。且市场需求仍将保持增长态势，未来三年，电动自行车替换量有望达到 8000 万辆，整体需求量约 1.8 亿辆，年均销售量超过 6000 万辆[3]。在此基础上，电动自行车换电柜作为共享经济的指导产物在 2019 年开始出现在上海城市中，在短短三年间得到了长足的发展。与传统的电动自行车充电方式相比，电动自行车换电有很多优势。首先，电池更换时间短，只需要几分钟就可以完成，相比之下，充电需要数小时。其次，换电可以解决电池寿命和充电时间的问题，使得电动自行车更加便捷和易于维护。以普陀区为例，在 2022 年 11 月《普陀区电动自行车充电设施建设实施方案》实施后，由区消防委牵头，在全区推进安装 30 万个充换电设施[4]。在大量换电设施推进的现状下，开发电动自行车换电系统管理平台就有其重要意义，不仅能用信息可视化的手段帮助管理者直观了解到目前城市换电功能的整体运行情况，同时也能让广大具有换电需求的用户快速地找到可用的换电桩，提升整体换电服务的使用体验。

1.2 编写目的`

本次课程选题《智慧上海电动自行车共享换电柜管理平台》，意在为电动自行车共享电柜换电系统的服务提供以下三点核心功能：

- 为管理者提供集成的可视化的站位信息总览，包括地理分布情况、利用率情况及一些趋势图表
- 为使用者提供取还电一体的基于电瓶型号的详细共享电柜查询
- 为突发的电柜或电瓶故障提供工单处理服务

在此基础上，《智慧上海电动自行车共享换电柜管理平台》将采取前后端分离的形式，采用网页形式进行数据可视化展示。

1.3 编写心得

需求分析、概念设计和逻辑设计在上学期的构思中已经成形了，这次课程设计主要的任务就是将这个《智慧上海电动自行车共享换电柜管理平台》从这份报告上实体化，变成一个可以运行的、美观的、完整的应用。为此，我采用了一系列先进的技术栈和工具，包括前端开发、后端开发、数据库和 API 设计。通过这次课程设计，我对数据库课程的学习有了更深刻的理解，也掌握了实际应用中的一些技能和经验。

在前端开发方面，我选择了基于 Vue3 和 Element UI 的技术栈。Vue3 是一个灵活、高效的 JavaScript 框架，它提供了丰富的工具和组件，使得前端开发更加简单和快速。Element UI 则是一个基于 Vue 的 UI 框架，它提供了美观、易用的界面组件，大大提高了用户体验。通过学习和使用这些工具，我能够快速搭建出可视化设计，满足用户的需求，并为他们提供良好的交互体验。

在后端开发方面，我选择了基于 FastAPI 和 SQLAlchemy 的技术栈。FastAPI 是一个高性能的 Python 框架，它提供了快速构建 Web API 的能力，并且具有异步请求处理的特性，能够有效地提高系统的并发性能。SQLAlchemy 是一个强大的 Python ORM 工具，它提供了对数据库的抽象和封装，使得数据库操作更加简单和灵活。通过这些工具的使用，我能够轻松设计和实现后端 API，并与数据库进行交互，为平台的功能提供强有力的支持。

在数据库选择方面，我采用了 MySQL 作为平台的数据库。MySQL 是一种开源的关系型数据库管理系统，具有稳定性高、性能好、易于使用等特点，非常适合于大规模数据存储和管理。我使用 Navicat 作为可视化工具，通过它我可以方便地查看和管理数据库的表结构、数据内容等信息，使得数据库的操作更加直观和便捷。

在 API 设计方面，我使用了 Apifox 平台进行整体管理和可用性测试。Apifox 是一个强大的 API 设计和管理工具，它提供了可视化的界面和丰富的功能，使得 API 的设计和管理变得更加简单和高效。通过使用 Apifox，我可以清晰地定义 API 的接口和参数，并进行灵活的测试和调试，确保 API 的稳定性和可靠性。

通过这次课程设计，我不仅掌握了前端开发、后端开发、数据库和 API 设计的基本概念和技能，还学会了如何将它们有机地结合起来，构建一个完整的应用系统。我深刻体会到了数据库在实际应用中的重要性和价值，它是支撑应用系统的核心，能够高效地存储和管理数据。

二.需求分析

2.1 功能需求

下面详细介绍本应用的功能需求。

1. 管理者站位信息总览：

- ✓ 提供集成的可视化界面，展示电动自行车共享换电柜的站位信息总览。
- ✓ 显示电动自行车换电柜的地理分布情况，可以在地图上标示各个换电柜的位置。
- ✓ 提供实时的利用率情况，包括换电柜的空闲率、占用率等数据，并以图表形式展示。
- ✓ 提供一些趋势图表，例如换电柜利用率的变化趋势，帮助管理者了解系统的运营情况。

2. 使用者共享电柜查询：

- ✓ 提供使用者可视化的界面，支持基于电瓶型号的详细共享电柜查询。
- ✓ 使用者可以输入电瓶型号进行查询，系统将返回与该电瓶型号兼容的共享电柜列表。
- ✓ 显示共享电柜的位置信息、电瓶存储量以及相关的使用说明等详细信息。
- ✓ 支持按照距离、电瓶存储量等条件进行排序和筛选，方便使用者选择最适合的共享电柜。

3. 故障工单处理服务：

- ✓ 提供突发的电柜或电瓶故障的工单处理服务，使得故障能够及时得到解决。
- ✓ 管理者可以提交电柜或电瓶故障的工单，并提供详细的故障描述和相关信息。
- ✓ 系统应记录工单的状态、优先级和处理进度，以便管理者随时跟踪和更新工单的状态。
- ✓ 工单分配给相应的技术人员进行处理，记录处理过程和结果，以便后续的工单分析和追踪。

此外，对系统还需要如下的需求：

可使用性需求：

1) 直观易用的界面：平台应具备直观、易用的用户界面，使管理者和使用者能够快速上手，并能轻松完成操作和查询。
2) 响应式设计：界面应能自适应不同的设备和屏幕尺寸，包括桌面、平板和手机等，以便用户在不同的设备上都能获得良好的使用体验。

