

上海市体育场馆管理系统

大数据管理系统课程设计报告



学 号 2050267

姓 名 秦天

专 业 数据科学与大数据技术

授课老师 李文根

目录

1.	概述.....	4
1.1	课题背景.....	4
1.2	编写目的.....	4
2.	需求分析.....	6
2.1	功能需求.....	6
2.2	数据字典.....	7
2.3	数据流图.....	9
3.	可行性分析.....	9
3.1	技术可行性.....	9
3.2	应用可行性.....	10
4.	概念设计.....	12
4.1	实体描述.....	12
4.2	实体属性局部 E-R 图.....	14
4.3	全局 E-R 图.....	16
5.	逻辑设计.....	17
5.1	E-R 图向关系模型的转变.....	17
5.2	数据模型的优化及规范化设计.....	18
6.	项目管理.....	21
6.1	框架选择.....	21
6.2	开发平台.....	21
7.	系统实现.....	23
7.1	系统架构搭建.....	23
7.2	系统逻辑设计.....	24
7.3	具体功能编写.....	25
7.4	功能测试.....	37
8.	总结.....	38

摘要：本课程设计报告旨在设计并实现一个名为上海市体育场馆管理系统的大数据管理系统。该系统基于上海市公共数据开放平台提供的数据，并分为管理员和用户两个角色。用户可以通过系统预约各种场馆设施，而管理员则负责管理系统的运行和维护。

该系统采用 **MySQL** 作为数据库管理系统，利用其可靠性和高效性，存储和管理大量的场馆、设施和用户信息。同时，为了实现系统的前后端分离和良好的用户体验，我们使用 **Flask** 作为后端框架，**Vue** 作为前端框架，通过 **API** 进行数据交互和展示。

在设计和开发过程中，我们着重考虑以下几个方面：首先，系统应具备用户友好的界面和良好的交互性，使用户能够轻松浏览和预约各种场馆设施；其次，系统应具备高效的数据管理能力，能够快速检索和更新各种信息，确保用户预约的准确性和及时性；最后，系统应具备良好的安全性和稳定性，保护用户隐私并确保系统的正常运行。

通过该系统的设计与实现，用户可以方便地查找和预约各类体育场馆设施，提高场馆资源的利用率，促进城市居民的体育健身活动。同时，管理员可以通过系统进行场馆资源的管理和调配，提升运营效率和服务质量。

关键词：数据库；上海市体育场馆；MySQL；Flask；管理系统

1. 概述

1.1 课题背景

随着城市化进程的加速和人们对健康生活的追求，体育运动在人们的日常生活中扮演着越来越重要的角色。作为中国的国际大都市，上海市拥有众多体育场馆和设施，为居民提供了广泛的运动选择和锻炼场所。然而，随之而来的问题是如何有效管理和利用这些场馆资源，以满足不断增长的需求。

传统的场馆预约和管理方式存在一些问题。首先，人工管理往往效率低下，容易出现预约冲突和信息不准确的情况。其次，缺乏统一的信息平台和系统化的管理手段，使得场馆资源的分配和调配难以有效进行。此外，对于用户而言，获取场馆信息和进行预约也存在一定的困难和不便。

为了解决这些问题，设计和开发一个上海市体育场馆管理系统具有重要意义。该系统将通过利用上海市公共数据开放平台提供的数据，实现对场馆、设施和用户信息的集中管理和统一展示。管理员可以通过系统进行场馆资源的调配和管理，提高资源的利用效率和服务质量。用户可以通过系统方便地查找和预约各类场馆设施，提升体育锻炼的便捷性和体验感。

此外，借助大数据管理系统的技术和手段，可以对场馆使用情况、用户需求等数据进行分析 and 挖掘，为场馆管理者提供决策支持和业务优化建议。通过系统的建立和运行，可以促进城市居民的体育健身活动，提高健康水平和生活质量。

因此，上海市体育场馆管理系统的设计与实现对于提升场馆资源的管理效率、满足居民体育需求、推动城市体育事业的发展具有重要的现实意义和应用前景。

1.2 编写目的

本课程设计旨在实现上海市体育场馆管理系统的设计与开发，旨在解决传统场馆管理方式存在的问题，并提供更高效、便捷的场馆预约和管理服务。具体编写目的如下：

1. 实现集中管理与统一展示：通过该系统，将上海市公共数据开放平台提供的场馆、设施和用户信息进行集中管理和统一展示，使管理员能够方便地了解和掌握场馆资源的使用情况，优化资源调配和管理。
2. 提升用户体验和便捷性：系统提供用户友好的界面和良好的交互性，使用户能够方便地浏览、搜索和预约各类场馆设施，提高用户的预约体验和便捷性。
3. 提高预约准确性和及时性：通过系统的预约管理功能，用户可以准确选择和预约自己所需的场馆设施，并及时获取预约结果和通知，避免预约冲突和信息不准确的问题。
4. 数据分析与决策支持：利用大数据管理系统的技术和手段，对场馆使用情况、用户需求等数据进行分析和挖掘，为场馆管理者提供决策支持和业务优化建议，促进场馆资源的合理配置和运营管理。
5. 推动城市体育事业发展：通过该系统的建立与运行，促进城市居民的体育健身活动，提高体育设施的利用率，推动城市体育事业的发展，提升居民的健康水平和生活质量。

通过以上目标的实现，上海市体育场馆管理系统将为场馆管理者和用户提供高效的管理和预约服务，优化资源利用和满足体育需求，推动城市体育事业的发展。同时，通过大数据分析和决策支持，为管理者提供合理的决策依据，进一步提升运营效率和服务质量。

2. 需求分析

2.1 功能需求

主要分为两种级别的用户

管理员功能需求：

1. 用户管理：管理员能够管理用户信息，包括添加新用户、编辑用户信息、删除和查询用户等操作。管理员还可以查看用户的预约情况等。
2. 体育馆管理：管理员能够管理体育馆信息，包括添加新的体育馆、编辑体育馆信息、删除体育馆和查询等操作。管理员可以查看体育馆的基本信息和可用设施。
3. 体育设施管理：管理员能够管理每个体育馆的体育设施，包括设施的添加、编辑和删除。管理员可以设置设施的开放时间段和预约限制。
4. 管理预约：管理员可以查看用户的预约请求，审核和确认预约申请，或者拒绝预约的场馆设施。
5. 个人信息修改：管理员可以修改自己的个人信息，如用户名、密码等。

普通用户功能需求：

1. 预约设施：用户可以浏览可用的体育馆和设施，选择合适的设施进行预约。用户可以查看设施的开放时间和预约限制，选择适合自己的预约时间段。
2. 个人信息修改：用户可以修改自己的个人信息，包括用户名、密码、联系方式等。用户可以随时更新和管理个人信息。
3. 查看预约信息：用户可以查看自己已预约的体育馆设施信息，包括预约时间、预约状态等。
4. 浏览体育馆信息：用户可以浏览可用的体育馆信息，查看体育馆的基本信息、设施情况和开放时间等。

以上是上海市体育场馆管理系统的功能需求，管理员主要负责用户管理、体育馆管理、设施

管理、管理预约和个人信息修改等功能，而普通用户主要涉及预约设施、个人信息修改和浏览体育馆信息等功能。通过这些功能的支持，系统将提供便捷的管理和预约服务，满足管理员和用户的需求。

2.2 数据字典

数据字典是一个用于描述数据库中各个表、字段以及其属性和关系的文档或工具。它提供了关于数据库中存储的数据和数据结构的详细信息，包括数据类型、大小、约束、关系等。

数据字典通常包含以下信息：

- 1. 表名和描述：列出数据库中的表以及每个表的简要描述。
- 2. 字段数据类型：指定每个字段的数据类型，如整数、字符串、日期等。
- 3. 字段长度和约束：指定每个字段允许的最大长度以及其他约束，例如唯一性约束、非空约束等。
- 4. 主键和外键关系：指定表中的主键和外键，以及它们之间的关系。
- 5. 默认值：指定字段的默认值，如果没有显式提供值，则使用默认值。

