**湘潭大学**

**软件设计说明书**

**题 目：企业职工信息管理系统设计与实现**

|  |  |
| --- | --- |
| **学 院：** | **计算机学院** |
| **专 业：** | **网络空间安全** |
| **组 长：** | **202205501204 李连志** |
| **组 员：** | **202205501204 李连志** |
| **指导教师：** | **刘新** |

**完成日期： 2025 年 8 月**

目 录

[摘要 I](#_Toc18578)

[Abstract II](#_Toc8504)

[第1章 引言 1](#_Toc9282)

[1.1 设计背景与意义 1](#_Toc14246)

[1.2 国内外研究发展现状 1](#_Toc7589)

[第2章 相关技术 3](#_Toc21794)

[2.1 开发框架 3](#_Toc10509)

[2.1.1 SpringBoot 3](#_Toc12494)

[2.1.2 Vue 3](#_Toc24626)

[2.2 系统部署 3](#_Toc10509)

[2.2.1 Docker 3](#_Toc12494)

[2.2.2 MySQL 3](#_Toc24626)

[第3章 需求分析 4](#_Toc18847)

[3.1 系统概述 4](#_Toc16075)

[3.2 系统功能需求分析 4](#_Toc14757)

[3.2.1 数据库API服务功能需求 5](#_Toc15469)

[3.2.2 数据展示系统功能需求 8](#_Toc4802)

[3.3 系统数据迁移同步 8](#_Toc2476)

[第4章 系统设计 1](#_Toc13662)0

[4.1 系统总体设计 10](#_Toc18017)

[4.2 系统详细设计 11](#_Toc18113)

[4.2.1 数据库API服务相关功能设计 11](#_Toc7257)

[4.2.2 数据展示后端相关功能设计 12](#_Toc1172)

[4.3 系统数据库设计 13](#_Toc2379)

[第5章 系统实现 16](#_Toc6565)

[5.1 系统文件结构 16](#_Toc18700)

[5.1.1 数据库API服务文件结构 16](#_Toc9639)

[5.1.2 数据展示后端文件结构 17](#_Toc27348)

[5.2 相关功能模块实现 18](#_Toc30718)

[5.2.1 数据库API服务相关功能实现 18](#_Toc12638)

[5.2.2 数据展示后端相关功能实现 24](#_Toc15830)

[5.3 系统难点及实现 31](#_Toc7443)

[5.3.1 数据库迁移同步 31](#_Toc9756)

[5.3.2 项目单程调用 37](#_Toc14513)

[第6章 系统功能测试 42](#_Toc21008)

[6.1 系统测试环境 42](#_Toc17420)

[6.2 数据库API服务功能测试 42](#_Toc24450)

[6.3 数据展示后端功能测试 43](#_Toc30159)

[第7章 总结与展望 44](#_Toc518)

[7.1 总结 44](#_Toc5238)

[7.2 展望 44](#_Toc2712)

[参考文献 45](#_Toc26575)

**企业职工信息管理系统设计与实现**

摘要：在数字化转型加速的时代背景下，企业高效、安全的人力资源管理日益成为提升组织效能的关键。本项目基于SpringBoot和Vue3开发了一套企业职工信息管理系统，实现了员工信息的集中化、标准化管理，有效支持现代企业灵活办公与精细化管理的需求。系统区分管理员与普通员工两类角色：管理员可全面管理员工信息、部门架构、考勤与工资数据，并进行请假审核；员工则可自主查看个人记录、提交请假申请及修改密码，体现了权限分离与用户自主性的设计理念。结合Spring Security权限控制与数据加密机制，系统在便捷性基础上全面保障信息安全。该系统不仅规范了人事流程，提升了管理效率，更契合后疫情时代企业远程协同与数据驱动决策的发展趋势，具有良好的实用性与推广价值。

**关键字：SpringBoot ,Vue3 ,Element Plus,数字化管理**

Enterprise Employee Information Digital Management System Design and Implementation

Abstract:In the context of an accelerating digital transformation, efficient and secure human resource management has become increasingly critical for enhancing organizational performance. This project developed an Enterprise Employee Information Management System based on SpringBoot and Vue3, which achieves centralized and standardized management of employee data, effectively supporting the needs of modern enterprises for flexible operations and refined management.The system distinguishes between two types of roles: administrators and general employees. Administrators have comprehensive control over employee information, departmental structure, attendance, and payroll data, as well as the authority to review leave requests. Employees, on the other hand, can independently access their personal records, submit leave applications, and change their passwords, reflecting a design philosophy that emphasizes both role-based access control and user autonomy. Integrated with Spring Security for permission management and data encryption mechanisms, the system ensures comprehensive information security without compromising usability.Not only does the system streamline personnel processes and improve management efficiency, but it also aligns with the trends of remote collaboration and data-driven decision-making in the post-pandemic era, demonstrating strong practical value and potential for broader adoption.

**Key words:SpringBoot ,Vue3 ,Element Plus,digitial management**

1. 引言
   1. 设计背景与意义

本系统旨在解决企业员工信息管理中的痛点，提升人力资源管理效率。随着企业规模的扩大和业务的复杂化，传统的人工管理方式已难以满足需求。本系统通过数字化手段，实现员工信息的集中化、标准化管理，规范人事管理流程，降低运营成本，提高决策效率。这对于提升企业竞争力，适应数字化时代发展具有重要意义。

* 1. 国内外研究发展现状

针对暗网中非法信息泛滥的问题，国内外的专家学者已经开始着手对暗网信息采集及监控的相关技术展开研究。汤艳君等人在分析暗网匿名通信系统Tor技术原理基础上,设计了一套基于Selenium的暗网爬虫,通过定义的流程自动化地采集暗网网页的数据，为暗网数据采集提供了一定的借鉴意义[1]。于浩佳等人通过Polipo与Tor相结合的方式，优化了爬虫接入Tor网络的速度，显著提高了信息采集效率，监管暗网提出了一种可行的技术手段[2]。国外的Cafarella M J等人开发了一种名为WISE-Integrator的工具,其可以对暗网网页进行自动化文本提取，

并根据内容进行分类[3]。2016年, Eric Nunes等人构建了一个专门用于从以暗网为主的网站手机网络威胁情报的操作系统，同时利用机器学习和自然语言处理技术，对收集到的情报进行分类与分析[4]。上述研究对于本文有着很大的借鉴意义，但随着互联网技术的快速发展，上述研究中所用到的一些技术已经过时或失效，同时也存在着功能缺失的问题，面对暗网中越来越泛滥的违法犯罪行为，对暗网信息监测分析的研究有着十分重要的现实意义。

* 1. 研究的目标和内容
     1. 研究目标

本系统的研究目标是开发一个基于SpringBoot+Vue的企业员工信息数字化管理系统，实现员工信息管理、部门管理、考勤工资管理、请假审批等核心功能，并确保系统的安全性、稳定性和易用性，提升企业人力资源管理效率。

* + 1. 研究内容

本系统的研究内容主要包括：

1. 后端架构设计与实现：采用SpringBoot+MyBatis+SpringSecurity，实现用户认证、权限控制、数据接口等。

2. 前端界面开发：使用Vue3+Element Plus，构建直观友好的用户界面，实现数据展示和交互。

3. 数据库设计：设计合理的数据库结构，存储员工、部门、考勤、工资、请假等信息。

4. 功能模块开发：实现管理员和员工的各项功能，包括信息维护、查询、审批等。

5. 系统安全与性能优化：确保数据安全，提升系统响应速度和并发处理能力。

* 1. 研究思路

本系统将采用前后端分离的开发模式，后端负责数据处理和接口提供，前端负责用户界面展示。开发流程将遵循需求分析、系统设计、编码实现、测试部署的迭代开发模式。技术选型上，后端采用成熟稳定的SpringBoot生态，前端选择流行的Vue3框架，以保证开发效率和系统性能。

* 1. 本文的章节安排

本文档将详细介绍企业员工信息数字化管理系统的设计与实现过程，具体章节安排如下：

第1章：引言。介绍项目背景、意义、研究目标和内容。

第2章：相关技术。概述系统开发所使用的主要技术栈。

第3章：需求分析。详细阐述系统功能需求和非功能需求。

第4章：系统设计。介绍系统总体架构、模块设计和数据库设计。

第5章：系统实现。展示各功能模块的具体实现细节。

第6章：系统功能测试。描述测试方法和测试结果。

第7章：总结与展望。对项目进行总结并提出未来展望。

1. 相关技术

本系统基于SpringBoot+Vue技术栈开发，后端采用SpringBoot+MyBatis+SpringSecurity架构，前端使用Vue3+Element Plus。数据库方面，将使用MySQL存储关系型数据。系统部署将考虑使用Docker进行容器化部署，以提高可移植性和部署效率。本章将对这些关键技术进行简要介绍。

* 1. 开发框架

2.1.1 SpringBoot

SpringBoot是Spring框架的子项目，旨在简化Spring应用的初始搭建以及开发过程。它提供了“约定优于配置”的理念，内置了Tomcat、Jetty等Web服务器，可以独立运行Spring应用，极大地提高了开发效率。

2.1.2 Vue3

Vue3是一套用于构建用户界面的渐进式JavaScript框架。它易于上手，可与其他库或已有项目整合。Vue3在性能、TypeScript支持和Composition API方面进行了优化，使其更适合构建大型和复杂的单页应用。

2.2 系统部署

2.2.1 Docker

Docker是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的Linux机器或Windows机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