3) 用户权限管理：平台应支持不同层级的用户权限管理，包括管理员和普通用户等，以保证各类用户只能访问和执行其所需的功能。

4) 错误处理和反馈：系统应提供明确的错误提示和反馈机制，对于用户输入的错误或操作异常，及时给出提示和解决方案，以减少用户的困惑和错误操作。

可维护性需求：

1) 模块化设计：平台应采用模块化的设计结构，将不同功能和模块进行分离和解耦，以便于进行维护和扩展。

2) 可读性和可维护性的代码：代码应具备良好的可读性和可维护性，包括合理的命名、注释清晰、代码结构清晰等，以便于理解和修改代码。

3) 定期的系统维护：系统应定期进行维护和更新，包括数据库的备份和优化、代码的版本管理和更新等，以保证系统的稳定性和可靠性。

性能需求：

1) 高并发处理能力：平台应具备较高的并发处理能力，能够同时处理大量的用户请求和数据操作，以确保系统的响应速度和稳定性。

2) 快速的数据查询和更新：数据库查询和更新操作应具备较高的执行效率和响应速度，以满足管理者和使用者对数据的实时查询和操作需求。

3) 良好的系统响应时间：平台的各项功能操作应具备较短的响应时间，以提供良好的用户体验和操作效率。

安全需求：

1) 数据安全性保护：平台应采取必要的安全措施，对用户的个人信息和敏感数据进行加密和保护，确保数据的安全性和隐私性。

2) 访问控制和身份认证：平台应具备访问控制机制和身份认证功能，确保只有经过授权的用户能够访问和操作相关数据和功能。

3) 防止恶意攻击和数据篡改：平台应采用安全的编码实践和防火墙等技术手段，防止恶意攻击和数据篡改行为，保证系统的安全性和完整性。

这些需求能够保证系统的易用性、可维护性、高性能和安全性，为用户提供良好的使用体验和保障。

2.2 数据字典

电动自行车（ECycle）

列名	数据类型
电动车 ID	唯一标识符
电动车型号	字符串
电瓶参数	字符串
拥有者	引用至 Owner

拥有者（Owner）

列名	数据类型
ID	唯一标识符
姓名	字符串
性别	字符串
驾驶年龄	整数

电瓶（Battery）

列名	数据类型
ID	唯一标识符
大小	字符串
电瓶参数	字符串
剩余电量	浮点数
状态	字符串
从属电瓶插槽 ID	引用至电瓶插槽

共享电柜（Cabinet）

列名	数据类型
ID	唯一标识符
位置	字符串
型号	字符串
供应商名	字符串
最大电瓶插槽数量	整数
经度	浮点数
纬度	浮点数

电瓶插槽（ySlot）

列名	数据类型
ID	唯一标识符
名字	字符串
从属电柜 ID	引用至共享电柜
可插电瓶参数	字符串
最大插入数量	整数

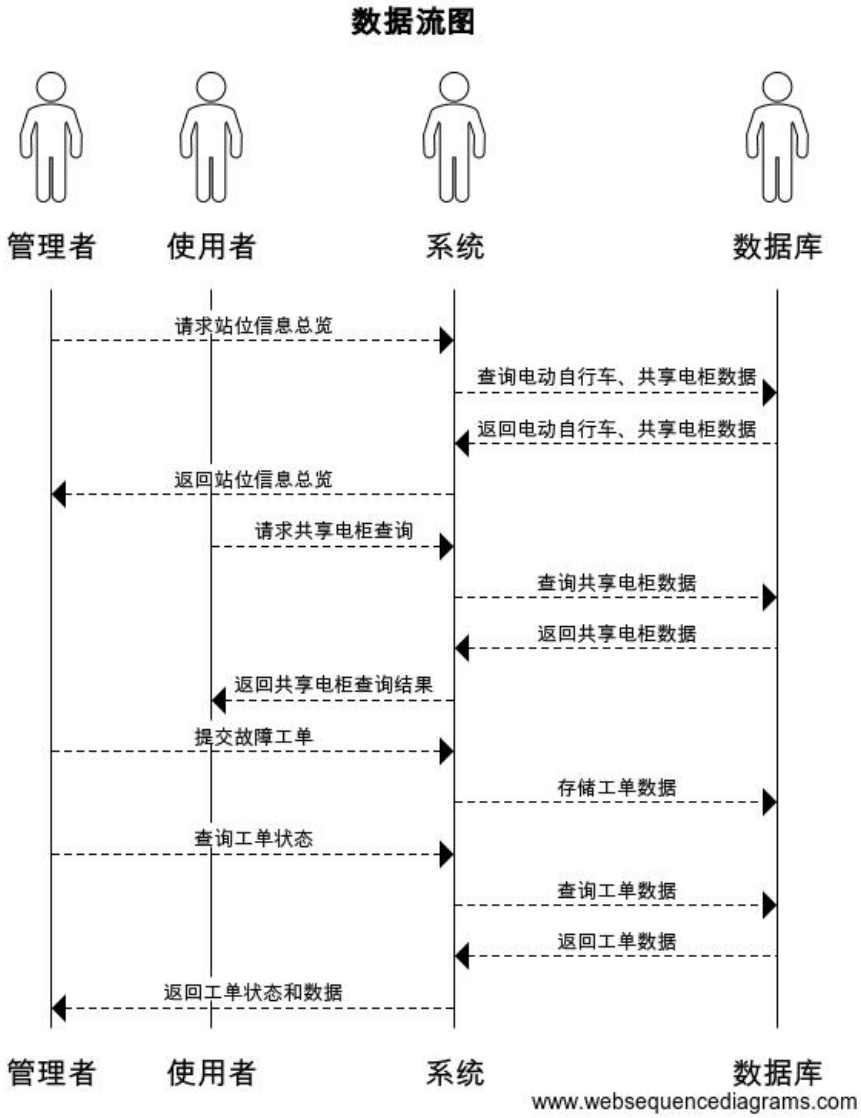
供应商（Company）

列名	数据类型
供应商名	字符串
所属地区	字符串

工单问题（WorkOrder）

列名	数据类型
工单 ID 号	唯一标识符
工单标题	字符串
工单详情	字符串
发起人 ID	引用至拥有者

2.3 数据流图



三.可行性分析

3.1 技术可行性

前端开发：

Vue3 是一种流行的前端 JavaScript 框架，具有灵活性、高效性和可扩展性。它支持组件化开发，可以轻松构建交互性强的用户界面。

Element UI 是一个基于 Vue 的 UI 组件库，提供了丰富的 UI 组件和样式，可帮助您快速构

建美观的前端界面。

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行时环境，具有高性能和可伸缩性。它适用于构建服务器端和网络应用程序。

后端开发：

FastAPI 是一个基于 Python 的现代、快速（高性能）的 Web 框架，适用于构建高性能的 API。它具有自动化的文档生成、类型注解和异步支持等特性，使开发更高效。

SQLAlchemy 是一个 Python 的 ORM（对象关系映射）库，提供了便捷的数据库操作和查询方式。它支持多种数据库后端，并提供了高层次的抽象，使数据库操作更简单和可维护。

数据库：

MySQL 是一个广泛使用的关系型数据库管理系统，具有稳定性、可靠性和良好的性能。它适用于存储和管理大量结构化数据。

Navicat 是一个可视化数据库管理工具，提供了直观的界面和丰富的功能，方便进行数据库的设计、查询和管理。

API 设计：

Apifox 是一个 API 设计和管理平台，它提供了一套工具和界面，帮助您定义、管理和测试 API。它可以提高开发效率，并确保 API 的可用性和稳定性。

综合来看，我选择的技术栈在功能和工具方面具备了该项目需求的潜力。Vue3、Element UI 和 Node.js 可以支持我构建直观且具有良好用户体验的前端界面。FastAPI 和 SQLAlchemy 提供了强大的后端开发工具，能够实现高性能的 API 设计和数据库操作。MySQL 作为可靠的数据库管理系统，适合我存储和管理大量数据。而 Apifox 平台则可以帮助我整体管理和测试 API 的可用性。

3.2 应用可行性

应用可行性有以下几个方面：

- ✓ 方便性和便捷性：我的应用提供了一个集成的可视化界面，方便管理者查看电动自行车共享换电柜的站位信息总览和利用率情况。对于使用者来说，他们可以通过应用快速地查询到符合其电瓶型号的共享电柜，并获取相关的详细信息。这种方便性和便捷性能够节省用户的时间和精力，提升使用体验。
- ✓ 数据可视化和趋势分析：我的应用通过可视化图表展示电动自行车共享换电柜的地理分