数据字典的主要目的是提供一个标准化的文档或工具，以便数据库管理员、开发人员和其他相关人员可以清楚地了解数据库的结构和数据元素的含义。它有助于维护数据的一致性、准确性和完整性，并为数据库设计、开发、维护和查询提供参考和指导。同时，数据字典也是数据管理、数据分析和数据治理的重要组成部分。以下是本项目的数据字典：

gym 表

列名	数据类型	约束
gym_id	INT	主键，自增
gym_district	VARCHAR(50)	默认空字符串
gym_name	VARCHAR(255)	非空
gym_address	VARCHAR(255)	非空

facility 表

列名	数据类型	约束
facility_id	INT	主键，自增
facility_name	VARCHAR(255)	非空
gym_id	INT	外键，参考 gym 表的 gym_id

user 表

列名	数据类型	约束
user_id	VARCHAR(255)	主键
user_name	VARCHAR(255)	非空
user_email	VARCHAR(255)	非空， CHECK (email REGEXP '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\$')
user_phone	VARCHAR(11)	非空， CHECK (LENGTH(user_phone) = 11)
user_password	VARCHAR(255)	非空， CHECK (LENGTH(user_password) >= 8)
user_type	ENUM	默认'user'

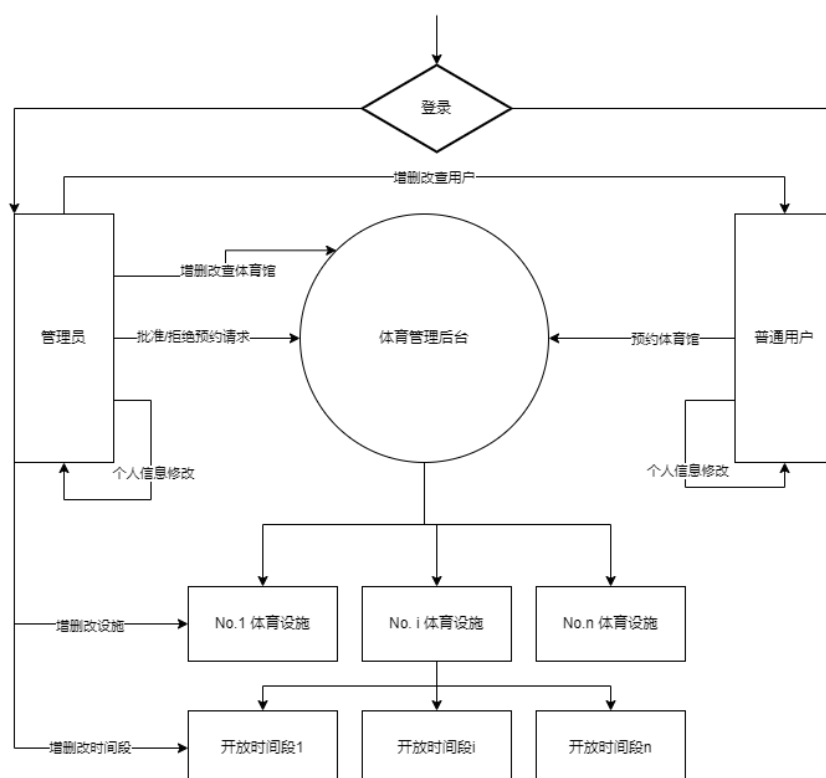
reservation 表

列名	数据类型	约束
reservation_id	INT	主键，自增
user_id	VARCHAR(255)	外键，参考 user 表的 user_id
facility_id	INT	外键，参考 facility 表的 facility_id
schedule_date	DATE	非空
schedule_time	VARCHAR(255)	非空
reservation_date	DATE	非空
status	INT	非空，默认为 1

time_slot 表

列名	数据类型	约束
time_id	INT	主键，自增
date	DATE	非空
start_time	TIME	非空
end_time	TIME	非空
facility_id	INT	外键，参考 facility 表的 facility_id

2.3 数据流图



3. 可行性分析

3.1 技术可行性

实现上海市体育场馆管理系统是技术上可行的，主要基于以下几点：

1. 数据库管理系统：使用 MySQL 数据库进行数据的存储和管理。MySQL 是一种成熟、可靠的关系型数据库管理系统，适用于存储和处理大量结构化数据。
2. 后端开发框架：使用 Flask 框架进行后端开发。Flask 是一个轻量级的 Python Web 框架，易

于学习和使用，具备良好的扩展性和灵活性，适合构建中小型的 Web 应用程序。

3. 前端开发框架：使用 Vue.js 进行前端开发。Vue.js 是一种流行的 JavaScript 框架，具备响应式设计和组件化开发的优势，可提供良好的用户界面和用户体验。
4. 数据集成：通过上海市公共数据开放平台获取相关的场馆、设施和过去开放时间信息。
5. 用户认证和权限管理：实现用户认证和权限管理功能，确保只有授权的管理员才能进行管理操作，而普通用户只能进行预约和个人信息修改。

综上所述，基于 MySQL 数据库、Flask 框架和 Vue.js 框架等技术，结合上海市公共数据开放平台提供的接口，实现上海市体育场馆管理系统是技术上可行的。然而，具体的实现和技术选择还需要根据项目需求、团队能力和资源预算进行进一步评估和决策。

3.2 应用可行性

实现上海市体育场馆管理系统的应用可行性主要考虑以下几个方面：

1. 需求可行性：上海市体育场馆管理系统的功能需求符合实际场景和用户需求。通过预约各种场馆设施、用户管理、个人信息修改等功能，系统可以提供便捷的体育场馆预约和管理服务。
2. 市场需求：上海是中国的一线大城市，拥有众多的体育场馆和运动爱好者。针对广大市民和游客的体育场馆需求，提供一个在线预约和管理系统，有助于提高场馆利用率、优化资源配置，并提供更好的用户体验。
3. 技术支持：相关的技术框架和工具（如 MySQL、Flask、Vue.js）在开发社区中得到广泛应用和支持，提供了丰富的文档、示例和插件。这些技术的成熟度和社区支持为系统的开发和维护提供了可靠的基础。
4. 可行性评估：进行项目的可行性评估，包括技术可行性、操作可行性等方面的考虑。评估项目所需的开发能力、系统的可操作性以及可维护性。

综上所述，基于需求可行性、市场需求、数据可获得性、技术支持和可行性评估等方面的考虑，实现上海市体育场馆管理系统在应用上是可行的。然而，还需要综合考虑项目的实施成本、

市场竞争、推广策略等因素，以确定系统的商业可行性和长期可持续发展。

4. 概念设计

4.1 实体描述

以下是给出的数据字典中每个表的实体描述：

gym 表

实体名称: 体育馆 (Gym)

属性:

- gym_id: 体育馆 ID
- gym_district: 体育馆所属区域或地区
- gym_name: 体育馆名称
- gym_address: 体育馆地址

facility 表

实体名称: 设施 (Facility)

属性:

- facility_id: 设施 ID
- facility_name: 设施名称
- gym_id: 所属体育馆 ID

user 表

实体名称: 用户 (User)

属性:

- user_id: 用户 ID
- user_name: 用户姓名

- user_email: 用户邮箱
- user_phone: 用户电话
- user_password: 用户密码
- user_type: 用户类型（管理员或普通用户）

reservation 表

实体名称: 预约 (Reservation)

属性:

- reservation_id: 预约 ID
- user_id: 预约用户 ID
- facility_id: 预约设施 ID
- schedule_date: 预约日期
- schedule_time: 预约时间
- reservation_date: 预约创建日期
- status: 预约状态

time_slot 表

实体名称: 时间段 (Time Slot)

属性:

- time_id: 时间段 ID
- date: 日期
- start_time: 开始时间
- end_time: 结束时间
- facility_id: 设施 ID