2.2.2 MySQL

MySQL是一个关系型数据库管理系统，是目前最流行的关系型数据库之一。它以其高性能、低成本、易于使用等特点，广泛应用于各种Web应用。

1. 需求分析
   1. 系统概述

企业员工信息管理系统旨在高效管理员工及部门信息，支持人事管理核心功能。系统分为管理员和普通员工两类用户：

管理员功能：

员工信息管理：维护员工基本信息（姓名、性别、职位等），仅允许密码修改权限下放至员工。

部门管理：支持部门及子部门的增删，便于组织架构调整。

出勤与工资管理：录入并管理员工的出勤天数与工资数据，员工仅可查看。

请假审核：处理员工请假申请，更新审核状态并反馈结果。

员工功能：

查看个人出勤、工资记录；

提交请假申请并跟踪状态；

修改个人密码。

系统需确保数据安全性与操作权限隔离，提升企业管理效率。

* 1. 系统功能需求分析

本系统主要功能需求如下：

管理员功能：

1. 员工信息管理：

- 维护员工基本信息（姓名、性别、职位、联系方式等）。

- 仅允许密码修改权限下放至员工，管理员不可直接修改员工密码。

2. 部门管理：

- 支持部门的增加、删除、修改。

- 支持子部门的增删改查，便于组织架构调整。

3. 出勤与工资管理：

- 录入员工每日出勤数据。

- 管理员工工资数据，包括基本工资、奖金、津贴等。

- 员工仅可查看自己的出勤和工资记录，无修改权限。

4. 请假审核：

- 查看所有员工提交的请假申请。

- 审核请假申请，更新审核状态（通过/驳回）。

- 反馈审核结果给员工。

员工功能：

1. 个人信息查看与修改：

- 查看个人基本信息。

- 修改个人密码。

2. 出勤与工资记录查看：

- 查看个人历史出勤记录。

- 查看个人工资明细。

3. 请假申请：

- 提交请假申请，填写请假类型、时间、原因等。

- 跟踪请假申请的审核状态。

* + 1. 数据库API服务功能需求

StaffManage数据库API服务是基于RESTful架构设计的标准化数据交互接口，为前端应用提供完整的业务数据支撑。该系统采用分层设计理念，包含统一的身份认证、数据操作、业务逻辑处理和标准化响应格式化等核心模块。API服务支持管理员和员工双身份体系，分别提供/admin/和/staff/两套接口路径，确保权限隔离和数据安全。所有接口均采用JWT令牌进行身份验证，请求头需携Authorization字段进行访问控制。数据交互采用JSON格式进行序列化，包含统一的响应状态码、业务数据和消息提示。系统提供完整的员工信息管理接口，支持员工信息的增删改查和密码重置功能。考勤管理模块提供每日签到记录和查询接口，支持按日期范围检索考勤数据。请假审批系统包含请假申请提交、状态查询和管理员审批流程，支持多状态筛选和分页查询。薪资管理接口提供工资记录创建、修改和按月查询功能，确保薪资数据的准确性和隐私性。部门管理模块支持部门创建、员工分配、部长设置和组织结构查询，满足企业层级管理需求。系统还提供数据统计和分析接口，包括考勤汇总、部门人数统计和薪资总额计算等业务指标。所有数据库操作都包含事务处理机制，保证数据的一致性和完整性。API服务采用参数验证和SQL注入防护措施，输入输出数据进行严格过滤和转义。服务支持跨域访问配置，提供完整的错误处理机制和日志记录功能，便于系统监控和故障排查。

其json统一格式如下：

ERD

admin {

varchar username PK

varchar password

}

staff {

bigint id PK

varchar username

varchar password

}

department {

int dept\_id PK

varchar dept\_name

bigint manager\_id FK

datetime created\_at

}

staff\_department {

bigint id PK

bigint staff\_id FK

int dept\_id FK

tinyint is\_manager

}

daily\_checkin {

bigint staff\_id PK, FK

date check\_date PK

}

leave\_application {

bigint id PK

bigint staff\_id FK

date start\_date

date end\_date

varchar reason

tinyint status

datetime created\_at

}

salary {

bigint id PK

bigint staff\_id FK

date date

decimal base\_salary

decimal bonus

decimal deduction

decimal total\_salary

datetime created\_at

}

staff ||--o{ staff\_department : "has"

department ||--o{ staff\_department : "has"

staff ||--o{ daily\_checkin : "checks in"

staff ||--o{ leave\_application : "applies for"

staff ||--o{ salary : "receives"

staff ||--o{ department : "manages"

* + 1. 数据展示系统功能需求

数据展示系统提供多维度数据可视化功能，支持员工考勤状态实时图表展示，包括月度出勤率统计和异常考勤标记。系统应集成请假数据看板，按部门、时间维度展示审批状态分布，并支持历史记录趋势分析。薪资模块生成工资条式明细视图。部门组织架构要求以树形图可视化呈现，清晰展示层级关系和人员配置。所有数据展示需支持条件筛选、数据导出和实时刷新功能，确保管理者能够直观获取人力资源关键指标。



图 3‑1 各模块功能图

* 1. 系统数据迁移同步

该企业员工信息管理系统在数据迁移同步方面，主要特点体现在其基于Spring Boot的后端API服务，通过定义清晰的Entity层（如`Admin.java`、`Employee.java`等）和Repository层（如`AdminRepository.java`、`EmployeeRepository.java`等），实现了对MySQL数据库的结构化数据管理和操作。系统通过RESTful API接口（如`AdminController.java`、`EmployeeController.java`等）提供数据访问服务，使得数据可以在前端应用与后端数据库之间进行高效、规范的交互。虽然没有直接看到明确的数据迁移或同步模块，但其模块化的设计和对Spring Data JPA的利用，使得在需要进行数据批量导入导出或与其他系统进行数据同步时，可以通过扩展Repository层或引入额外的批处理/消息队列机制来实现，保证了数据的一致性和可扩展性。

1. 系统设计
   1. 系统总体设计

该企业员工框架系统总体如图4-1所示。



图 4‑1 系统总体设计图

数据库API服务是为了简化数据采集子模块的开发而设计的，在数据采集子模块采集完数据后无需关心如何连接数据库、怎样存储数据、以什么样的格式存储数据，只需要参照数据库API接口文档，调用相应的接口，即可完成数据的存取。数据库API服务的业务流程如图4-2所示。

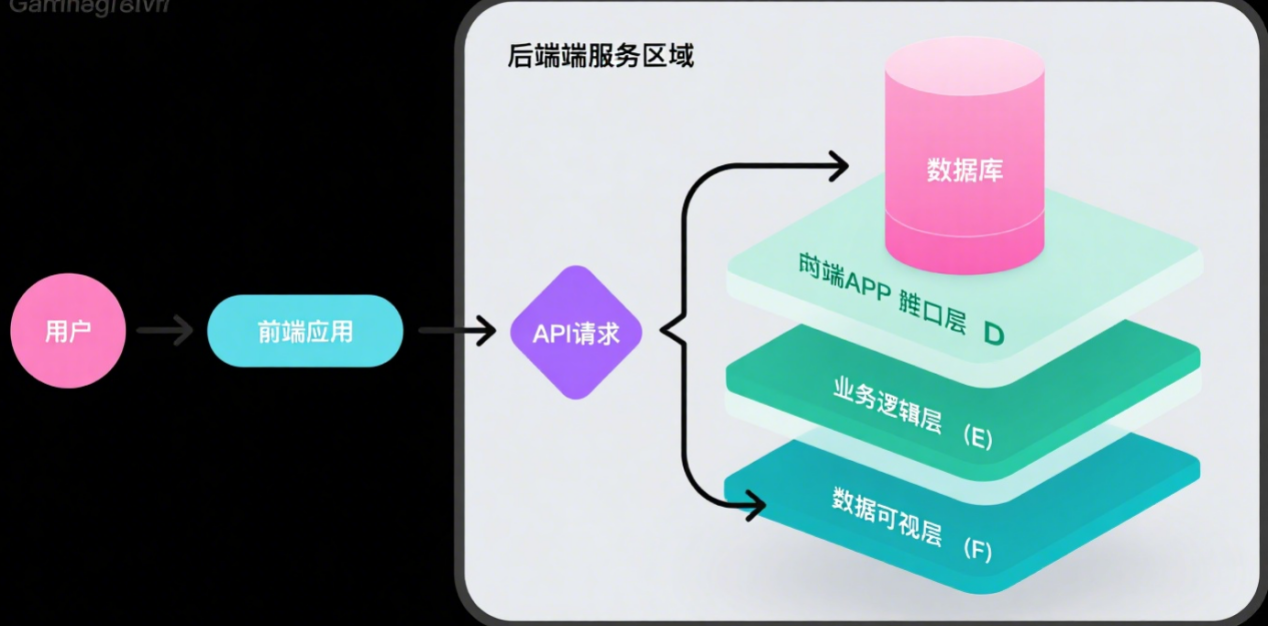


图 4‑2 数据库API服务业务流程设计图

数据展示系统后端是为前端信息展示、渲染图表、分类查询等功能提供数据支撑，前端使用POST或GET方式向后端相应接口发送请求并携带参数，后端接收到参数后会根据条件构造查询语句在数据库中进行查询，并根据前端需要的格式对数据进行去重、合并、格式化等操作，最终以JSON格式返回给前端。数据展示前后端的mermaid格式的数据展示流程图如图4-3所示。