布情况、利用率情况以及趋势图表等。这种数据的可视化呈现可以帮助管理者直观地了解到系统的运营情况和趋势，帮助他们做出有效的决策和优化。

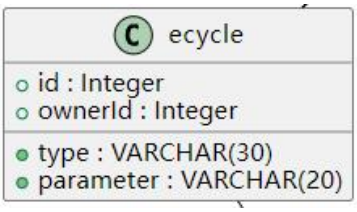
- ✓ 故障处理服务：我的应用提供突发的电柜或电瓶故障的工单处理服务。这意味着用户在遇到问题时可以快速提交工单，并得到及时的解决方案。这种故障处理服务可以提高系统的可靠性和稳定性，增强用户对系统的信任感。
- ✓ 绿色低碳出行意识：随着绿色低碳发展的推动，电动自行车作为一种环保出行方式受到了越来越多人的关注和选择。我的应用提供了方便的电动自行车共享换电柜管理平台，使得用户能够更便捷地使用电动自行车换电服务，提升电动自行车的使用体验，进而减少用户对私家燃油车的依赖，从而贡献于环保事业。

综上所述，人们可能会选择使用我的应用是因为它的方便性、数据可视化和趋势分析功能、故障处理服务以及对绿色低碳出行的支持。这些因素能够提升用户的使用体验，并满足他们对便捷、可靠和环保的需求。

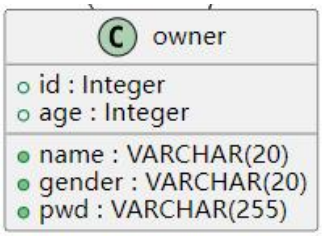
四.概念设计

4.1 实体

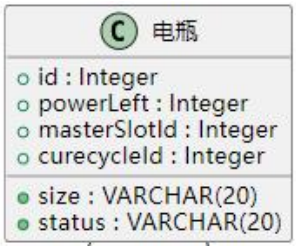
1. 电动自行车：代表一辆电动自行车，具有唯一的电动车 ID，电动车型号，电瓶参数和拥有者信息。它记录了电动自行车的相关属性和归属信息。



2. 用户：代表电动自行车的拥有者，包含了拥有者的 ID、姓名、性别和驾驶年龄等信息。它记录了电动自行车的所有者相关的个人信息。



3. 电瓶：代表电动自行车所使用的电瓶，具有唯一的电瓶 ID，大小，电瓶参数，剩余电量和状态等属性。它记录了电动自行车所使用的电瓶的相关信息。



4. 共享电柜：代表共享电动自行车电瓶的存放设备，包括电瓶插槽、位置、型号、供应商名、最大电瓶插槽数量以及经度和纬度等属性。它记录了共享电瓶存放设备的相关信息。

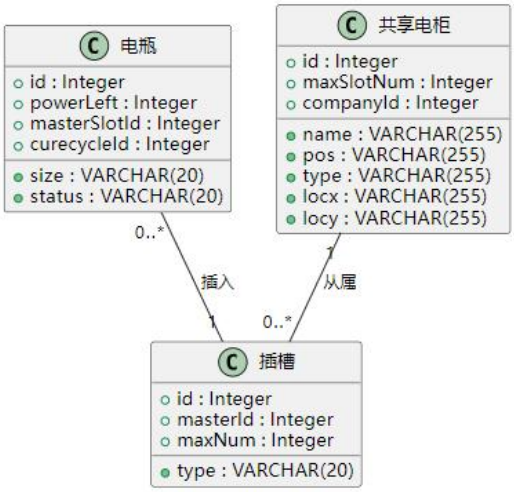
5. 电瓶插槽：代表共享电柜中的电瓶插槽，具有唯一的插槽 ID、名称、从属电柜 ID、可插电瓶参数和最大插入数量等属性。它记录了电瓶插槽的相关信息，包括可插入的电瓶类型和最大容量。

6. 供应商：代表电动自行车和共享电柜的供应商，包含供应商的 ID 和名称等信息。它记录了供应商的相关信息，用于标识和区分不同的供应商。

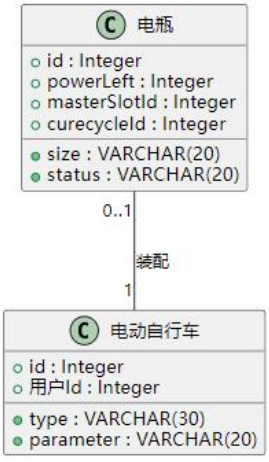
7. 工单问题：代表工单问题，包括工单 ID 号、工单标题、工单详情和发起人 ID 等属性。它记录了与电动自行车和共享电柜相关的问题工单，用于工单的管理和处理。

4.2 实体属性局部 E-R 图

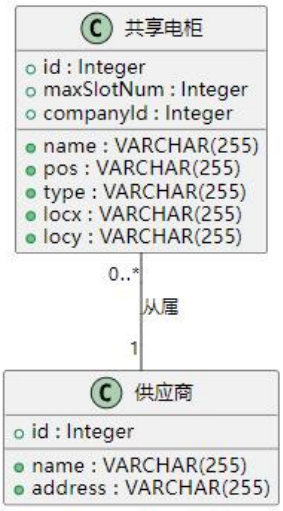
电柜联系：一个共享电柜可以含有一个或多个电瓶插槽，一个电瓶插槽可以插入 0 个或多个电瓶，其关系对应如下：



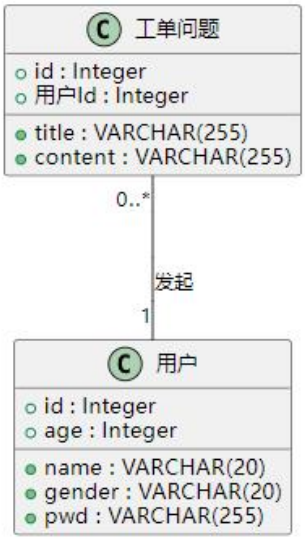
电动自行车联系：一个电动自行车可以装配 0 个或 1 个电瓶，其关系对应如下：



共享电柜从属联系：一个共享电柜属于一个确定的供应商：

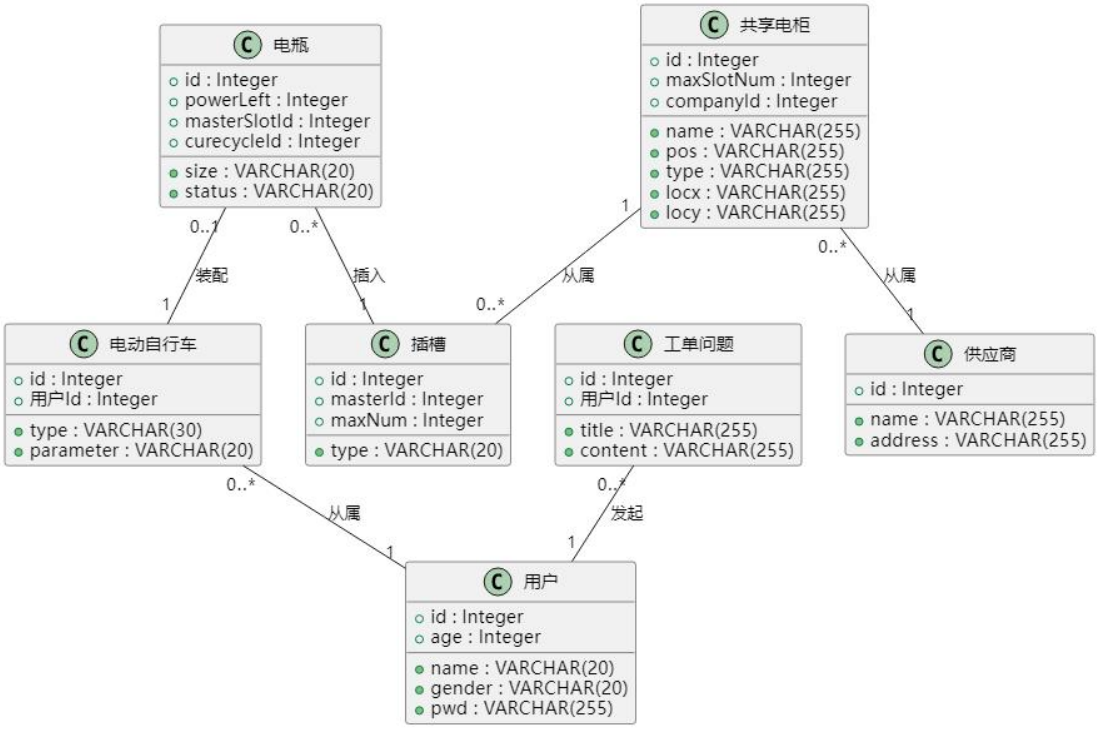


工单联系：一个用户可以发起一个或多个工单：



4.3 全局 E-R 图

全局 E-R 图如下图所示：



五.逻辑设计

5.1 E-R 图向关系模型的转变

在完成数据库的 E-R 模型设计后，将 E-R 图按照一定的规则转化为对应的关系模式，同时在合并冗余的前提下得到对应的表单。在将 E-R 图转变为关系模式时，主要遵循的是以下这些规则：