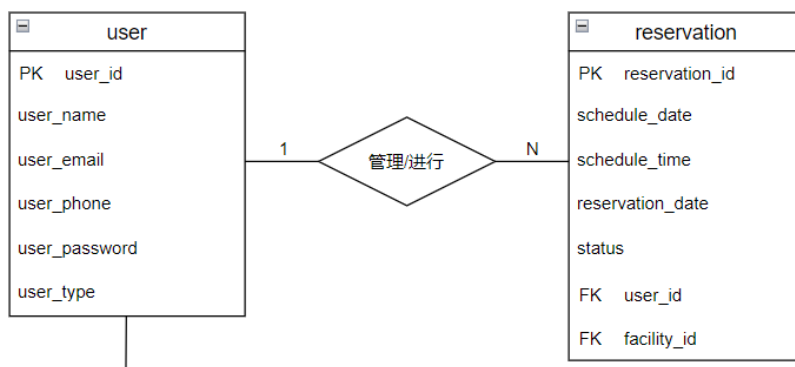
4.2 实体属性局部 E-R 图

实体属性局部 E-R 图（Partial Entity-Relationship Diagram）是在 E-R 图中的一个局部部分，用于表示实体（Entity）以及它们之间的属性和关系。它展示了一组相关实体及其属性，但不涵盖整个数据库的结构。以下是实体属性局部 E-R 图：

用户管理/进行预约

关系描述：用户分为两种（管理员/普通用户），前者对预约信息进行管理，后者进行预约。

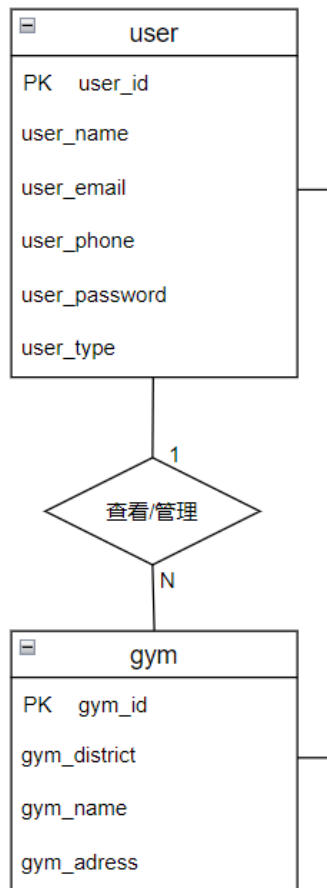
映射基数：一位用户可以管理/创建多条预约，但一条预约仅能由唯一的用户创建，因此映射是一对多的。



用户管理/查看体育馆

关系描述：用户分为两种（管理员/普通用户），前者对体育馆进行管理，后者进行查看。

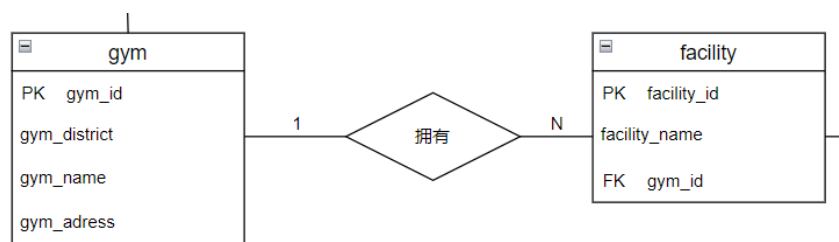
映射基数：一位用户可以管理多个体育馆，但一个体育馆仅能由唯一的用户管理，因此映射是一对多的。



体育馆拥有体育设施

关系描述：体育馆中可以存在有多个体育设施。

映射基数：一个体育馆可以拥有多个体育设施，但一个体育设施仅能隶属于唯一的体育馆，因此映射是一对多的。

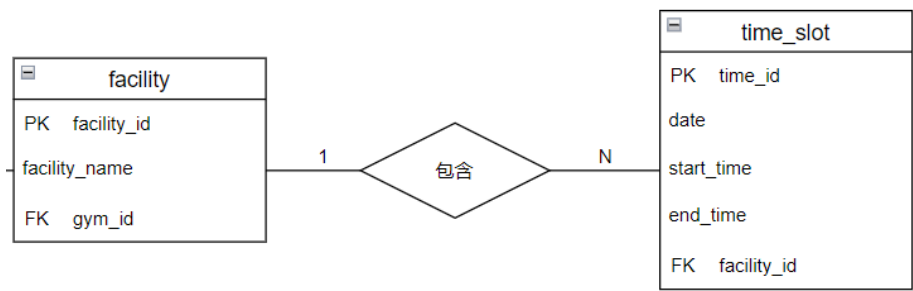


体育设施包含开放时间

关系描述：每个体育设施有多个不同的开放时间。

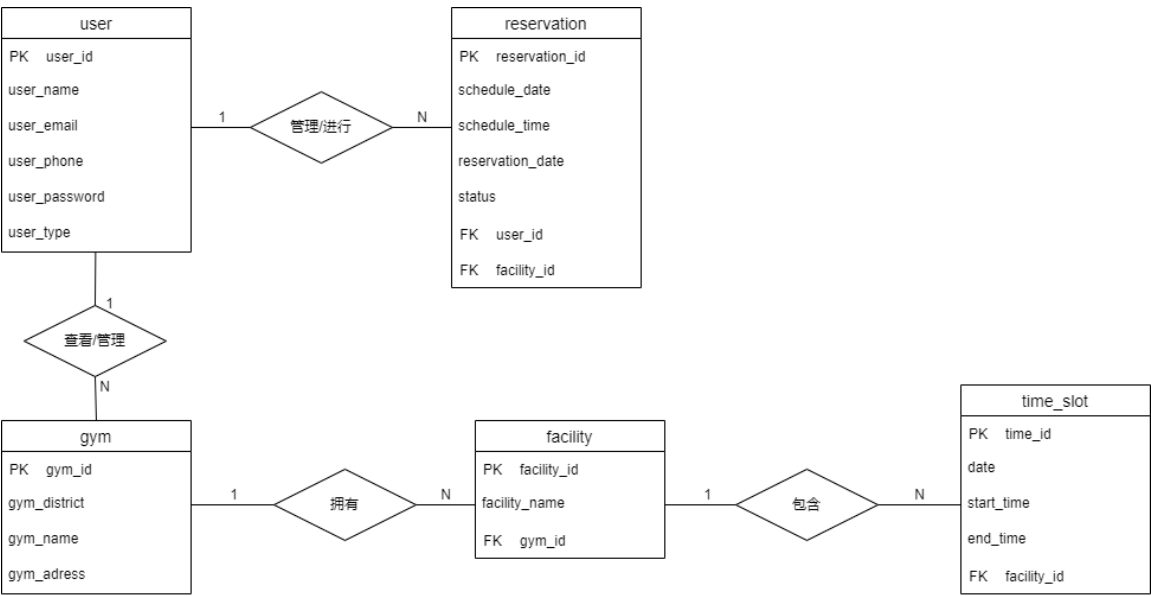
映射基数：每个体育设施可以包含有多个不同的开放时间，但一个开放时间段仅能属于唯一的

体育馆，因此映射是一对多的。



4.3 全局 E-R 图

全局 E-R 图（Global Entity-Relationship Diagram）是数据库设计中的一个工具，用于表示整个数据库的结构和组成部分。它展示了数据库中所有实体（Entities）、属性（Attributes）以及它们之间的关系（Relationships）。



5. 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，需要将 E-R 图按照一定规则转化为相应的关系模式并合并冗余，得到对应表单。在将 E-R 模型转化为关系模式时，主要遵循如下规则：

① 检查所有强实体集，为其建立对应的关系模式，对应关系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。

在设计 E-R 模型中，所有实体集均为强实体集，因此可以构造如下关系模式：

Gym (体育馆)

`gym(gym_id, gym_district, gym_name, gym_address)`

Facility (设施)

`facility(facility_id, facility_name, gym_id)`

User (用户)

`user(user_id, user_name, user_email, user_phone, user_password, user_type)`

Reservation (预约)

`reservation(reservation_id, user_id, facility_id, schedule_date, schedule_time, reservation_date, status)`

Time_slot (时间段)

`time_slot(time_id, date, start_time, end_time, facility_id)`

检查所有联系集，管理、包含、拥有等联系均为一对多的，它们所需添加的属性均已在逻辑

辑设计中添加到实体集中，不需进行额外操作。

5.2 数据模型的优化及规范化设计

将 E-R 模型转化为关系模式后，还需对得到的关系模式进行规范化，使其满足 3NF 范式。

3NF（第三范式）是数据库设计中的一种范式，用于消除数据表中的传递依赖性。它是在满足第二范式（2NF）的基础上进一步分解关系模式，以达到数据的高度规范化和减少数据冗余的目的。