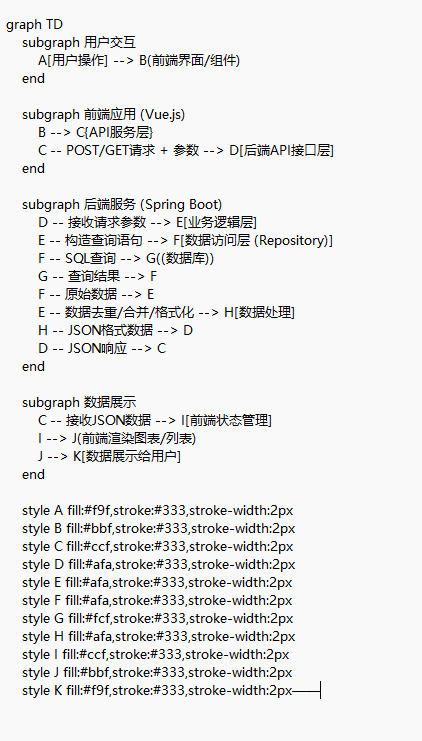


图 4‑3 数据展示系统前后端数据展示流程图

* 1. 系统详细设计
     1. 数据库API服务相关功能设计

StaffManage数据库API服务采用模块化架构设计，提供完整的RESTful接口体系。身份认证模块采用双路JWT令牌方案，为管理员和员工分别提供独立的登录接口（/admin/login和/staff/login）和权限控制中间件，令牌有效期内保持会话状态且支持自动续期。数据操作层采用多种技术，对所有输入参数进行类型验证和SQL注入过滤，通过事务机制保证数据一致性。业务逻辑层包含员工管理模块、考勤服务（每日签到记录和时间段查询）、请假工作流（申请提交、状态跟踪和审批操作）、薪资管理以及部门组织（树形结构管理和人员调配）等核心功能。接口响应标准化模块统一封装数据返回格式，包含状态码、业务数据和消息体，支持分页查询和字段选择性返回。系统内置操作日志记录，所有数据修改操作均留存审计痕迹，同时提供实时数据统计接口，部门人数、薪资等可视化指标数据。

* + 1. 数据展示后端相关功能设计

该项目的数据展示后端主要基于Spring Boot框架构建，核心职责是为前端提供数据支撑。它通过一系列RESTful API接口（如 `AdminController` 、 `EmployeeController` 、 `AttendanceController` 等）接收前端的POST或GET请求及参数。后端接收到请求后，会利用Spring Data JPA（通过 Repository 接口，如 `AdminRepository` ）与MySQL数据库进行交互，根据前端提供的条件构造查询语句并执行。查询结果经过业务逻辑层的处理，包括数据去重、合并、格式化等操作，最终以标准JSON格式返回给前端，供前端进行信息展示、图表渲染和分类查询等功能。整个后端设计遵循模块化原则，确保数据处理的准确性和高效性。

该项目的数据展示后端通过RESTful API为前端提供数据支撑。主要接口包括：

用户认证接口：处理用户登录、注册，返回认证令牌，确保数据访问安全。

员工信息查询接口：根据前端请求参数，查询员工基本信息，支持模糊搜索和分页。

部门信息查询接口：提供部门列表及详情，用于前端展示部门结构和筛选。

考勤数据接口：获取员工考勤记录，支持按日期、员工ID筛选，用于考勤统计。

薪资数据接口：提供员工薪资详情，支持按月份、员工ID查询，用于薪资管理。

数据统计接口：聚合各类数据，提供统计概览，如员工总数、部门人数分布等，用于图表展示。

数据去重与合并接口：对从数据库获取的原始数据进行清洗，去除重复项并合并相关数据。

数据格式化接口：将处理后的数据转换为前端所需JSON格式，确保数据结构统一，便于前端渲染。

这些接口共同构建了一个高效、安全的数据服务层，确保前端能够准确、及时地获取并展示所需数据。

数据展示后端系统设计如图4-4所示。

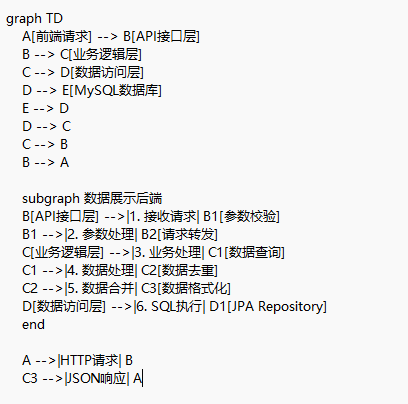


图 4‑4 数据展示后端系统设计图

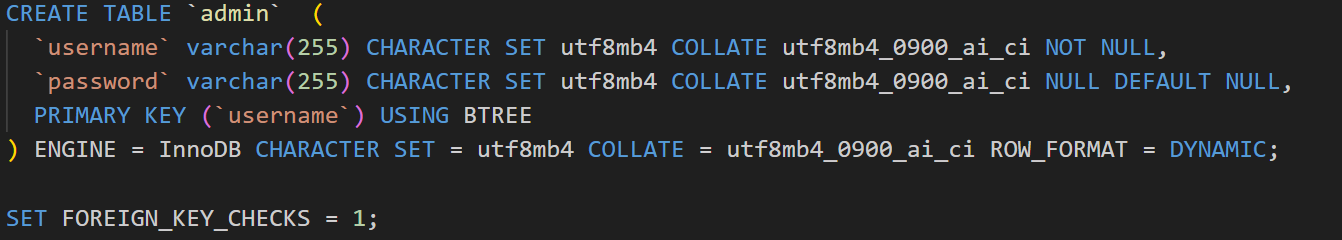
* 1. 系统数据库设计

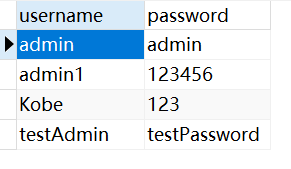
StaffManage 员工管理系统数据库介绍

StaffManage 是一个功能完善的员工管理系统，涵盖了员工信息管理、考勤签到、请假审批、薪资管理、部门组织等核心功能。系统采用模块化设计，通过多张数据表协同工作，为企业提供全面的员工管理解决方案。数据库设计注重数据完整性和业务逻辑的合理性，通过外键约束确保各表之间的关联关系正确性。