① 检查所有强实体集，为其建立对应的关系模式，关系必须都包含实体集的所有属性，且主码与实体集相同。

② 检查所有弱实体集，使用弱实体集及其依赖强实体集的主码属性建立对应的关系模式。

③ 检查所有联系集，依据映射基数决定是否建立单独关系。若映射为一对一或一对多，则在其中一项实体集的关系中添加另一项实体集（应当为“一”）的主码属性；若为多对多，则需要为该联系建立一个新关系，关系的两个属性分别为两个实体集的主码属性。

本次设计的 E-R 模型中，所有实体集均为强实体集，同时不存在多对多关系，因此可以构造如下关系模式：

battery (**id**, size, powerLeft, status, masterSlotId, curUserId)

cabinet (**id**, pos, type, maxSlotNum, companyId, locx, locy)

company (**id**, name)

ecycle (**id**, type, parameter, ownerId)

owner (**id**, name, gender, age)

slot (**id**, masterId, status, cabinetId)

workorder (**id**, title, content)

此外，用户的账号密码管理等内容采用现成的 MD5 加密存储模块进行，不放入本次的 mysql 数据库中，因此不在此另行给出。

5.2 数据模型的优化及规范化设计

优化后的表结构如下：

1. 电动自行车 (ECycle)

属性	字段名	类型	长度	约束
电动车 ID	id	Integer		Primary Key, Not Null
电动车型号	type	VARCHAR(30)	30	
电瓶参数	parameter	VARCHAR(20)	20	
拥有者	ownerId	Integer		Foreign Key (owner.id), Not Null

2. 用户 (Owner)

属性	字段名	类型	长度	约束
ID	id	Integer		Primary Key, Not Null
姓名	name	VARCHAR(255)	255	
性别	gender	VARCHAR(20)	20	
驾驶年龄	age	Integer		

3. 电瓶 (Battery)

属性	字段名	类型	长度	约束
ID	id	Integer		Primary Key, Not Null
大小	size	VARCHAR(20)	20	
电瓶参数	parameter	VARCHAR(20)	20	
剩余电量	powerLeft	Integer		Not Null

属性	字段名	类型	长度	约束
状态	status	VARCHAR(20)	20	
从属电瓶插槽 ID	masterSlotId	Integer		Foreign Key (slot.id)
当前用户 ID	curUserId	Integer		Foreign Key (ecycle.id)

4. 共享电柜 (Cabinet)

属性	字段名	类型	长度	约束
ID	id	Integer		Primary Key, Not Null
位置	pos	VARCHAR(255)	255	
型号	type	VARCHAR(255)	255	
最大电瓶插槽数量	maxSlotNum	Integer		
公司 ID	companyId	Integer		Foreign Key (company.id), Not Null
经度	locx	VARCHAR(20)	20	
纬度	locy	VARCHAR(20)	20	

5. 电瓶插槽 (Slot)

属性	字段名	类型	长度	约束
ID	id	Integer		Primary Key, Not Null
名字	name	VARCHAR(255)	255	
从属电柜 ID	masterId	Integer		Foreign Key (cabinet.id), Not Null
可插电瓶参数	type	Integer		
最大插入数量	maxNum	Integer		

6. 供应商 (Company)

属性	字段名	类型	长度	约束
供应商 ID	id	integer		Primary Key, Not Null
供应商名	name	VARCHAR(255)	255	
所属地区	Position	VARCHAR(255)	255	

7. 工单 (Workorder)

属性	字段名	类型	长度	约束
工单 ID 号	id	Integer		Primary Key, Not Null
工单标题	title	VARCHAR(255)	255	
工单详情	content	VARCHAR(255)	255	

属性	字段名	类型	长度	约束
发起人 ID	creatorId	Integer		

根据提供的实体集合，我们来分析每个实体的属性依赖关系，以确定是否符合第三范式（3NF）。

1. battery(**id**, size, powerLeft, status, masterSlotId, curECycleId)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

2. cabinet(**id**, name, pos, type, maxSlotNum, companyId, locx, locy)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

3. company(**id**, name, address)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

4. ecycle(**id**, type, parameter, ownerId)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

5. owner(**id**, name, gender, age, pwd)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

6. slot(**id**, masterId, maxNum, type)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

7. workorder(**id**, title, content, ownerId)

该实体没有明显的传递依赖关系，每个属性都直接依赖于主键 id。符合 3NF。

综上所述，根据提供的实体集合的属性依赖关系分析，每个实体都符合第三范式（3NF）。每个属性都直接依赖于主键，没有传递依赖关系存在。

六.项目管理

6.1 框架选择

前端开发：基于 Vue3 + Element UI + Node.JS 的可视化设计

后端开发：基于 FastAPI + SQLArchemy 的后端 API 设计与功能实现。

数据库：选用 MySql 进行对接，采用 Navicat 进行可视化查看。

API 设计：使用 Apifox 平台进行整体管理和可用性测试。

6.2 开发平台

开发环境为 Windows 10 x64 平台，前端服务器运行平台为 node 17.19.4，后端服务器运行平台为 fastapi 2.0.4。

七.系统实现

7.1 系统架构搭建

1、前端架构搭建：

① 安装 Node.js：Vue.js 是基于 Node.js 运行的，因此首先需要安装 Node.js。从 Node.js 官方网站（https://nodejs.org）下载 WIndows10 操作系统的安装程序，并按照提示进行安装。

② 安装 npm：Node.js 安装完成后，npm（Node 包管理器）也会一同安装。npm 是用于安装和管理 JavaScript 包的工具，使用它来安装 Vue 和其他相关依赖。

③ 安装完 Node.js 和 npm 后，执行以下步骤来搭建 Vue 系统环境：

1) 打开命令行界面（如命令提示符或终端）。

```
mkdir my-vue-app
cd my-vue-app
```

2) 创建一个新的目录来存放 Vue 项目，并进入该目录：

```
npm init -y
```

④ 安装完 Node.js 和 npm 后，执行以下步骤来搭建 Vue 系统环境：

```
npm install vue
```

⑤ 创建一个 JavaScript 文件来编写 Vue 应用的逻辑（例如，main.js）

⑥ 在命令行界面中，运行一个本地开发服务器来查看 Vue 应用：

```
npm install -g http-server
http-server
```

2、后端架构搭建

① 安装了 Python（建议使用 Python 3.7 或更高版本）。可以从 Python 官方网站（https://www.python.org）下载适合 Windows10 的安装程序，并按照提示进行安装。