以下是对 3NF 范式的解释：

第一范式（1NF）要求每个数据表中的每个属性都是原子的，即属性不可再分。这确保了数据表中的每个单元格只包含单一的数据值。

第二范式（2NF）要求在满足 1NF 的基础上，非主键属性必须完全依赖于候选键（主键）而不是部分依赖。换句话说，一个表中的每个非主键属性都必须完全依赖于表的主键。

第三范式（3NF）要求在满足 2NF 的基础上，消除非主键属性之间的传递依赖性。具体来说，如果一个表中存在非主键属性之间的传递依赖，那么应该将这些属性分离到新的表中，使每个表只包含相互独立的数据。

通过将关系模式进一步规范化为 3NF，可以实现以下优点：

数据冗余减少：通过消除传递依赖性，减少了数据表中的冗余数据，提高了数据存储的效率。

数据一致性：将数据分解为更小的表，可以更好地保持数据的一致性和完整性。

查询优化：规范化的数据模型可以更好地支持复杂的查询操作，提高数据库的查询性能。

需要注意的是，过度规范化也可能导致表之间的连接操作复杂化和查询效率下降，因此在设计数据库时，需要权衡规范化的程度和查询的性能要求。

根据先前给出的几个表的结构，现在进行检查以确定是否满足第三范式（3NF）的要求：

表：gym(体育馆)

1. gym_id(主键)

2. gym_district

3. gym_name

4. gym_address

该表满足 3NF 范式，因为每个属性都直接依赖于主键（gym_id），不存在传递依赖关系。

表：facility (设施)

1. facility_id (主键)

2. facility_name

3. gym_id (外键，参考 gym 表的主键 gym_id)

该表满足 3NF 范式，因为每个属性都直接依赖于主键（facility_id），不存在传递依赖关系。

表：user (用户)

1. user_id (主键)

2. user_name

3. user_email

4. user_phone

5. user_password

6. user_type

该表满足 3NF 范式，因为每个属性都直接依赖于主键（user_id），不存在传递依赖关系。

表：reservation (预约)

1. reservation_id (主键)

2. user_id (外键，参考 user 表的主键 user_id)

3. facility_id (外键，参考 facility 表的主键 facility_id)

4. schedule_date

5. schedule_time

6. reservation_date

7. status

该表满足 3NF 范式，因为每个属性都直接依赖于主键（reservation_id），不存在传递依赖关系。

表：time_slot (时间段)

1. time_id (主键)

2. date

3. start_time

4. end_time

5. facility_id (外键，参考 facility 表的主键 facility_id)

该表满足 3NF 范式，因为每个属性都直接依赖于主键（time_id），不存在传递依赖关系。

根据检查，先前给出的几个表的结构都满足第三范式（3NF）的要求。每个表中的属性都直接依赖于其主键，不存在传递依赖关系，因此表结构在数据规范化方面是合理和有效的。

6. 项目管理

6.1 框架选择

本项目的框架如下：

1. 后端框架：Flask

- Flask 是一个轻量级的 Python 后端框架，适合快速构建 Web 应用程序。
- 它提供了简单而灵活的方式来处理路由、请求处理、数据库操作等后端功能。

2. 前端框架：Vue.js

- Vue.js 是一个流行的 JavaScript 前端框架，用于构建交互性强的用户界面。
- 它提供了响应式数据绑定、组件化开发等功能，使前端开发更加高效和可维护。

3. 数据库：MySQL

- MySQL 是一种常见的关系型数据库管理系统，适用于处理结构化数据。
- 它提供了可靠的数据存储和高性能的查询功能，适合于您的体育场馆管理系统。

这个框架充分利用了 Flask 作为后端框架和 Vue.js 作为前端框架的优势。可以使用 Flask 处理后端逻辑和 API 的开发，包括用户管理、体育馆管理、预约管理等。同时，Vue.js 可以用于构建动态、交互性的用户界面，与后端 API 进行数据交互和展示。

对于数据库，您可以使用 MySQL 作为数据存储和管理系统。通过 Flask 提供的数据库集成，您可以轻松地进行数据库操作，包括数据的存储、查询和更新等。

6.2 开发平台

在上海市体育场馆管理系统的开发中，使用 Visual Studio Code (VS Code)作为集成开发环境：

1. 跨平台支持：VS Code 可以在多个操作系统上运行，包括 Windows、macOS 和 Linux。

这意味着团队成员可以在不同的操作系统下使用相同的开发环境，方便协同开发和统

一的代码风格。

2. 强大的代码编辑功能: VS Code 具有智能代码补全、语法高亮、自动格式化等功能, 可以提高开发效率和代码质量。它支持多种编程语言, 包括 Python 和 JavaScript, 适用于本项目的后端和前端开发。
3. 内置终端: VS Code 集成了终端功能, 可以在编辑器中直接执行命令行指令。这对于运行项目、执行数据库操作或运行测试等任务非常方便, 无需离开编辑器界面。
4. 丰富的插件生态系统: VS Code 拥有庞大的插件市场, 可以根据项目需求选择和安装合适的插件。对于上海市体育场馆管理系统的开发, 您可以使用各种插件来增强编辑器功能, 例如 Python 插件、Vue.js 插件、Git 插件等。
5. 轻量级和快速启动: VS Code 是一个轻量级的编辑器, 启动速度快, 不会占用太多系统资源。这使得您可以快速启动编辑器并开始开发工作, 提高开发效率。

综上所述, 使用 VS Code 作为开发工具, 可以在上海市体育场馆管理系统的开发中获得跨平台支持、强大的代码编辑功能、内置终端、丰富的插件生态系统、Git 集成以及轻量级和快速启动等好处。这将提供良好的开发体验和高效的工作流程, 有助于项目开发的顺利进行。

7. 系统实现

7.1 系统架构搭建

本系统的系统架构搭建如下：

整体架构概述：

该系统采用了经典的三层架构模式，包括表示层（前端）、应用层（后端）和数据层（数据库）。这种架构模式具有良好的可扩展性、可维护性和可测试性，能够有效分离不同层次的功能和责任。

1. 表示层（前端）：

- 前端采用 Vue.js 作为主要的 JavaScript 框架，用于构建用户界面和实现交互逻辑。
- 使用 HTML、CSS 和 JavaScript 技术开发前端页面，包括用户预约界面、管理员管理界面、个人信息修改界面等。
- 前端通过 HTTP 协议与后端进行通信，发送请求并接收响应。

2. 应用层（后端）：

- 后端采用 Flask 作为主要的 Web 框架，用于处理前端请求、业务逻辑处理和返回响应。
- 使用 Python 语言开发后端应用程序，包括用户管理、体育馆管理、设施预约管理等功能模块。
- 后端与前端进行数据交互，处理用户请求、验证权限、调用服务层处理业务逻辑，并通过 API 接口返回结果。

3. 数据层（数据库）：

- 数据库采用 MySQL 作为数据存储和管理系统，用于持久化存储用户信息、体育馆信息、设施预约等数据。
- 使用 SQL 语言进行数据库的建模、表设计和数据操作，确保数据的完整性、一致性和安全性。

- 后端通过 MySQL 连接库与数据库进行交互，执行数据查询、插入、更新和删除操作。

本项目采用了经典的三层架构模式，通过 Vue.js 作为前端框架、Flask 作为后端框架，结合 MySQL 数据库实现了用户管理、体育馆管理、设施预约等功能。这种架构模式使系统的各个层次分离，降低了耦合性，提高了系统的可维护性和可扩展性，同时使前后端的开发和测试工作更加独立和高效。

7.2 系统逻辑设计

本项目的系统逻辑设计基于上海市体育场馆管理系统的功能需求，以下是系统的功能实现逻辑原理：

1. 用户管理：

- 用户注册：用户在系统中进行注册，提供必要的个人信息并选择用户类型（管理员或普通用户）。
- 用户登录：注册后的用户使用其登录凭据（用户名和密码）登录系统。
- 用户身份验证：系统验证用户身份，并根据用户类型（管理员或普通用户）分配相应的权限和功能。
- 用户查询：输入用户名称，点击查询可以看到含有关键字的用户。