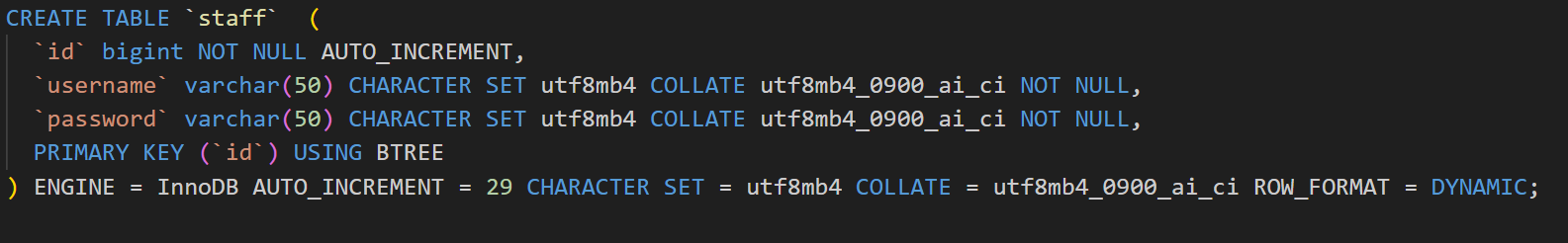
数据表总览

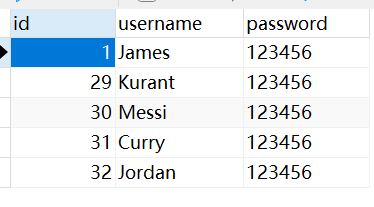
1. admin 表（管理员表）



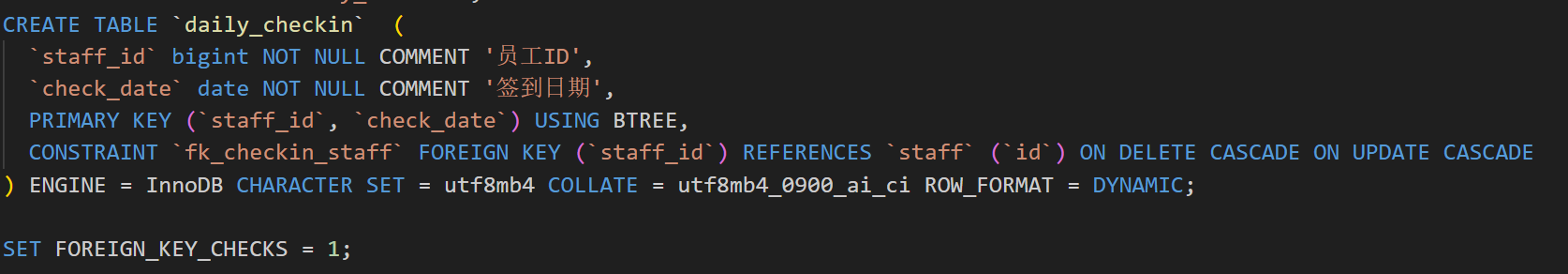


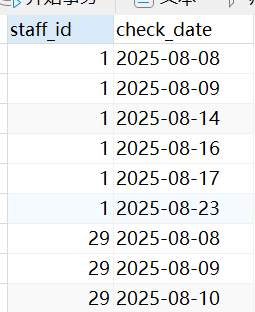
1. staff 表（员工表）



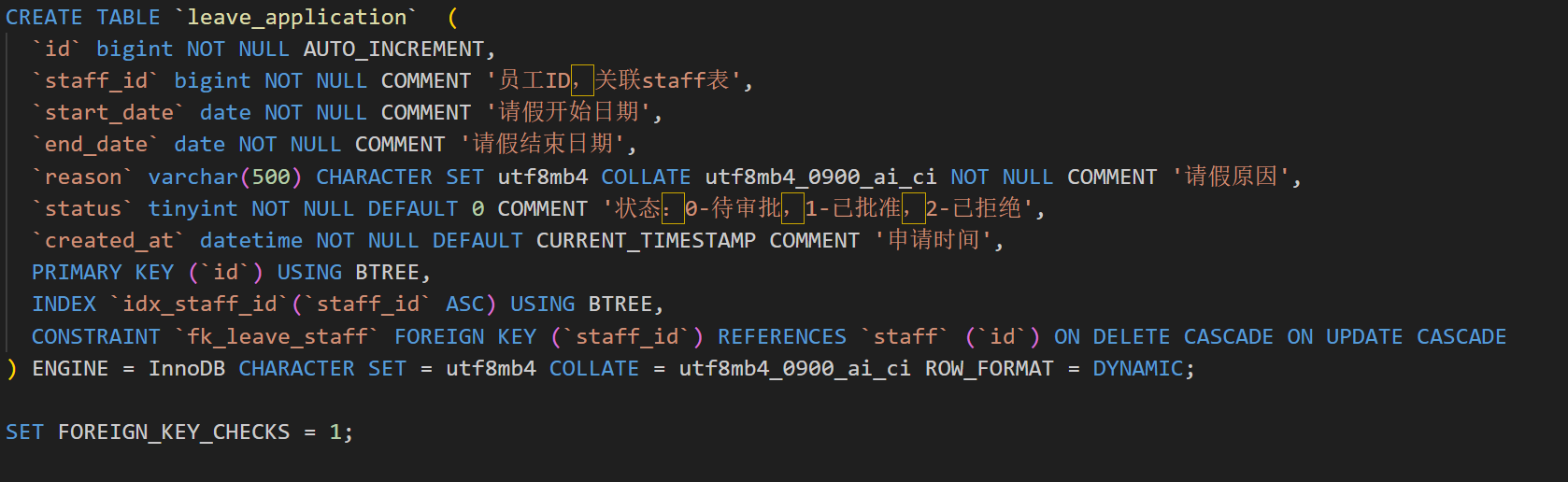


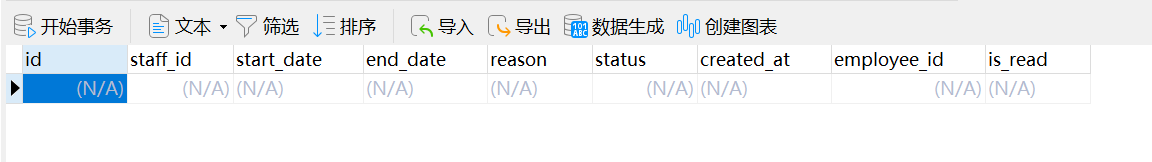
1. daily\_checkin 表（每日签到表）



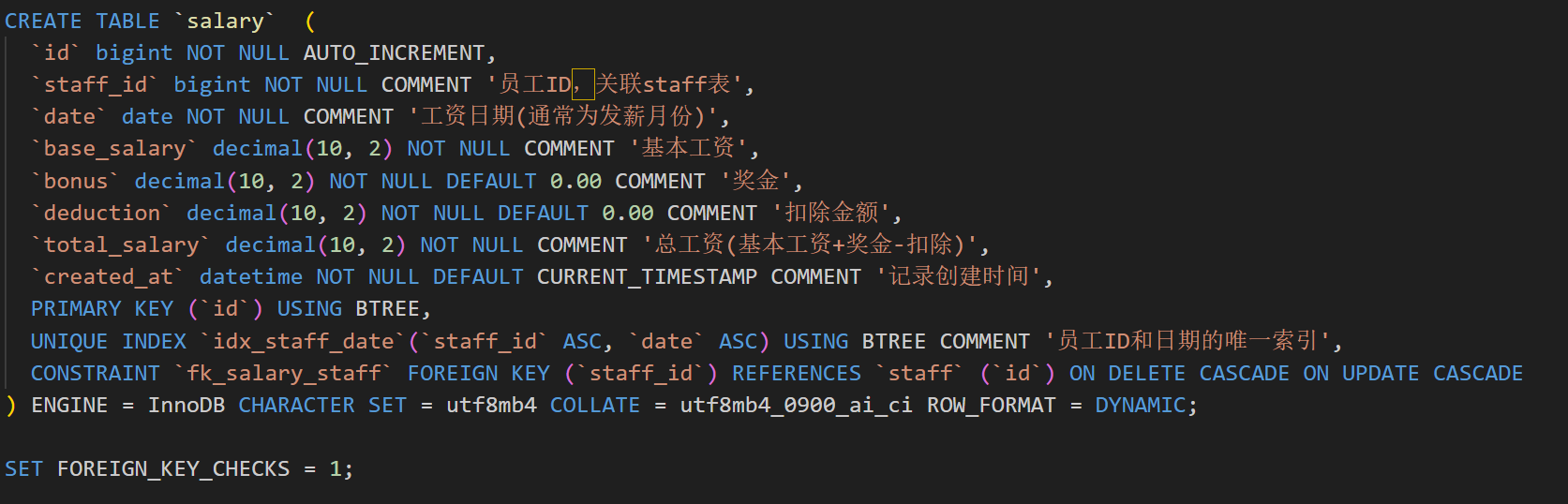


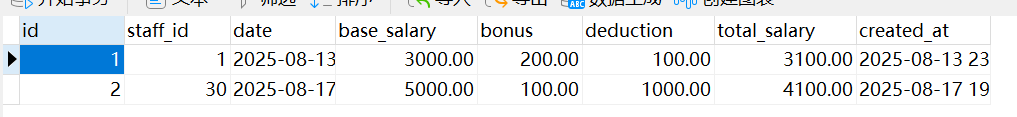
1. leave\_application 表（请假申请表）



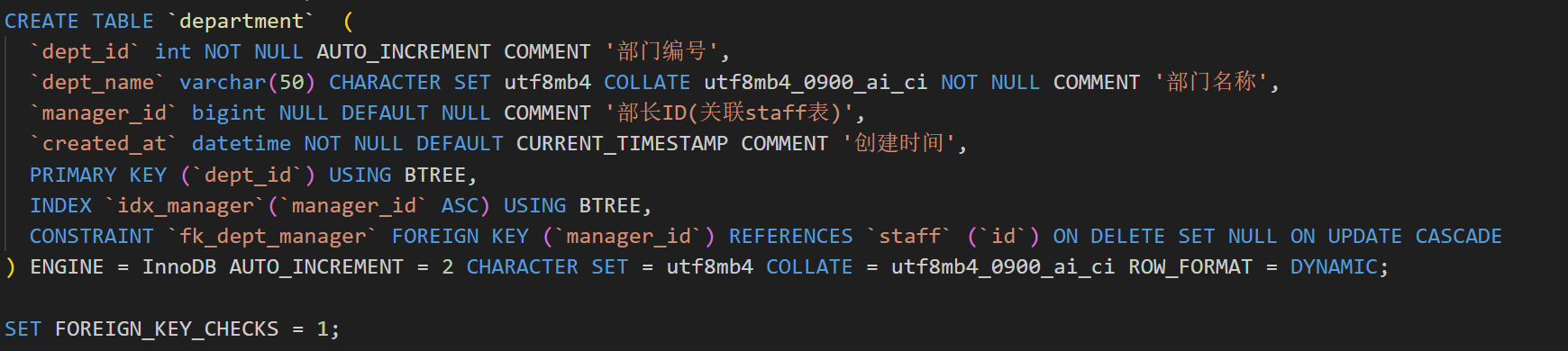


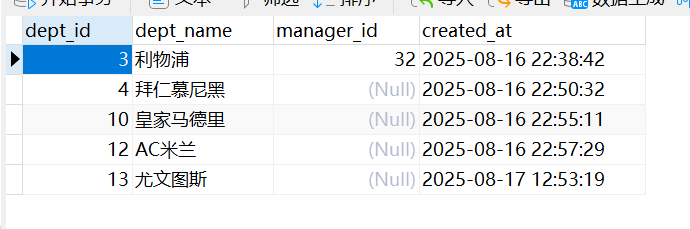
1. salary 表（薪资表）



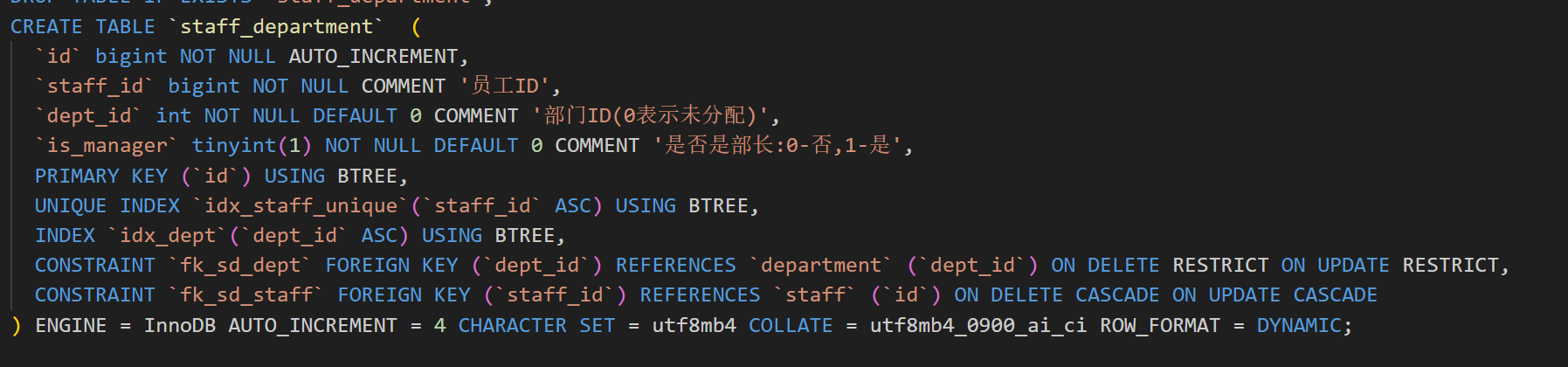


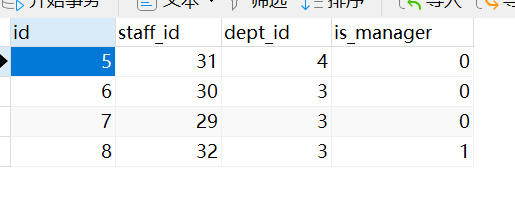
1. department 表（部门表）





1. staff\_department 表（员工部门关系表）





该系统为企业提供了完整的员工管理解决方案，从基础信息管理到复杂的业务流程处理，都能得到良好的支持。

1. 系统实现
   1. 系统文件结构
      1. 数据库API服务文件结构

该企业员工信息管理系统的后端数据库API服务文件结构清晰，主要围绕Spring Boot框架构建，遵循了经典的三层架构模式。核心文件包括位于 controller 目录下的各类 Controller 文件（如 `AdminController.java` , `EmployeeController.java`,`AttendanceController.java` ,`LeaveApplicationController.java` 和 `SalaryController.java` ），它们负责处理HTTP请求并调用相应的业务逻辑； entity 包（例如 Admin.java 、 Employee.java 等）定义了与数据库表对应的实体类，是数据传输和持久化的基础； repository 包（例如 AdminRepository.java 、 EmployeeRepository.java 等）则包含了数据访问接口，通过Spring Data JPA简化了数据库操作。此外， `SecurityConfig.java` 文件负责系统的安全配置，包括CORS设置和请求权限控制，确保API的安全性。数据库连接配置则集中在 application.properties 文件中，包含了数据库URL、用户名、密码等关键信息。整体而言，这种分层结构使得系统模块化、易于维护，并能有效支持各项业务功能的实现。如图5-1所示。



图 5‑1 数据库结构树状图

* + 1. 数据展示后端文件结构

整个后端结构清晰地展示了一个典型的 Spring Boot 应用的分层架构。 controller 层负责接收请求， entity 层定义数据模型， repository 层处理数据持久化，并通过 SecurityConfig 实现了全面的安全控制。 resources 目录则包含了应用程序的配置和数据库初始化脚本。这种结构使得代码模块化、职责分离，易于开发、维护和扩展。



后端框架结构图

* 1. 相关功能模块实现
     1. 数据库API服务相关功能实现

存该项目的后端数据库API服务模块是基于 Spring Boot 框架构建的，主要负责处理与数据库相关的业务逻辑和数据交互。它采用了经典的三层架构模式：

1. Controller 层 ：位于 `controller` 目录下，包含 `AdminController.java` 、 `EmployeeController.java` 等多个控制器。这些控制器接收来自前端的 HTTP 请求，解析请求参数，并调用 Service 层的方法来处理业务逻辑。它们是API的入口点，负责定义各种数据操作的RESTful接口。

Controller 层是 API 的入口点，负责接收前端的 HTTP 请求并处理业务逻辑。它通过 @RestController 和 @RequestMapping 定义 RESTful 接口。

例如， `AdminController.java` 中的注册接口：

@PostMapping("/register")

public Map<String, Object> register(@RequestBody Map<String, String> user) {

// ...

Admin admin = new Admin(username, password);

adminRepository.save(admin); // 数据存入数据库

// ...

}

该接口使用 POST 请求方式，通过 @RequestBody 接收前端发送的 JSON 数据（包含用户名和密码），然后调用 adminRepository.save(admin) 将数据存入数据库。类似地， `EmployeeController.java` 中的创建员工接口也采用 POST 方式，通过 employeeRepository.save(employee) 将员工信息存入数据库。其他接口如查询 ( @GetMapping )、更新 ( @PutMapping ) 和删除 ( @DeleteMapping ) 也都通过调用相应的 Repository 方法与数据库进行交互，实现数据的增删改查。



图5-1 获取员工信息API文档图

1. Entity 层 ：位于 `entity` 目录下，定义了与数据库表结构一一对应的 Java 实体类，如 `Admin.java` 、 `Employee.java` 、 `Department.java` 等。这些实体类是数据在应用程序内部的表示形式，通常与 JPA 或 MyBatis 等ORM框架配合使用，实现对象与关系数据库之间的映射。

例如， `Admin.java` 类使用 @Entity 声明为实体， @Id 标识主键：

@Entity

public class Admin {

@Id

private String username;

private String password;

// ...

}

又例如：`Employee.java` 类则展示了自增主键的配置：

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