② 打开命令行界面（如命令提示符或终端）。

③ 创建一个新的目录来存放 FastAPI 项目，并进入该目录：

④ 创建一个 Python 虚拟环境（可选但推荐），以便在项目中隔离依赖项，并将其激活

```
python -m venv venv
venv\Scripts\activate
```

⑤ 安装 FastAPI 和 Uvicorn（用于运行 FastAPI 应用的 ASGI 服务器）：

⑥ 在命令行界面中，使用 Uvicorn 运行 FastAPI 应用：

```
uvicorn main:app --reload
```

这将启动 Uvicorn 服务器，并运行 FastAPI 应用。main:app 表示 main.py 文件中的 app 变量，--reload 参数用于在代码发生更改时自动重新加载应用。

3、Mysql 搭建

按照以下步骤安装 MySQL：

① 打开浏览器，并访问 MySQL 官方网站（https://www.mysql.com）。

② 在官方网站上找到并点击“Downloads”（下载）链接。

③ 在“MySQL Community Downloads”（MySQL 社区版下载）页面上，向下滚动并找到“MySQL Community Server”（MySQL 社区服务器）部分。

④ 在“MySQL Community Server”部分，选择适合您操作系统的版本（如 Windows、macOS、Linux 等），并点击下载链接。

⑤ 下载完成后，根据操作系统的要求，运行下载的安装程序。

⑥ 在安装向导中，根据提示选择安装选项，并接受许可协议。

⑦ 在安装选项中，选择自定义安装路径和配置，或者保持默认设置并继续。

⑧ 等待安装程序完成 MySQL 的安装过程。

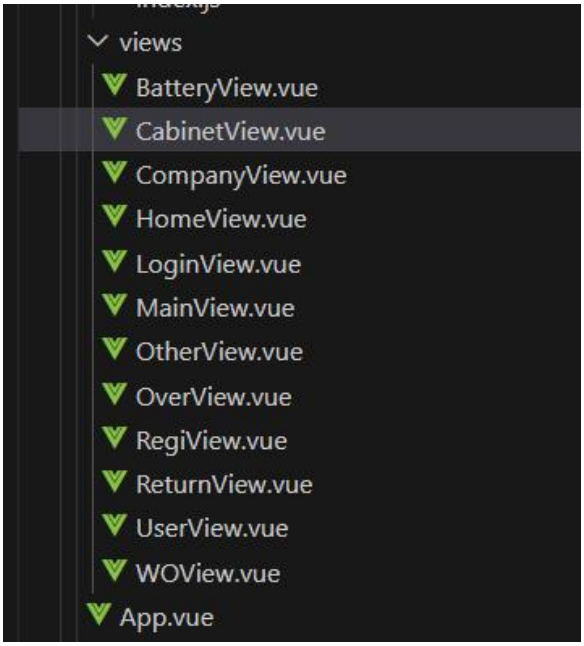
⑨ 完成安装后，MySQL 将作为一个服务在后台运行。可以在操作系统的服务列表中找到它，并启动或停止该服务。

7.2 系统逻辑设计

1. 前端逻辑

前端从 APP.js 中进入，然后从 APP.js 中根据不同的请求跳转到不同的页面，具体有如

下这些页面：



跳转函数在 index.js 文件中定义：

```
    name: 'home',
    component: () => import('../views/HomeView.vue')
  },
  {
    path: '/overview',
    name: 'overview',
    component: () => import('../views/OverView.vue')
  },
  {
    path: '/user',
    name: 'user',
    component: () => import('../views/UserView.vue')
  },
  {
    path: '/other',
    name: 'other',
    component: () => import('../views/OtherView.vue')
  },
  {
    path: '/return',
    name: 'return',
    component: () => import('../views/ReturnView.vue')
  },
  {
    path: '/available',
```

一个 Vue 文件包含三个主要部分：<template>、<script> 和 <style>。

✧ <template>：这部分定义了 Vue 组件的模板，用于描述组件的结构和布局。它使用类似 HTML 的语法，并且可以包含 Vue 特定的指令、插值表达式和事件绑定

等。

- <script>：这部分包含了 Vue 组件的 JavaScript 代码。它定义了组件的行为和数据逻辑，并导出一个 Vue 实例或一个组件对象。
- <style>：这部分用于定义组件的样式。可以使用普通的 CSS 或预处理器（如 Sass 或 Less）来编写样式规则。例如：

2. 前端与后端交互

前端设计 reqAPI 函数，

```
XMLHttpRequest
export function reqAPI(type, addr, data) {
    if(type != 'POST') type = 'GET';
    const xhr = new XMLHttpRequest();
    xhr.open(type, 'http://localhost:5555' + addr, false);
    xhr.send(JSON.stringify(data));
    if (xhr.status === 200) {
      const data = JSON.parse(xhr.responseText);
      return data;
    } else {
      console.log('Error:', xhr.status);
    }
}
```

以根据不同的 API 来发送不同的 HTTP 请求。

3. 后端逻辑

后端采用 FastAPI 设计，其代码逻辑如下：

- 路由和请求处理：可以定义不同的路由来处理不同的 HTTP 请求方法和路径。通过使用装饰器，可以将路由函数与特定的请求方法（如 GET、POST、PUT、DELETE 等）和 URL 路径进行关联。当客户端发起请求时，FastAPI 将匹配对应的路由，并执行相应的处理函数。
- 数据验证和输入处理：FastAPI 提供了强大的数据验证功能，可以自动解析请求体中的 JSON 数据、查询参数、路径参数和表单数据，并根据预定义的模型和类型进行验证。可以使用 Pydantic 模型来定义请求和响应数据的结构，并在路由函数中进行类型注解，FastAPI 将自动处理数据的验证和转换。
- 业务逻辑处理：在路由函数中，可以编写与业务逻辑相关的代码，处理请求并执行特定的操作。这可能涉及从数据库中检索数据、处理计算、调用外部服务等各种操作。FastAPI 并没有限制业务逻辑代码的结构或组织方式，可以根据需求自由组织

代码。

- ✧ 错误处理和异常处理：FastAPI 提供了对错误和异常的处理机制。您可以使用 HTTPException 类或自定义异常类来抛出异常，并在路由函数中使用异常处理器来捕获和处理这些异常。可以返回适当的 HTTP 响应以及相应的错误信息，以提供友好的错误处理和响应。
- ✧ 响应处理：路由函数可以返回各种类型的响应，包括 JSON 数据、HTML 页面、文件下载等。FastAPI 会自动根据返回值的类型进行适当的转换，并设置正确的 HTTP 头部和状态码。

```python
@app.post("/battery/addbattery")
async def add_battery(request: Request):
    data = await request.json()
    bt = battery(size=data['size'], powerLeft=data['powerLeft'
    db.add(bt)
    db.commit()
    db.refresh(bt)
    return {'done': True}

@app.post("/battery/editbattery/{id}")
async def edit_battery(id:int, request: Request):
    data = await request.json()
    bt = db.query(battery).filter(battery.id == id).first()
    bt.size = data['size']
    bt.powerLeft = data['powerLeft']
    bt.status = data['status']
    bt.masterSlotId = data['masterSlotId']
    curECycleId=None
    db.commit()
    db.refresh(bt)
    return {'done': True}
```