2. 体育馆管理：

- 体育馆信息展示：系统展示可预约的体育馆列表，包括体育馆名称、所在区域、地址等信息。
- 体育馆设施管理：管理员可以添加、编辑或删除体育馆设施信息，包括设施名称、所属体育馆等。
- 开放时间段管理：管理员可以设定每个体育馆的开放时间段，包括日期和时间。

3. 设施预约管理：

- 预约设施：用户可以选择所需的设施、预约日期和时间段进行预约。
- 预约确认和拒绝：管理员可以确认预约，或拒绝预约。

4. 个人信息修改：

- 用户个人信息展示：用户可以查看和编辑自己的个人信息，包括用户名、邮箱、电话等。
- 密码修改：用户可以修改登录密码，确保账户安全性。

系统的功能实现逻辑原理是通过前端和后端之间的交互实现的：

- 前端通过界面和用户交互，将用户的操作请求发送到后端。
- 后端接收前端的请求，根据请求的类型和数据进行相应的处理。
- 后端根据具体的业务逻辑，对数据库进行读取、写入和修改操作，以满足用户的需求。
- 后端将处理结果返回给前端，前端根据返回的结果进行界面的更新和展示。

通过这样的交互过程，系统能够实现用户管理、体育馆管理、设施预约管理和个人信息修改等功能，提供用户友好的界面和良好的用户体验。同时，系统的逻辑设计确保了数据的完整性、一致性和安全性，以及对用户权限的正确控制和验证。

7.3 具体功能编写

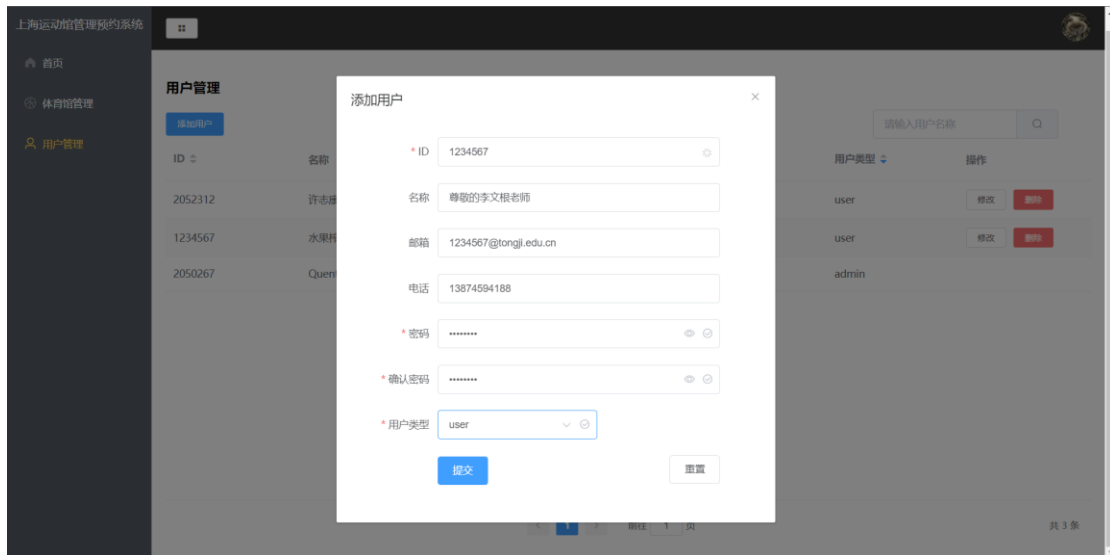
本项目的具体功能编写十分复杂，因此挑选了一些具有代表性的功能进行讲解：

1. 用户管理：

用户注册

首先介绍一下用户的注册功能，目前仅限于管理员注册用户，后续项目改进可能在登陆界面增添注册功能。

以管理员身份登录，点击左侧菜单列中的“用户管理”，点击左上角蓝色按钮“添加用户”，填写相关的信息后点击“提交”，一个用户就创建完毕了，如下图所示：



具体的代码流程如下图所示：

```
<h1>用户管理</h1>
<div style="display: flex; align-items: center; justify-content: space-between;">
  <el-button type="primary" @click="add_dialog_visible = true, showAlert = false" size="small">添加用户</el-button>
  <!-- 搜索 -->
  <el-input placeholder="请输入用户名称" v-model="user_name.user_name" class="input-with-select"
    style="width: 20%; padding-right: 45px;">
    <el-button slot="append" icon="el-icon-search" @click="handleSearch()"></el-button>
  </el-input>
</div>
```

这是一个 Vue.js 组件中的模板代码，用于渲染一个按钮，并绑定点击事件。点击按钮时，会触发 `add_dialog_visible` 变量的更新为 `true`。

```

<el-dialog title="添加用户" :visible.sync="add_dialog_visible" width="40%" :before-close="handleClose">
  <el-form ref="ruleFormRole" :model="user_form2" status-icon :rules="rules" label-width="120px"
    class="add-ruleForm">
    <el-form-item label="ID" prop="user_id">
      <el-input v-model="user_form2.user_id" autocomplete="off"></el-input>
    </el-form-item>
    <el-form-item label="名称" prop="user_name">
      <el-input v-model="user_form2.user_name" autocomplete="off"></el-input>
    </el-form-item>
    <el-form-item label="邮箱" prop="user_email">
      <el-input v-model="user_form2.user_email" autocomplete="off"></el-input>
    </el-form-item>
    <el-form-item label="电话" prop="user_phone">
      <el-input v-model="user_form2.user_phone" autocomplete="off"></el-input>
    </el-form-item>
    <el-form-item label="密码" prop="user_password">
      <el-input type="password" v-model="user_form2.user_password" autocomplete="off"
        show-password></el-input>
    </el-form-item>
    <el-form-item label="确认密码" prop="check_password">
      <el-input type="password" v-model="user_form2.check_password" autocomplete="off"
        show-password></el-input>
    </el-form-item>
    <el-form-item label="用户类型" prop="user_type">
      <el-select v-model="user_form2.user_type" placeholder="请选择类型">
        <el-option label="user" value="user"></el-option>
        <el-option label="admin" value="admin"></el-option>
      </el-select>
    </el-form-item>
    <el-form-item>
      <el-button type="primary" @click="submitForm(ruleFormRole)"
        :disabled="!user_form2.user_id || !user_form2.user_type">提交</el-button>
      <el-button @click="resetForm(ruleFormRole)" style="float: right;">重置</el-button>
    </el-form-item>
  </el-form>
  <el-alert v-if="showAlert" title="该用户已存在，请更改ID" type="warning" show-icon @close="showAlert = false" />
  <el-alert v-if="showAlertPwd" title="密码不匹配，请检查" type="warning" show-icon @close="showAlertPwd = false" />
</el-dialog>

```

这里用于渲染一个弹出对话框（dialog）以添加用户的表单，当 `add_dialog_visible` 为 `true` 时弹出，接着使用 Element UI 库提供的表单组件，填写相关信息后点击“提交”按钮触发 `submitForm` 函数。

```

const submitForm = (form) => {
  if (user_form2.user_password !== user_form2.check_password) {
    showAlertPwd.value = true
    console.log('提交失败');
    return
  }
  axios.post('http://127.0.0.1:5000/user/', user_form2).then((response) => {
    const result = response.data;
    if (result.status === 'success') {
      add_dialog_visible.value = false
      form.resetFields()
      getUsers()
    }
    else {
      showAlert.value = true
      console.log('提交失败');
    }
  })
}

```

`submitForm` 函数先验证密码和确认密码是否匹配。如果密码不匹配，将 `showAlertPwd` 变量设置为 `true`，显示密码不匹配的警告，并打印“提交失败”的提示信息。