// ...

}

此外，`Department.java` 类则更详细地展示了 @Column 用于指定列名、非空约束和默认值，以及 @Transient 用于标记不映射到数据库的字段：

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "dept\_id")

private Integer id;

@Column(name = "dept\_name", nullable = false)

private String name;

@Transient

private String managerName;

// ...

}

3. Repository 层 ：位于 `repository` 目录下，包含 `AdminRepository.java` 、 `EmployeeRepository.java` 等接口。这些接口通常是 Spring Data JPA 的抽象，通过继承 JpaRepository 等接口，无需编写具体的实现代码即可获得基本的CRUD（创建、读取、更新、删除）操作能力，极大地简化了数据访问层的开发。

例如， `AdminRepository.java` 接口继承了 JpaRepository<Admin, String> ，其中 Admin 是对应的实体类， String 是实体类主键的类型：

package com.example.back.repository;

import com.example.back.entity.Admin;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface AdminRepository extends JpaRepository<Admin, String> {

Admin findByUsername(String username);

}

除了 JpaRepository 提供的标准方法（如 save() , findById() , findAll() , delete() ），还可以通过定义方法签名来创建自定义查询，例如 findByUsername(String username) 。同样EmployeeRepository.java接口继承了 JpaRepository<Employee, Long> ，并定义了 findByUsername(String username) 方法：

package com.example.back.repository;

import com.example.back.entity.Employee;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

Employee findByUsername(String username);

}

这些接口使得开发者能够专注于业务逻辑，而无需关心底层数据库操作的实现细节。

通过这种分层设计，后端模块实现了业务逻辑与数据访问的解耦，提高了代码的可维护性、可测试性和扩展性。API服务通过RESTful接口对外提供数据服务，支持前端应用进行员工信息、部门、考勤、工资、请假等各类数据的管理和查询。



考勤查看图



个人工资记录



个人请假审核

4.Dto层：（Data Transfer Object）层在项目中主要用于封装数据，以便在不同层（如Controller层和Service层，或Service层和前端）之间进行数据传输。它的主要作用包括：

数据封装 ：将多个实体或部分实体数据封装成一个对象，方便一次性传输。

数据裁剪 ：只包含前端或特定业务逻辑所需的数据，避免暴露敏感信息或传输不必要的数据。

数据聚合 ：将来自不同实体的数据聚合到一个对象中，简化前端的数据处理。

papackage com.example.back.dto;

import java.util.Date;

public class AttendanceStatusDTO {

private Long staffId; // 员工ID，用于唯一标识员工

private String staffName; // 员工姓名

private Date checkDate; // 考勤日期

private Integer status; // 考勤状态，例如：1-正常，2-迟到，3-早退，4-缺勤，5-请假等

private String leaveReason; // 请假原因，如果考勤状态为请假，则记录具体原因

// 构造函数：无参构造函数，用于Spring或其他框架进行对象实例化

public AttendanceStatusDTO() {

}

// 构造函数：包含所有字段的构造函数，方便一次性初始化所有属性

public AttendanceStatusDTO(Long staffId, String staffName, Date checkDate, Integer status, String leaveReason) {

this.staffId = staffId;

this.staffName = staffName;

this.checkDate = checkDate;

this.status = status;

this.leaveReason = leaveReason;

}

// Getter 和 Setter 方法：提供对私有字段的访问和修改接口

// 获取员工ID

public Long getStaffId() {

return staffId;

}

// 设置员工ID

public void setStaffId(Long staffId) {

this.staffId = staffId;

}

// 获取员工姓名

public String getStaffName() {

return staffName;

}

// 设置员工姓名

public void setStaffName(String staffName) {

this.staffName = staffName;

}

// 获取考勤日期

public Date getCheckDate() {

return checkDate;

}

// 设置考勤日期

public void setCheckDate(Date checkDate) {

this.checkDate = checkDate;

}

// 获取考勤状态

public Integer getStatus() {

return status;

}

// 设置考勤状态

public void setStatus(Integer status) {

this.status = status;

}

// 获取请假原因

public String getLeaveReason() {

return leaveReason;