4. 后端与数据库交互

使用 Pandanic 定义数据原型，例如：

```python
class battery(Base):
    __tablename__ = 'battery'
    id = Column(Integer, primary_key=True, nullable=False)
    size = Column(VARCHAR(20))
    powerLeft = Column(Integer, nullable=False)
    status = Column(VARCHAR(20))
    masterSlotId = Column(Integer, ForeignKey('slot.id'))
    curECycleId = Column(Integer, ForeignKey('ecycle.id'))

class cabinet(Base):
    __tablename__ = 'cabinet'
    id = Column(Integer, primary_key=True, nullable=False)
    name = Column(VARCHAR(255), nullable=False)
    pos = Column(VARCHAR(255), nullable=False)
    type = Column(VARCHAR(255), nullable=False)
    maxSlotNum = Column(Integer, nullable=False)
    companyId = Column(Integer, ForeignKey('company.id'), nullable=False)
    locx = Column(VARCHAR(255), nullable=False)
    locy = Column(VARCHAR(255), nullable=False)
```

然后创建 DB 引擎链接到 Mysql 服务器上，就可以进行增删改查：

```python
engine = create_engine('mysql+mysqlconnector://root:@localhost:3306/ecbattery',echo=True)
DBSession = sessionmaker(bind=engine)
db = DBSession()
```

例如添加信息：

```python
@app.post("/cabinet/addcabinet")
async def add_cabinet(request: Request):
    data = await request.json()
    cb = cabinet(name=data['name'] ,pos=data['p
    db.add(cb)
    db.commit()
    db.refresh(cb)
    return {'done': True}
```

7.3 具体功能编写

1. 登陆注册页面：

前端界面绘制如下：

实现功能：登陆注册。在该界面注册的均为普通用户，管理者用户只能通过后台设置 admin 账号进行注册。

后端共设计了 6 个 API，具体设计的 API 如下：

2. 主页页面：

前端界面绘制如下：

实现功能：各种可视化数据的显示。采用 echarts[5]编写和开发，通过后端 API 调取分布和插槽情况数据，然后将其绘制成 echarts 图表。

没有额外设计 API，而是采用各个数据库查询的 API 进行显示。

3. 信息总览页面：

前端界面绘制如下：

实现功能：地图可视化点位显示。将可用电瓶和点位情况实时展示在地图上，并且允许用户通过点击跳转到相应的电柜，从而为用户提供便捷直观的换电点位查询。

开发采用百度地图开放平台[6]进行开发，通过在地图上添加地图构筑物图层来实现额外信息的可视化显示。

没有设计额外 API，而是直接采用电柜的相应查询 API，具体在后文给出。

4. 取电页面：

前端界面绘制如下：

客户显示的界面：

管理员显示的界面：

取电页面是核心页面之一，这里考虑到两种用户的需求，因此分别为管理员和客户设计打造了两个不同的页面，从而支持两个用户的不同功能。对客户而言只能进行电瓶的查询和查找，因此核心功能 1 为取用，2 为定位；而对于管理员而言，其不需要对电瓶进行取用和定位操作，取而代之的是添加、编辑和删除功能。

后端共设计了 9 个 API，其具体设计如下：

电瓶（接口 9 个）

输入关键

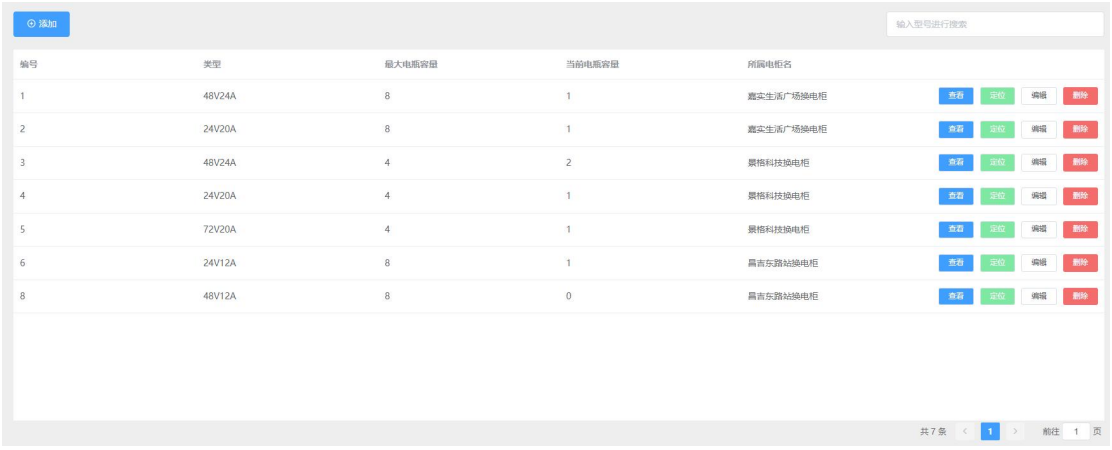
接口名称	请求类型	接口路径	接口分组	接口状态	标签
获取所有电瓶	GET	/battery/getallbattery	电瓶	开发中	-
添加电瓶	POST	/battery/addbattery	电瓶	开发中	-
修改电瓶信息	POST	/battery/editbattery/{id}	电瓶	开发中	-
查看电瓶信息	GET	/battery/battery/{id}	电瓶	开发中	-
删除电瓶	POST	/battery/deletebattery/...	电瓶	开发中	-
link电瓶	POST	/battery/link,{id},{cyclei...	电瓶	开发中	-
unlink电瓶	POST	/battery/unlink,{id},{ma...	电瓶	开发中	-
获取插槽电瓶	GET	/battery/getslotbattery...	电瓶	开发中	-
获取我的电瓶	GET	/getallmybattery/{id}	电瓶	开发中	-

5. 还电页面：

前端界面绘制如下：

客户显示的界面：

管理员显示的界面：



还电页面也是核心页面之一，这里考虑到两种用户的需求，因此分别为管理员和客户设计打造了两个不同的页面，从而支持两个用户的不同功能。对客户而言有归还、定位两个与上面类似的功能，除此以外还支持用户查询相应插槽的整体情况；而对于管理员而言依旧支持添加、编辑和删除功能。

后端共设计了 6 个 API，具体 API 设计如下：

插槽（接口 6 个）

接口名称	请求类型	接口路径	接口分组	接口状态	标签
获取插槽	GET	/slot/slot/{id}	插槽	开发中	-
获取所有插槽	GET	/slot/getallslot	插槽	开发中	-
添加插槽	POST	/slot/addslot	插槽	开发中	-
编辑插槽	POST	/slot/editslot/{id}	插槽	开发中	-
删除插槽	POST	/slot/deleteslot/{id}	插槽	开发中	-
获取电柜的所有插槽	GET	/slot/getcabinetslot/{id}	插槽	开发中	-

6. 车辆管理

前端界面绘制如下：

客户显示的界面：

管理员显示的界面：

实现功能：车辆管理是用户所使用的车辆的实体化，电瓶的取用操作也必须绑定到相应的车辆上才可以执行，这里用户和管理员的区别是用户只能添加和维护属于自己的车辆，而管理员可以做到管理和维护所有人的车辆。