如果匹配，使用 axios 库发送 HTTP POST 请求到指定的 URL

'http://127.0.0.1:5000/user/'，并传递表单数据 user_form2。在请求返回后的回调函数中，根据返回结果判断是否提交成功。如果返回结果的 status 为 'success'，则表示提交成功，将 add_dialog_visible 变量设置为 false，关闭对话框；调用 form.resetFields() 重置表单项；调用 getUsers() 函数更新用户列表。如果提交失败，将 showAlert 变量设置为 true，显示提交失败的警告，并打印“提交失败”的提示信息。

```
class USERapi(MethodView):
    def get(self, user_id): ...

    def delete(self, user_id): ...

    def post(self):
        form = request.json
        user_id = form.get('user_id')
        user_name = form.get('user_name')
        user_email = form.get('user_email')
        user_phone = form.get('user_phone')
        user_password = form.get('user_password')
        user_type = form.get('user_type')

        # 检查一下重复插入
        cursor.execute("SELECT COUNT(*) FROM user WHERE user_id = %s", user_id)
        result = cursor.fetchone()
        if result['COUNT(*)']:
            return {
                'status': 'failure',
                'message': '数据添加失败',
            }

        cursor.execute("insert into user (user_id, user_name, user_email, user_phone, user_password, user_type) values(%s,%s,%s,%s,%s,%s)",
            (user_id, user_name, user_email, user_phone, user_password, user_type))
        db.commit()
        return {
            'status': 'success',
            'message': '数据添加成功',
        }
```

在后端，首先，通过`app.add_url_rule()`函数将 URL 规则`/user/`与对应的视图函数`user_view`关联起来，并指定支持的 HTTP 方法。这意味着当用户访问`/user/`时，可以执行对应的视图函数进行处理。

接下来，定义了视图函数`post()`，用于处理 POST 请求。在函数中，首先从请求的 JSON 数据中获取用户的相关信息，如`user_id`、`user_name`、`user_email`等。

然后，通过执行 SQL 语句检查是否存在重复插入。如果不存在重复插入，则使用`cursor.execute()`函数执行 SQL 插入语句，将用户信息插入到数据库中，并通过`db.commit()`提交事务。

最后，返回一个包含成功状态和消息的 JSON 响应给前端。

用户注册

接着介绍一下用户注册，界面如下图所示，可以通过用户名和密码来判断是否是管理员还

是普通用户。



```
<el-form :model="loginForm" :rules="rules" ref="loginForm" label-width="21%" class="loginForm">
  <el-form-item label="账户" prop="username" style="width: 380px">
    <el-input v-model="loginForm.username"></el-input>
  </el-form-item>
  <el-form-item label="密码" prop="password" style="width: 380px">
    <el-input type="password" v-model="loginForm.password"></el-input>
  </el-form-item>
  <el-form-item class="button-group" type="string">
    <el-button type="primary" @click="submitForm('loginForm')>登录</el-button>
    <el-button @click="resetForm('loginForm')>重置</el-button>
  </el-form-item>
</el-form>
```

这是一个表单，就是上图中白底显示的部分，用于接收数据。

```

submitForm(formName) {
  this.$refs[formName].validate((valid) => {
    if (valid) {
      // 表单验证成功
      axios.post('http://127.0.0.1:5000/login/', this.loginForm).then(res => {

        // 拿到结果
        let status = res.data.status;
        let message = res.data.message;
        this.loginForm.id = res.data.user_id;
        this.loginForm.user_type = res.data.user_type;

        // 判断结果
        if (status) {
          /*登陆成功*/
          const now = new Date();

          // 格式化时间
          this.dateInfo.year = now.getFullYear();
          this.dateInfo.month = ('0' + (now.getMonth() + 1)).slice(-2);
          this.dateInfo.day = ('0' + now.getDate()).slice(-2);
          Element.Message.success(message);
          this.cookie.setCookie(this.loginForm, 1)
          this.cookie.setCookie(this.dateInfo, 365)
          if (res.data.user_type === 'admin') {
            /*跳转页面*/
            router.push('/admin/home')
          }
          else {
            /*跳转页面*/
            router.push('/user/home')
          }
        } else {
          /*打印错误信息*/
          Element.Message.error(message);
        }
      })
    }
  })
}

```

这里就是根据用户名和密码去后端查询，值得注意的是这里使用了 `setCookie()` 方法来设置浏览器的 Cookie，这样做的好处就可以使得浏览器记住这个用户是否登录过，并且后续还可以根据 cookie 直接获得当前进行操作的用户的一些信息，避免了不必要的查询，十分方便。

接着根据服务器返回的响应数据中的 ``user_type`` 字段的值进行条件判断，并根据不同的值来进行页面跳转。

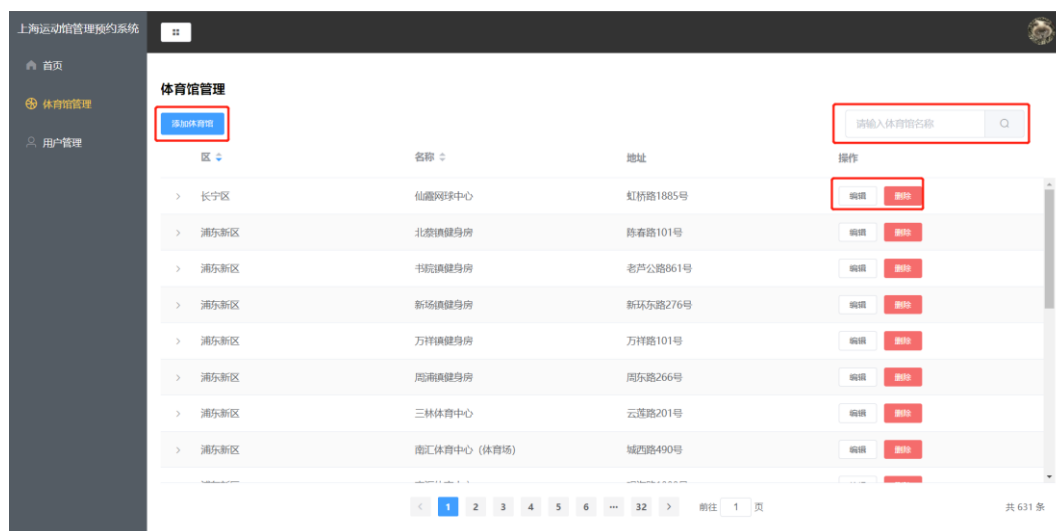
如果 ``user_type`` 的值为 ``admin``，则通过 ``router.push('/admin/home')`` 将页面跳转到管理员的首页。

如果 ``user_type`` 的值不为 ``admin``，即为其他值（可能是 ``user`` 或其他），则通过 ``router.push('/user/home')`` 将页面跳转到普通用户的首页。

2. 体育馆管理:

体育馆的增删改查

以管理员身份登录, 点击菜单列的“体育馆管理”就可以看到此页面, 然后就可以实现体育馆的增删改查, 前端代码 (/admin/Gym.vue) 过于冗长不在此展示。



```
class GYMapi(MethodView):  
    def __init__(self): ...  
  
    def get(self, gym_id): ...  
  
    def delete(self, gym_id, facility_name=None): ...  
  
    def post(self): ...  
  
    def put(self, gym_id): ...
```

增删改查每一个操作对应一个 http 请求, 都需要编写一个 API 接口, 上图中代码实现了一个名为 GYMapi 的类, 该类继承了 MethodView, 用于处理与健身房相关的 API 请求。

该类中定义了以下几个方法:

get(self, gym_id): 处理 GET 请求, 根据传入的 gym_id 参数进行查询。如果 gym_id 为空, 则返回所有健身房的信息, 否则返回指定 gym_id 的健身房信息。

delete(self, gym_id, facility_name=None): 处理 DELETE 请求, 根据传入的 gym_id 和可选的 facility_name 参数进行删除操作。如果只传入 gym_id, 则删除该健身房及其相关