}

// 设置请假原因

public void setLeaveReason(String leaveReason) {

this.leaveReason = leaveReason;

}

}

* + 1. 数据展示后端相关功能实现

1. controller层:

该层主要是定义接口地址、请求方式，接收请求的携带的数据，将数据交给service层做进一步的处理。controller层无需关心业务具体的逻辑如何实现，只需要关注数据接收和数据返回。

pom.xml 文件中与 Controller 层直接相关的依赖主要是

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

这个依赖提供了构建 RESTful API 所需的核心功能，包括 Spring MVC、内嵌的 Tomcat 服务器等，是 Controller 层能够处理 HTTP 请求和响应的基础。此外， spring-boot-starter-validation 也支持 Controller 层进行数据校验。

application.properties 文件是 Spring Boot 的核心配置文件。它主要配置了数据库连接（MySQL）、JPA/Hibernate 的行为（如 ddl-auto=none 表示不自动创建表， show-sql=true 显示SQL），以及服务器端口（8081）。此外，还配置了字符编码和CORS（跨域资源共享）策略，允许来自 http://localhost:8080 的请求访问后端服务，这对于前端与后端交互至关重要。

server.port=8081

server.servlet.encoding.charset=UTF-8

server.servlet.encoding.force=true

server.servlet.encoding.force-response=true

spring.mvc.cors.enabled=true

spring.mvc.cors.allowed-origins=http://localhost:8080

spring.mvc.cors.allowed-methods=GET,POST,PUT,DELETE,OPTIONS

spring.mvc.cors.allowed-headers=\*

spring.mvc.cors.allow-credentials=true

spring.mvc.cors.max-age=3600

配置完成后，即可进入设计的前端界面：



controller层无需关心业务具体的逻辑如何实现，只需要关注数据接收和数据返回。例如：StaffController.java控制器主要负责员工的注册、登录、密码修改以及员工信息的查询。它通过注入 StaffRepository 、 DailyCheckinRepository 、 DepartmentRepository 、 StaffDepartmentRepository 和 AdminRepository 来实现数据操作。

关键功能和代码说明，方法为POST类型：

接收用户名和密码进行注册。会检查用户名是否已存在，如果不存在则创建新员工并保存。返回一个 Map 表示注册成功或失败及相应消息。

@PostMapping("/api/staff/register")

public Map<String, Object> register(@RequestBody Map<String, String> user) {

Map<String, Object> response = new HashMap<>();

String username = user.get("username");

String password = user.get("password");

if (username != null && !username.isEmpty() && password != null && !password.isEmpty()) {

if (staffRepository.findByUsername(username) != null) {

response.put("success", false);

response.put("message", "用户名已存在");

} else {

Staff staff = new Staff(username, password);

staffRepository.save(staff);

response.put("success", true);

response.put("message", "注册成功");

}

} else {

response.put("success", false);

response.put("message", "用户名和密码不能为空");

}

return response;

}

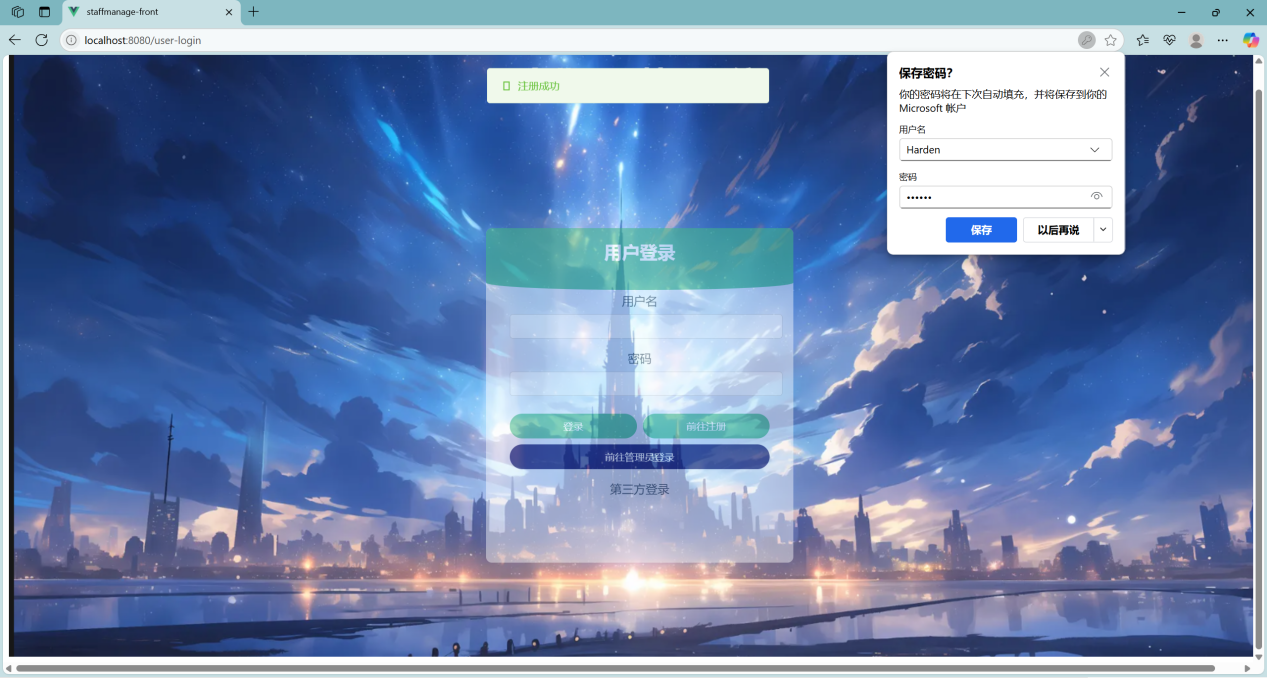


图5.2显示注册成功

1. Entity层

Entity 层主要负责定义数据模型，直接映射到数据库表，是数据展示的源头。它不直接涉及“数据展示后端相关功能实现”的逻辑，而是提供数据结构供 Repository 层进行数据操作，再由 Service 层和 Controller 层进行业务处理和数据展示。

以下是核心实体类的详细介绍：

1. Admin.java (管理员实体)

作用 ：定义了管理员的数据结构，包括用户名 ( username ) 和密码 ( password )。关键代码 ：@Entity

public class Admin {

@Id

private String username;

private String password;

……

}

2：Employee.java(员工实体)

作用 ：定义了员工的数据结构，包括 ID、姓名、性别、职位、用户名、密码和所属部门 ID。

关键代码 ：

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String gender;

private String position;

private String username;

private String password;

private Long departmentId;

……

}

3.Department.java`(部门实体)

作用 ：定义了部门的数据结构，包括部门 ID、部门名称、经理 ID 和创建时间。@Entity ：表示这是一个实体类。

@Id 和 @GeneratedValue ：同上，用于主键自增长。

@Column ：JPA 注解，用于指定字段映射的数据库列名、是否可为空等属性。

@Transient ：表示 managerName 字段不映射到数据库，它可能用于在业务逻辑中临时存储经理的姓名，而不是数据库中的 manager\_id 。

关键代码 ：

@Column(name = "dept\_name", nullable = false)

private String name;

@Column(name = "manager\_id")

private Long managerId;

@Column(name = "created\_at", nullable = false, updatable = false, columnDefinition = "datetime DEFAULT CURRENT\_TIMESTAMP")

private LocalDateTime createdAt;

@Transient

private String managerName;

// ... Getters 和 Setters ...

1. Respository层

Repository 层主要负责与数据库进行交互，提供数据访问接口。在这个项目中，Repository 接口通过继承 Spring Data JPA 的 JpaRepository 来实现对实体类的基本 CRUD（创建、读取、更新、删除）操作，并可以定义自定义查询方法。

以下是几个关键的 Repository 接口及其功能说明：

StaffRepository.java这个接口用于对 Staff 实体进行数据操作。它继承了 JpaRepository<Staff, Long> ，提供了针对 Staff 实体（主键类型为 Long ）的数据库操作方法。

package com.example.back.repository;

import com.example.back.entity.Staff;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface StaffRepository extends JpaRepository<Staff, Long> {

Staff findByUsername(String username);

}

通过该接口可以实现员工信息的显示功能，包括账号密码以及考勤数据：



员工信息显示图

DepartmentRepository.java这个接口用于对 Department 实体进行数据操作。它继承了 JpaRepository<Department, Integer> ，提供了针对 Department 实体（主键类型为 Integer ）的数据库操作方法。

package com.example.back.repository;

import com.example.back.entity.Department;

import org.springframework.data.jpa.repository.JpaRepository;