后端共设计了 5 个 API，具体 API 设计如下：

电车（接口 5 个）

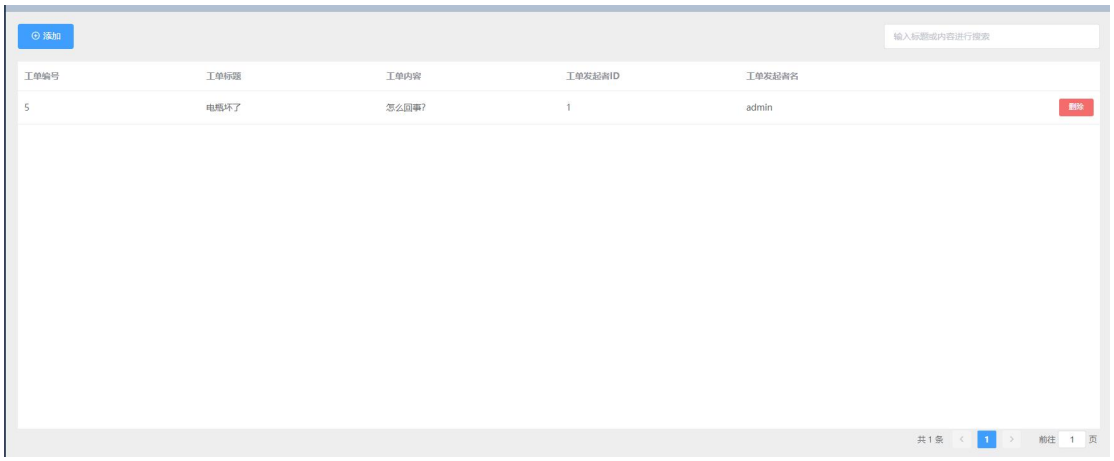
接口名称	请求类型	接口路径	接口分组	接口状态	标签
获取用户电车信息	GET	/ecycle/{user}	电车	开发中	-
添加电车	POST	/addECycle	电车	开发中	-
编辑电车	POST	/editECycle/{id}	电车	开发中	-
删除电车	POST	/deleteECycle/{id}	电车	开发中	-
获取指定电车	GET	/getecycle/{id}	电车	开发中	-

7. 工单管理

前端界面绘制如下：

客户显示的界面：

管理员显示的界面：



实现功能：工单的发起和管理。实际运行的系统会发生各种各样不同的故障，用户在遇到故障时可以通过点击上方的添加按钮发起故障工单，发起后管理员就可以在后台看到用户发起的工单并进行处理。

共实现了 3 个 API，具体 API 设计如下：

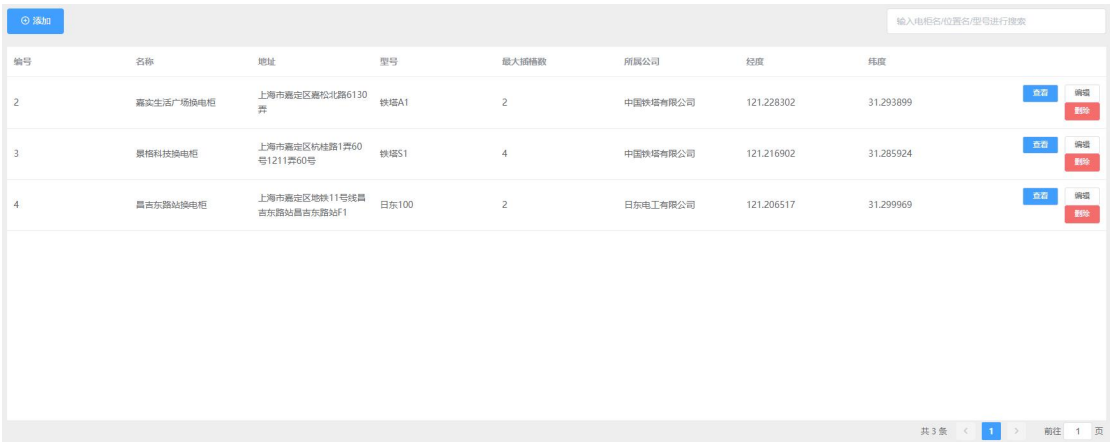
工单（接口 3 个）

接口名称	请求类型	接口路径	接口分组	接口状态
添加工单	POST	/workorder/addworkor...	工单	开发中
获取所有工单	GET	/workorder/getallwork...	工单	开发中
删除工单	POST	/workorder/deletewor...	工单	开发中

8. 电柜管理

前端界面绘制如下：

管理员显示的界面：



实现功能：管理员对电柜的管理。管理员在与公司协商通过后，可以将其所属的电柜添加到系统中并分配相应的插槽，同时也可以修改或删除发生相应情况的电柜，从而实现对电柜资源的可持续维护。

共实现了 5 个 API，具体 API 设计如下：

电柜（接口 5 个）

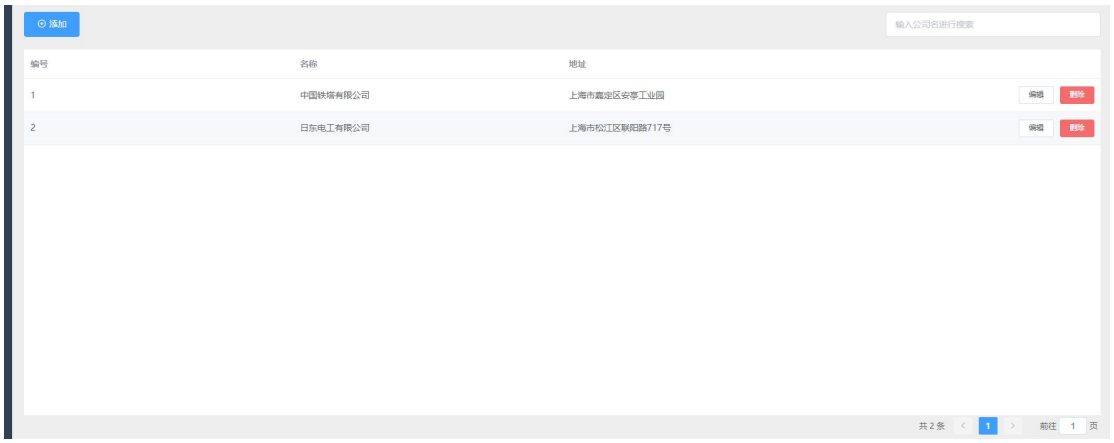
输入关键字

接口名称	请求类型	接口路径	接口分组	接口状态	标签
获取电柜详情	GET	/cabinet/cabinet/{id}	电柜	开发中	-
获取所有电柜	GET	/cabinet/getallcabinet	电柜	开发中	-
添加电柜	POST	/cabinet/addcabinet	电柜	开发中	-
修改电柜	POST	/cabinet/editcabinet/{id}	电柜	开发中	-
删除电柜	POST	/cabinet/deletecabinet...	电柜	开发中	-

9. 公司管理

前端界面绘制如下：

管理员显示的界面：



实现功能：公司的注册和增删改查。电柜必须与公司进行绑定，不可以出现没有从属公司的电柜。

共实现了 3 个 API，具体 API 设计如下：

公司（接口 5 个）

接口名称	请求类型	接口路径	接口分组	接口状态	标签
获取公司	GET	/company/company/{id}	公司	开发中	-
获取所有公司	GET	/company/getallcomp...	公司	开发中	-
增加公司	POST	/company/addcompany	公司	开发中	-
修改公司	POST	/company/editcompan...	公司	开发中	-
删除公司	POST	/company/deletecomp...	公司	开发中	-