的设施和时间表；如果同时传入 `gym_id` 和 `facility_name`，则删除指定健身房的指定设施及其相关时间表。

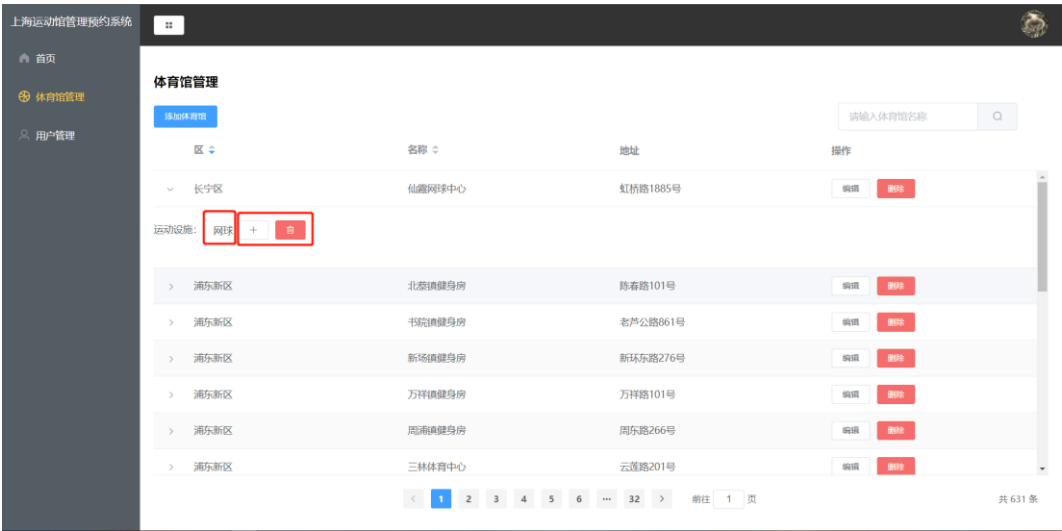
`post(self)`：处理 POST 请求，用于添加数据。根据请求中的参数进行插入操作，可以插入健身房信息、设施信息等。

`put(self, gym_id)`：处理 PUT 请求，用于更新数据。根据传入的 `gym_id` 和请求中的参数，更新指定健身房的信息。

每个方法根据具体的请求类型和参数，执行相应的数据库操作，并返回对应的响应结果。

体育设施的增删改

对每一行体育馆的记录，点击“区”这列记录左侧的展开，可以清晰地看到这个体育馆中有哪些运动设施，并且点击左侧也可以实现增添和删除功能。



进一步地，点击设施“网球”本身，将可以对该设施的开放时间段进行增删改，如下图所示，具体代码与其余增删改代码结构类似，实现细节可以查看代码（`/admin/PageOne.vue`）。增删改本身都类似，主要问题就是如何传递并依靠正确的信息对数据库进行操作，不同功能需要特定的代码与处理方法。

日期	开放时间	结束时间	操作
2022-02-06	08:00:00	21:00:00	编辑 删除
2022-02-05	08:00:00	21:00:00	编辑 删除
2022-02-04	08:00:00	21:00:00	编辑 删除
2022-02-03	08:00:00	21:00:00	编辑 删除
2022-02-02	08:00:00	21:00:00	编辑 删除
2022-02-01	08:00:00	16:00:00	编辑 删除
2022-01-31	08:00:00	16:00:00	编辑 删除

3. 设施预约管理：

预约设施

以用户身份登录，点击菜单列的“体育馆预约”，选择一个想要预约的体育馆，点击绿色按钮“预约”，在对话框中选择信息并可以选择“提交”。

点击提交函数后，函数首先对表单的一些信息进行补充，然后 post 到后端，本质上和增加一条记录一样，增加一条预约信息，函数代码如下所示：

```

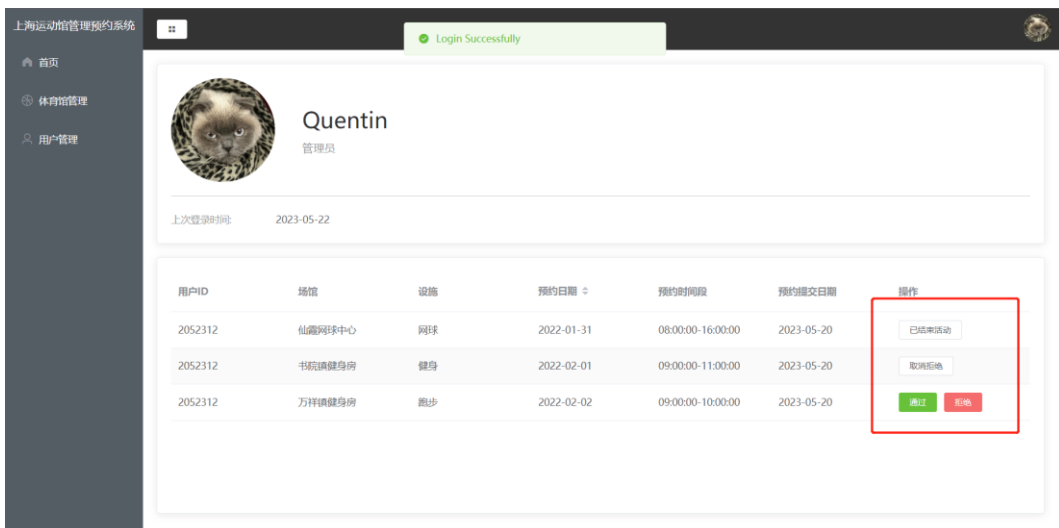
submitForm() {
  if (this.reservation.schedule_time.length == 0) {
    this.showAlert_ = true
    console.log('提交失败');
    return
  }
  this.year = this.cookie.getCookie('year')
  this.month = this.cookie.getCookie('month')
  this.day = this.cookie.getCookie('day')
  this.reservation.reservation_date = `${this.year}-${this.month}-${this.day}`
  this.reservation.user_id = this.cookie.getCookie('id')
  console.log(this.reservation);

  axios.post('http://127.0.0.1:5000/gym/', this.reservation).then((response) => {
    const result = response.data;
    if (result.status == 'success') {
      this.reserve_dialog_visible = false
      this.$refs.ruleFormRole.resetFields();
      this.getGyms()
      this.open1()
    }
    else {
      this.showAlert = true
      console.log('提交失败');
    }
  })
},

```

预约确认或拒绝

随后可以以管理员身份登录，在主页就可以看到用户预约的请求，并进行同意或拒绝。



The screenshot shows the '上海运动馆管理预约系统' (Shanghai Sports Center Management System) interface. A user profile for 'Quentin' (管理员) is displayed, showing a login time of 2023-05-22. Below the profile is a table of reservations with columns for user ID, gym, facility, reservation date, reservation time, and reservation submission date. The table contains three rows of reservations. A red box highlights the '操作' (Action) column, which contains buttons for '已结束活动' (Activity Ended), '取消预约' (Cancel Reservation), '通过' (Approve), and '拒绝' (Reject).