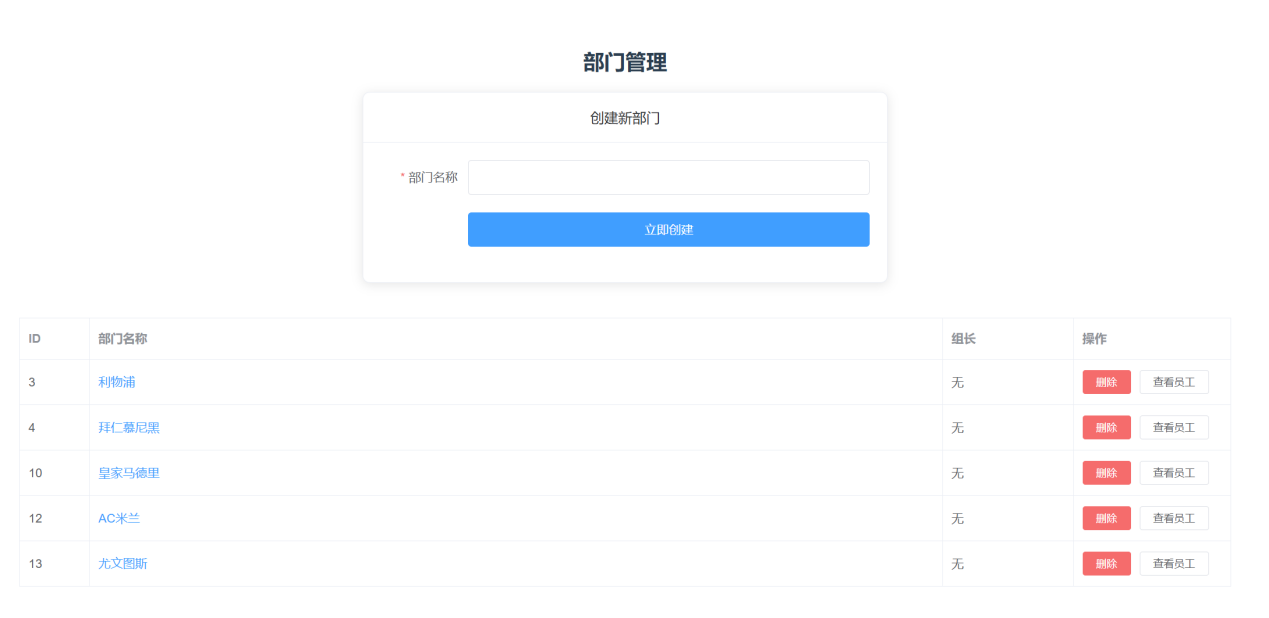
import java.util.Optional;

public interface DepartmentRepository extends JpaRepository<Department, Integer> {

Optional<Department> findByName(String name);

}

通过该接口可以实现部门信息的显示：



部门显示图

Repository 层通过 Spring Data JPA 极大地简化了数据访问层的开发。开发者只需定义接口并继承 JpaRepository ，Spring 就会在运行时自动实现这些接口，提供强大的数据操作能力。自定义查询方法则通过约定好的命名规则，让开发者能够轻松地实现复杂的查询逻辑，而无需编写 SQL 语句。

1. Dto层

DTO（Data Transfer Object）层在后端数据展示中扮演着重要角色，它主要用于在不同层之间传输数据，特别是从后端向前端传输数据。DTO 的设计目标是封装数据，而不是业务逻辑，通常只包含字段、构造函数、以及 getter/setter 方法。

* 1. 系统难点及实现
     1. 数据库迁移同步

企业员工信息管理系统在技术实现上存在多重难点。在后端，核心挑战在于构建一个既安全又高效的数据管理层。这包括实现精细化的数据安全和权限控制，例如通过 `SecurityConfig.java` 配置来区分管理员和普通员工对员工信息、部门、考勤和薪资等数据的访问权限，确保数据隔离和操作合规性。同时，处理考勤（ `daily\_checkin.sql` ）、薪资计算（ `salary.sql` ）和请假审批（ `leave\_application.sql` ）等复杂业务逻辑时，需要确保 `controller` 层、Service 层和 Repository 层之间的职责划分清晰，以保证代码的可维护性和可扩展性。随着数据量的增长，对 `staff.sql` 等核心表的频繁操作可能导致性能瓶颈，因此数据库查询优化和潜在的缓存机制引入是必要的。在前端，主要难点在于如何高效地展示和管理大量复杂数据，例如在 `App.vue` 、 `components` 和 `views` 中实现流畅的列表渲染、筛选、排序和分页功能，同时保证用户体验。此外，Vue.js 应用中的全局状态管理（可能需要 Vuex）以及基于用户角色的动态路由（ `router` ）和页面权限控制也是复杂且容易出错的环节。最后，前后端分离架构带来的接口联调、跨域问题以及部署策略的选择，也都是项目成功实施的关键技术难点。

1. 数据安全与权限控制

难点说明 ：在企业员工管理系统中，不同角色（如管理员、普通员工）对数据的访问和操作权限是不同的。实现精细化的权限控制，确保数据不被未授权访问或篡改，是核心难点。同时，用户认证（登录）和会话管理也是安全的关键。

代码示例与分析 ：

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.csrf().disable() // 禁用CSRF，因为我们使用JWT或Session来处理认证

.authorizeRequests()

.anyRequest().permitAll() // 允许所有请求，包括/login

.and()

.cors() // 启用CORS

.and()

.formLogin().disable() // 禁用表单登录

.httpBasic().disable() // 禁用HTTP Basic认证

.logout().disable(); // 禁用默认的注销功能

return http.build();

}

@Bean

public CorsConfigurationSource corsConfigurationSource() {

CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(Arrays.asList("http://localhost:8080")); // 允许来自http://localhost:8080的请求

configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS")); // 允许的HTTP方法

configuration.setAllowedHeaders(Arrays.asList("\*")); // 允许所有头

configuration.setAllowCredentials(true); // 允许发送凭证（如cookies）

configuration.setMaxAge(3600L); // 预检请求的缓存时间

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/\*\*", configuration); // 对所有路径应用CORS配置

return source;

}

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.csrf().disable() // 禁用CSRF，因为我们使用JWT或Session来处理认证

.authorizeRequests()

.anyRequest().permitAll() // 允许所有请求，包括/login

.and()

.cors() // 启用CORS

.and()

.formLogin().disable() // 禁用表单登录

.httpBasic().disable() // 禁用HTTP Basic认证

.logout().disable(); // 禁用默认的注销功能

return http.build();

}

@Bean

public CorsConfigurationSource corsConfigurationSource() {

CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(Arrays.asList("http://localhost:8080")); // 允许来自http://localhost:8080的请求

configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS")); // 允许的HTTP方法

configuration.setAllowedHeaders(Arrays.asList("\*")); // 允许所有头

configuration.setAllowCredentials(true); // 允许发送凭证（如cookies）

configuration.setMaxAge(3600L); // 预检请求的缓存时间

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/\*\*", configuration); // 对所有路径应用CORS配置

return source;

}

当前 <mcsymbol name="SecurityConfig.java"filename="SecurityConfig.java" path="d:\staffmanage\back\src\main\java\com\example\staffmanage\SecurityConfig.java" startline="1" type="class"></mcsymbol> 中的配置 `anyRequest().permitAll()` 意味着所有请求都被允许，这在实际生产环境中是极不安全的。真正的难点在于如何在此基础上实现基于角色的访问控制（RBAC）或基于资源的访问控制（ReBAC），例如只有管理员才能访问 `/api/admin` 下的某些接口，而普通员工只能访问自己的考勤和工资信息。这通常需要结合Spring Security的更高级功能，如 `hasRole()` 或自定义权限表达式。

`AdminController.java`\*\*：

```java:%2Fd%3A%2Fstaffmanage%2Fback%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2Fback%2Fcontroller%2FAdminController.java

@PostMapping("/login")

public Map<String, Object> login(@RequestBody Map<String, String> user) {

// ...

if (existingAdmin != null && existingAdmin.getPassword().equals(password)) {

response.put("success", true);

response.put("message", "登录成功");

response.put("username", username);

} else {

response.put("success", false);

response.put("message", "用户名或密码错误");

}

return response;

}

```

<mcsymbol name="AdminController.java" filename="AdminController.java" path="d:\staffmanage\back\src\main\java\com\example\back\controller\AdminController.java" startline="1" type="class"></mcsymbol> 中的登录逻辑直接比对明文密码 (`existingAdmin.getPassword().equals(password)`)，这是一个严重的安全漏洞。实际应用中，密码必须进行哈希加密存储（如使用 bcrypt），并在登录时比对哈希值。这是数据安全方面的一个重要难点。

2.复杂业务逻辑的API设计

难点说明：考勤、薪资、请假等业务逻辑往往涉及多表查询、复杂计算和状态流转。如何设计清晰、高效、可扩展的RESTful API来支撑这些业务，同时处理好各种边缘情况和数据一致性，是后端开发的核心挑战。

代码示例与分析：

AttendanceController.java：

java:%2Fd%3A%2Fstaffmanage%2Fback%2Fsrc%2Fmain%2Fjava%2Fcom%2Fexample%2Fback%2Fcontroller%2FAttendanceController.java

@GetMapping("/status/all/{date}")

public List<AttendanceStatusDTO> getAllEmployeesAttendanceStatusForDate(@PathVariable String date) {

List<AttendanceStatusDTO> statuses = new ArrayList<>();

try {

Date checkDate = dateFormat.parse(date);

// ... 获取所有员工ID的逻辑 ...

for (Long staffId : allStaffIds) {

Optional<Attendance> attendance = attendanceRepository.findById\_StaffIdAndId\_CheckDate(staffId, date);

if (attendance.isPresent()) {

// ... 处理已签到 ...

} else {

List<LeaveApplication> approvedLeaves = leaveApplicationRepository.findByStaffIdAndStatus(staffId, 1);

boolean onLeave = approvedLeaves.stream().anyMatch(leave ->

!checkDate.before(leave.getStartDate()) && !checkDate.after(leave.getEndDate())

);

if (onLeave) {

// ... 处理请假 ...

} else {

// ... 处理未签到 ...