7.4 功能测试

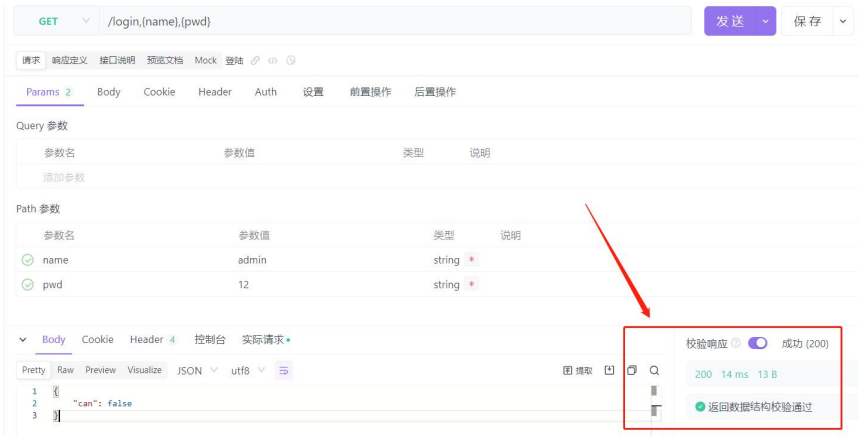
整体测试设计包含前端测试和后端测试。

首先介绍后端测试，采用 APIFOX 的 MOCK 测试进行，具体过程如下：

1. 在对应的 API 界面，设计请求模式和响应定义

2. 编写并添加测试数据，点击发送进行测试

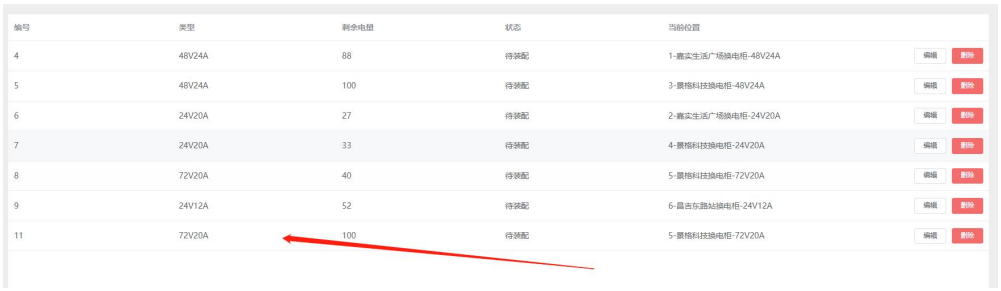
成功样例：可以看到返回成功（200），并显示数据校验结果通过，如果失败则会显示相应的报错码（比如 404、500 等）。



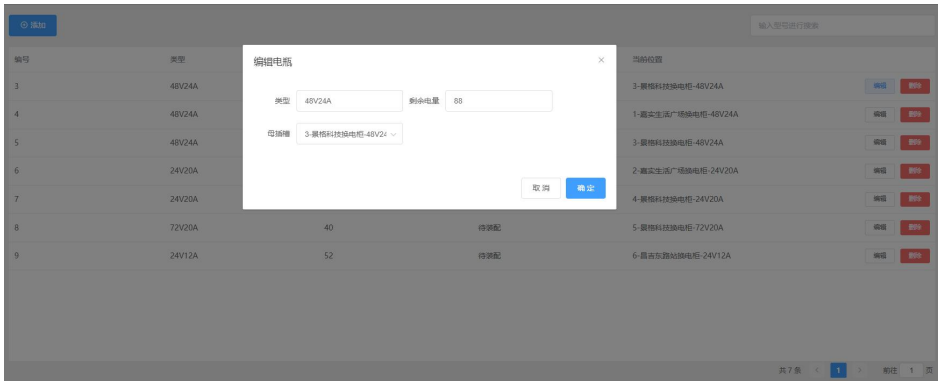
本次数据库课设编写的 37 个测试均通过了 Mock 测试，具体测试可见 APIFOX 分享链接[7]。

前端采用单元测试的方法，对各个功能采用手动交互的方式进行基本路径测试，例如电瓶管理：

添加：



编辑：



删除：

等等。11 个不同页面的模块都经过了基本路径测试，因为测试过程同质化且截图非常类似，这里就不再全部具体展示了，只是放了电瓶管理页面测试的流程和结果。

此外，还对数据库的后端 API 进行了压力测试，测试参数为每秒 10000 请求，整体考核本

次数据库课设的性能和鲁棒性，压力测试结果如下：

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	5816	658	1535	181	16808	1099	40.4	4.6
POST	/login	17571	4470	29967	527	139904	706	122.0	31.0
POST	/register	11202	1306	2499	359	18443	836	77.7	9.1
	Aggregated	**34589**	**6434**	**16291**	**181**	**139904**	**814**	**240.1**	**44.7**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/	1100	1200	1500	1800	2400	3000	15000	17000
POST	/login	3300	9400	32000	79000	98000	119000	134000	140000
POST	/register	2100	2400	2700	3100	3800	4600	16000	18000
	Aggregated	**2300**	**2700**	**3400**	**12000**	**80000**	**98000**	**130000**	**140000**

整体 Failure 在 0.4%左右，压力测试下是合格的。

八.总结

8.1 项目心得

在本次数据库课程设计中，我将课堂上的知识点进行了集中运用，不仅设计了一个包含七个表的数据库，用课堂上学习的 Mysql 命令进行建库和增改，同时也系统性地学习了目前比较流行的前后端设计的方法，比如用 ECharts 生成可视化图表，用百度地图开放平台实现了可视化地图标注，用 fastAPI 进行后端 API 设计等等，这里的每一个技术栈都是第一次学习、第一次运用，在完成课程设计之后再回头看我其实十分惊讶：一是惊讶于我第一次学习也可以编写出私以为非常优秀的项目（笑），二也是惊讶于这些主流的开发工具和开发语言竟然可以带来如此巨大的效率提升，比我在大二下写 Web 时使用的 Html+CSS+flask 的技术栈效率要高出好几倍。技术栈的更新确实是很重要的一件事。

相较于其他课程设计，本次其实更注重于数据库的设计、访问性能和稳定性，这是目前来看数据库非常重要的一件事，无论前端多么美观、功能多么繁杂，最主要的最核心的依旧是数据库性能和数据库鲁棒性。12306 订票系统花费上千万人民币打造而成，寄托了非常多的期望，可是在上线的几年每逢春节就会崩溃、卡死，用户体验十分糟糕，从这个例子中就

可以看出数据库性能和数据库鲁棒性的强弱对整个系统的重要含义了。这次在进行完数据库开发后，特地对数据库进行了压力测试，结果也是符合期望的。

大数据时代下，展望前方，本次设计的系统其实可以更进一步为管理员提供人工智能下的数据分析服务，当收集到的用户数据足够多，通过人工智能、机器学习的分析是可以得出不同时段下用户取用/归还电瓶的趋势，从而实现可预测的电瓶迁移服务，从存量高用量少的地方的电瓶迁移到存量少用量多的地方，进而实现可用性、效率和吞吐量的提升。我也期待着未来真的能有公司实现这一功能，让电瓶做到 7x24 小时的可用性，也算是我对本次课设的一些小小的幻想。

参考文献

[1] 国务院关于上海市城市规划的总体回复 国函〔2017〕147 号 [EB/OL].上海市规划和自然资源局, https://ghzyj.sh.gov.cn/ghjh/20200110/0032-811864.html, 2017

[2] 国务院关于上海市城市规划的总体回复 国函〔2017〕147 号 [EB/OL].上海市规划和自然资源局, https://ghzyj.sh.gov.cn/ghjh/20200110/0032-811864.html, 2017

[3] 2023 电动两轮车行业现状与市场发展前景趋势分析[EB/OL].中研网, https://www.chinairn.com/news/20230324/154633513.shtml , 2023

[4] 强化电动自行车消防安全管理，上海推进安装 30 万个充换电设施[EB/OL]. 新闻晨报, https://mp.weixin.qq.com/s?__biz=MzAxOTY2Njg3Mg , 2022-11

[5] Apache ECharts：一个基于 JavaScript 的开源可视化图表库 [EB/OL]. Apache organization https://echarts.apache.org/zh/index.html.

[6] 北斗百度地图开放平台快速入门指南.[OL] https://lbsyun.baidu.com/

[7] EC-Battery 项目：后端 37 个 API 测试结果.[OL] APIFOX. https://apifox.com/apidoc/shared-97acf471-790f-42dd-b716-ed160b280de1