用户ID	场馆	设施	预约日期	预约时间段	预约提交日期	操作
2052312	仙霞网球中心	网球	2022-01-31	08:00:00-16:00:00	2023-05-20	已结束活动
2052312	书院镇健身房	健身	2022-02-01	09:00:00-11:00:00	2023-05-20	取消预约
2052312	万祥镇健身房	跑步	2022-02-02	09:00:00-10:00:00	2023-05-20	通过 拒绝

```

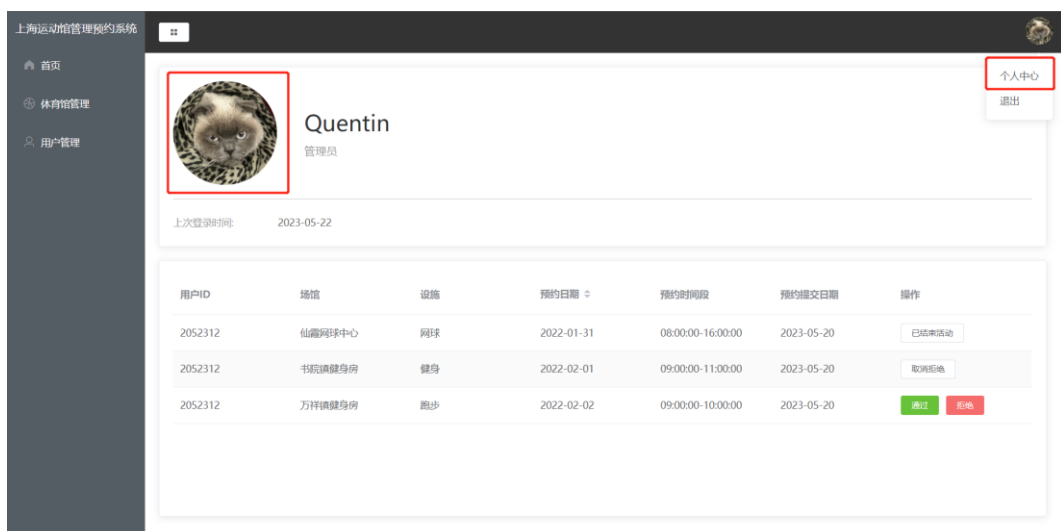
Pass(index, row) {
  this.reservation[index].status = 2
  this.editStatus(index, row.reservation_id)
},
Reject(index, row) {
  this.reservation[index].status = 4
  this.editStatus(index, row.reservation_id)
},
reversePass(index, row) {
  this.reservation[index].status = 1
  this.editStatus(index, row.reservation_id)
},
reverseReject(index, row) {
  this.reservation[index].status = 1
  this.editStatus(index, row.reservation_id)
},
finish(index, row) {
  this.reservation[index].status = 3
  this.editStatus(index, row.reservation_id)
},
editStatus(index, id) {
  axios.put(`http://127.0.0.1:5000/reservation/${id}`, this.reservation[index]).then((res) => {
    this.getReservation()
  })
}
}

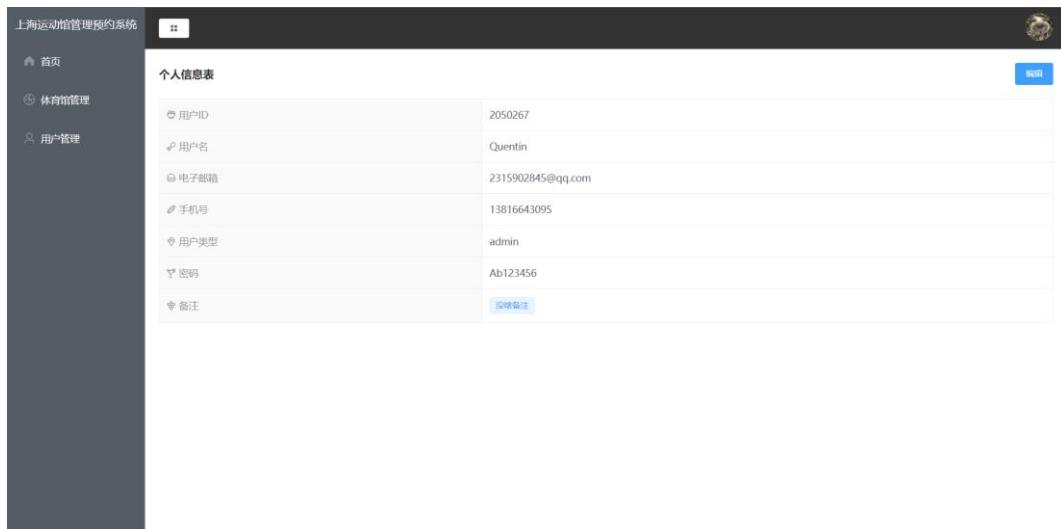
```

在代码中就是对 status 进行修改，以表达预约请求不同的状态，然后执行 put 请求对数据库进行修改。

4. 个人信息修改

无论是普通用户还是管理员，点击红框都可以进入个人信息页面。





```
<el-descriptions class="margin-top" title="个人信息表" :column="3" border>
  <template slot="extra">
    <el-button type="primary" size="small" v-if="!isEditing" @click="isEditing = true">编辑</el-button>
    <el-button type="primary" size="small" v-else @click="submitEditForm()">保存</el-button>
  </template>
  <el-descriptions-item span="3">
    <template slot="label">
      <i class="el-icon-food"></i>
      用户ID
    </template>
    <div>{{ ID }}</div>
    <!-- <el-input size="mini" v-else v-model="user.user_id"></el-input> -->
  </el-descriptions-item>
  <el-descriptions-item span="3">
    <template slot="label">
      <i class="el-icon-chicken"></i>
      用户名
    </template>
    <div v-if="!isEditing">{{ user.user_name }}</div>
    <el-input size="mini" v-else v-model="user.user_name"></el-input>
  </el-descriptions-item>
  <el-descriptions-item span="3">
    <template slot="label">
      <i class="el-icon-burger"></i>
      电子邮箱
    </template>
    <div v-if="!isEditing">{{ user.user_email }}</div>
    <el-input size="mini" v-else v-model="user.user_email"></el-input>
  </el-descriptions-item>
</el-descriptions>
```

```
getOneUser() {
  axios.get(`http://127.0.0.1:5000/user/${this.ID}`).then(res => {
    this.user = res.data
    console.log('更新数据')
  })
},
handleClose(done) {
  done();
},
submitEditForm() {
  axios.put(`http://127.0.0.1:5000/user/${this.ID}`, this.user).then((res) => {
    this.isEditing = false
    this.getOneUser()
  })
}
```

然后根据情况进行修改以及保存等。

7.4 功能测试

详细功能测试移步运行答辩 PPT，其中从登录到增删改查预约等功能都有完整的演示。

8. 总结

本项目是一个健身房管理系统，旨在提供一个全面的健身房管理解决方案。项目的开发旨在满足健身房管理的需求，包括用户管理、设施管理、预约管理等功能。在项目的开发过程中，我们使用了现代化的技术栈，包括 Vue.js 作为前端框架、Flask 作为后端框架以及 MySQL 作为数据库。

在项目的架构设计上，我们采用了前后端分离的架构方式。前端使用 Vue.js 框架搭建了用户界面，通过与后端的 API 进行数据交互和页面渲染。后端使用 Flask 框架搭建了 RESTful 风格的 API，处理前端的请求，并与 MySQL 数据库进行数据交互。整个系统的架构清晰简洁，各组件之间的交互有机地配合，保证了系统的稳定性和扩展性。

在功能实现方面，本项目涵盖了用户管理、设施管理和预约管理等核心功能。用户管理模块提供了用户注册、登录、权限管理等功能，确保系统安全可靠。设施管理模块支持添加、删除和查询健身房设施，帮助管理员有效管理设施资源。预约管理模块允许用户预约健身房设施，管理员可以审核和管理预约信息。这些功能的实现基于前后端的配合，提供了用户友好的界面和顺畅的操作体验。

在数据库设计阶段，需要仔细分析业务需求，理清实体之间的关系，并根据需求合理地设计数据库模式。在设计过程中，我们尽量遵循数据库设计的范式原则，确保数据的一致性和完整性。

在开发过程中，我们充分利用了现代化的开发工具和技术。集成开发环境 VS Code 提供了便捷的代码编写和调试环境，大大提高了开发效率。此外，通过使用第三方库和插件，如 Element UI、Axios 和 MySQL Connector 等，我们简化了开发过程，提供了更多的功能和可靠性。

总的来说，本项目是一个功能完备、架构清晰、技术先进的健身房管理系统。通过该系统，用户可以方便地管理健身房的用户、设施和预约信息，实现了高效的健身房管理。在项目

开发过程中，我们积极运用现代化的技术和工具，提高了开发效率和代码质量。这个项目的成功实施为健身房管理提供了一个可靠的解决方案，具有广泛的应用前景和市场价值。

数据库设计和管理是一个持续的过程，需要不断地学习和改进。随着项目的发展和需求的变化，数据库可能需要进行调整和优化。因此，要保持对新技术和最佳实践的学习，并灵活地应用到数据库设计和管理中，以满足不断变化的需求和提升系统的性能和可靠性。

未来可能会进一步完善相关功能预约冲突检测，检测用户预约的设施、日期和时间是否与其他预约发生冲突；登录页实现注册；用户可以取消已预约的设施或集成一些运动或地图相关等功能，为管理员增添一些分析统计功能，并且在系统运行和代码上进行一些优化。