}

}

}

} catch (ParseException e) {

// Handle parse exception

}

return statuses;

}

```

<mcsymbol name="AttendanceController.java" filename="AttendanceController.java" path="d:\staffmanage\back\src\main\java\com\example\back\controller\AttendanceController.java" startline="1" type="class"></mcsymbol> 中的 `getAllEmployeesAttendanceStatusForDate` 方法展示了考勤业务

* + 1. 项目单程调用

不本项目主要基于 Spring Boot 构建，核心依赖包括：

spring-boot-starter-data-jpa ：用于数据持久化，简化数据库操作。

spring-boot-starter-web ：构建 Web 应用程序，提供 RESTful API 支持。

mysql-connector-j ：MySQL 数据库连接驱动。

lombok ：简化 Java Bean 开发，减少样板代码。

spring-boot-starter-security ：提供安全认证和授权功能。

spring-boot-starter-validation ：用于数据校验。

这些依赖共同构成了后端服务的基础框架，支持数据访问、Web 接口、安全控制和数据验证等核心功能。本项目未发现微服务远程调用相关依赖。

1. pom.xml相关依赖

<parent> 部分 ：

spring-boot-starter-parent ：这是所有 Spring Boot 项目的推荐父 POM。它提供了 Maven 的默认配置，包括 Java 版本、编码、资源过滤等，并管理了许多常用依赖的版本，使得您无需手动指定每个依赖的版本，从而简化了依赖管理。

<groupId> , <artifactId> , <version> , <name> , <description> ：

这些是 Maven 项目的基本坐标信息，用于唯一标识您的项目。

<properties> 部分 ：

java.version ：定义了项目使用的 Java 版本，这里是 17 。

<dependencies> 部分 ：

spring-boot-starter-data-jpa ：引入 Spring Data JPA 及其相关依赖，用于简化数据库操作，支持 ORM（对象关系映射）。

spring-boot-starter-web ：引入 Spring Web MVC 及其相关依赖，用于构建 RESTful API 和 Web 应用程序。它包含了嵌入式 Tomcat 服务器。

mysql-connector-j ：MySQL 数据库的 JDBC 驱动，用于连接 MySQL 数据库。 scope 设置为 runtime 表示该依赖只在运行时需要。

lombok ：一个代码生成库，通过注解自动生成 Java Bean 的 getter/setter、构造函数等，减少样板代码。

spring-boot-starter-test ：用于测试的启动器，包含了 JUnit、Mockito、Spring Test 等测试框架和库， scope 设置为 test 表示只在测试阶段需要。

spring-boot-starter-security ：引入 Spring Security 及其相关依赖，用于实现应用程序的认证和授权功能。

spring-boot-starter-validation ：引入 Spring Boot 对 Java Bean Validation API 的支持，用于数据校验。

<build> 部分 ：

spring-boot-maven-plugin ：Spring Boot 提供的 Maven 插件，用于将 Spring Boot 应用程序打包成可执行的 JAR 或 WAR 文件。它还提供了运行应用程序、创建服务安装包等功能。

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>back</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>back</name>

<description>back</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

1. 启动类BackApplication.java

package com.example.back; ：定义了该类所属的包。

Import org.springframework.boot.SpringApplication; 和 import org.springframework.boot.autoconfigure.SpringBootApplication; ：导入了 Spring Boot 相关的核心类。

@SpringBootApplication(scanBasePackages = {"com.example.back"}) ：

这是一个组合注解，包含了 @Configuration 、 @EnableAutoConfiguration 和 @ComponentScan 。

@Configuration ：将该类标记为配置类。

@EnableAutoConfiguration ：启用 Spring Boot 的自动配置机制，根据 classpath 中的 jar 包依赖自动配置 Spring 应用程序。

@ComponentScan ：启用组件扫描，默认扫描当前包及其子包中的 Spring 组件（如 @Component , @Service , @Repository , @Controller 等）。这里显式指定了 scanBasePackages = {"com.example.back"} ，确保 Spring 扫描 com.example.back 包及其子包下的所有组件。

public class BackApplication { ... } ：定义了应用程序的主类。

public static void main(String[] args) { SpringApplication.run(BackApplication.class, args); } ：

这是 Java 应用程序的标准入口方法。

SpringApplication.run(BackApplication.class, args) ：这是启动 Spring Boot 应用程序的核心方法。它会创建一个 Spring 应用上下文，执行自动配置，并启动嵌入式 Web 服务器（如果存在）。

package com.example.back;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication(scanBasePackages = {"com.example.back"})

public class BackApplication {

public static void main(String[] args) {

SpringApplication.run(BackApplication.class, args);

}

}

1. 系统功能测试
   1. 系统测试环境

测试环境如下表6-1，表6-2所示：

表6-1服务器环境

| **类型** |  | **配置参数** |
| --- | --- | --- |
| 硬件 | 处理器  内存 | 12thGenIntel(R)Core(TM)i7-12700H 16GB |
|  | 硬盘 | 512GB |
|  | 带宽 | 6Mbps |

表6-2软件环境

| **类型** |  | **配置参数** |
| --- | --- | --- |
| 软件 | 编程语言  编程工具 | Java、vscode  Vue、ideal |
|  | 系统 | Windows |
|  | 其他工具 | 无 |

* 1. 数据库API服务功能测试

对数据库API进行测试，并在数据库中查看数据是否正确更新，测试结果如表6-3所示：

表6-3 数据库API功能测试表

| **测试项目** | **测试描述** | **测试结果** |
| --- | --- | --- |
| 判断是否能实现登录注册 | 通过POST表单提交用户名/密码，返回JWT/状态码。 | 成功 |
| 判断是否能够实现考勤 | 通常通过HTTP POST方法提交员工ID、日期和考勤状态等信息，返回JSON格式的数据，包含考勤操作是否成功、考勤记录的详细信息（如打卡时间、状态等）以及可能的错误消息。 | 成功 |
| 判断是否能够加入部门 | POST提交部门信息，返回JSON（成功/失败状态）。 | 成功 |
| 判断用户是否能够修改密码 | PUT提交新密码，返回JSON（修改结果）。 | 成功 |
| 判断管理员是否能够管理部门 | POST/PUT/DELETE提交部门信息，返回JSON（操作结果）。 | 成功 |

* 1. 数据展示后端功能测试

前后端对接完成后，通过在前端页面操作，来测试数据展示后端各个模块运行是否正常，测试结果如表6-4所示：

表6-4数据展示后端功能测试表

| **测试项目** | **测试描述** | **测试结果** |
| --- | --- | --- |
| **登录注册功能** | **查看注册是否能够正常存入数据库** | **成功** |
| **员工修改密码功能** | **用户是否能够正常修改密码** | **成功** |
| **部门管理功能** | **用户是否能够加入部门，管理员是否能够设置部长、建立新部门** | **成功** |
| **请假提交审核功能** | **用户是否能够提交请假表，管理员是否能够审核** | **成功** |
| **管理员发放工资及员工查询个人工资功能** | **管理员是否能正确发放工资，员工是否能够及时接收** | **成功** |

1. 总结与展望
   1. 总结

本系统成功实现了企业员工信息的数字化管理，提升了人事管理效率。通过前后端分离的架构，系统具备良好的可扩展性和可维护性。各项功能模块的实现满足了初期需求，为企业提供了便捷高效的员工管理工具。该项目是一个基于 Spring Boot 的单体应用，主要功能围绕员工管理展开，涵盖了员工、部门、考勤、薪资和请假等核心业务流程。从 pom.xml 和 BackApplication.java 的分析来看，项目采用了主流的 Java 后端技术栈，包括 Spring Boot 作为开发框架，Spring Data JPA 进行数据持久化，MySQL 作为数据库，以及 Spring Security 进行安全管理。项目结构清晰，依赖管理规范，是一个典型的企业级应用开发实践。目前未发现明确的微服务远程调用框架，表明其主要通过内部组件协作完成业务逻辑。

* 1. 展望

未来可以考虑增加移动端应用、集成第三方考勤系统、引入大数据分析进行人力资源预测等功能，进一步提升系统的智能化和实用性。该项目可以从多个方面进行扩展和优化。首先，可以考虑引入微服务架构，将不同业务模块拆分为独立服务，提高系统的可伸缩性和弹性，例如将考勤、薪资等模块独立部署。其次，可以集成消息队列实现异步通信，优化高并发场景下的性能。在前端方面，可以进一步完善用户体验，增加数据可视化报表。此外，引入自动化测试、持续集成/持续部署（CI/CD）流程，将有助于提升开发效率和代码质量。最后，可以探索集成 AI/ML 能力，例如基于历史数据进行员工绩效预测或智能排班，为企业管理提供更深层次的洞察。

参考文献

[1] 齐治昌, 谭火生, 宁连举. 软件工程[M]. 第4版. 北京: 高等教育出版社, 2018.

[2] Pressman R S, Maxim B R. Software Engineering: A Practitioner's Approach[M]. 9th ed. McGraw-Hill Education, 2019.

[3] 汪文君. Spring Boot实战派[M]. 北京: 电子工业出版社, 2020.

[4] 翟永超. Spring Cloud微服务实战[M]. 北京: 电子工业出版社, 2018.

[5]Spring Boot官方文档：https://docs.spring.io/spring-boot/docs/current/reference/html/

[6] 官方文档：MyBatis 3 User Guide. https://mybatis.org/mybatis-3/zh/index.html

[7] 官方文档：Spring Security Reference. https://docs.spring.io/spring-security/reference/

[8] 尤雨溪. Vue.js官方指南. https://cn.vuejs.org/guide/introduction.html

[9] Element Plus 官方文档. https://element-plus.org/zh-CN/

代码附录：