

项目名称：天气预报管理系统

林廷翰

指导教师：宋桂琴

日期：2023.10.21

目录

1. 项目概述 (Executive Summary)
2. 技术栈简介
3. 主要成果概览
4. 需求分析 (Requirement Analysis)
5. 功能需求
6. 启动须知
7. 技术架构 (Technical Architecture)
8. 数据库设计图
9. API接口
10. 功能演示

项目概述 (Executive Summary)

项目目的

本项目旨在开发一个高效、稳定且用户友好的天气预报管理系统。系统提供及时准确的天气信息服务，支持天气数据的查询、管理和预报功能。通过这个系统，用户可以获得实时天气更新，管理员可以管理天气数据，确保信息的准确性和时效性。此外，该系统也作为技术展示，展现了现代全栈技术在复杂应用开发中的应用。

技术栈简介

本项目采用了当前流行的技术栈，包括：

- Spring Boot:** 作为后端框架，简化了企业级应用开发，提供了快速开发的便利，同时也支持微服务架构的实现。
- Vue.js:** 前端框架，以数据驱动和组件化的思想，为用户提供了动态的界面交互体验。
- Redis:** 用作缓存系统，优化了数据处理速度和系统响应时间，提高了大数据量处理的效率。
- MySQL:** 作为关系型数据库，存储所有的天气数据和用户信息，确保数据的持久化，并且使用Mybatis plus交互。
- Nginx:** 用作反向代理服务器，提高了网站的加载速度和安全性，处理静态内容和负载均衡。

主要成果概览

- 提供了一个可靠的天气信息服务，能够处理并展示多源天气数据。
- 实现了一套完整的用户界面，包括天气信息的实时展示、历史数据查询和数据管理功能。
- 系统具备高性能的特点，通过Redis缓存减少了数据库的读取次数，Nginx的使用优化了静态资源的加载。

本系统不仅满足了用户对天气信息的需求，而且还提供了数据管理功能，为未来可的数据分析和预测提供了可能。

需求分析 (Requirement Analysis)

用户需求

用户需求聚焦于提供一个直观、易用的接口来访问天气信息，具体包括：

- 实时天气信息访问：**用户需要能够快速获取当前的天气状况，包括温度、湿度、风速等基本指标。
- 天气预报：**用户期望系统能提供至少一周的天气预测，包括降水概率和天气趋势。
- 多地点管理：**用户希望能管理多个地点的天气信息，特别是对于旅行者或多地点业务运营者来说尤为重要。

系统需求

系统需求关注于确保系统的稳定性、性能和可维护性，具体需求如下：

- 高可用性：**系统需要具备高可用性，能够处理高并发的用户访问，不受天气信息更新频率的影响。
- 数据准确性：**系统需确保提供的天气信息准确无误，包括实时数据和历史数据。
- 响应速度：**用户界面需要快速响应用户的查询请求，实时天气信息需要即时更新。
- 数据安全：**用户数据和天气信息的存储需要安全可靠，防止数据泄露或损坏。
- 可扩展性：**系统设计应考虑未来可能的功能扩展或技术升级。

功能需求

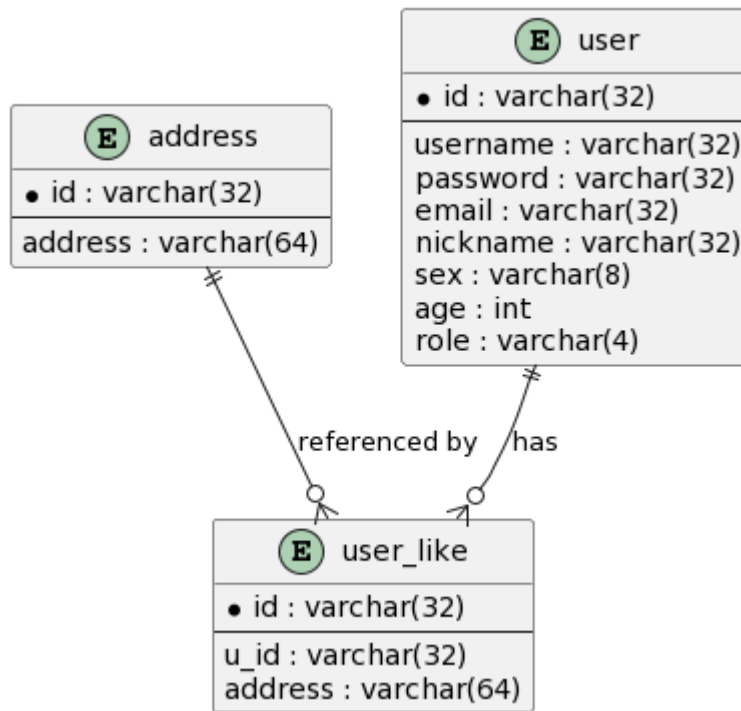
功能需求定义了系统的具体功能点，包括但不限于：

- 天气数据展示：**系统应能展示包括温度、湿度、风力等在内的天气数据。
- 用户界面：**系统需要有一个清晰、友好的用户界面，允许用户进行天气查询和设置个性化选项。
- 数据管理后台：**为管理员提供一个后台管理平台，用于管理天气数据源、更新频率和用户反馈。
- 日志记录：**系统应记录操作日志和系统日志，以便问题追踪和性能监控。
- 用户账户管理：**用户可以创建账户，自定义设置，如选择感兴趣的地点和接收通知的偏好。

技术架构 (Technical Architecture)

数据库设计图

- user** 表存储了用户的基本信息。
- address** 表包含了地址数据。
- user_like** 表创建了用户和地址之间的多对多关系，通过 `u_id` 和 `address` 字段与对应的表相联系。



```

1  create table address
2  (
3      id      varchar(32) not null
4          primary key,
5      address varchar(64) null
6  )
7
8      charset = utf8mb3
9      row_format = DYNAMIC;
10
11 create table user
12 (
13     id      varchar(32)          not null
14         primary key,
15     username varchar(32)          null,
16     password varchar(32)          null,
17     email    varchar(32)          null,
18     nickname varchar(32)          null,
19     sex      varchar(8)           null,
20     age      int                  null,
21     role     varchar(4) default '0' null comment '1'
22 )
23
24     charset = utf8mb3
25     row_format = DYNAMIC;
26
27 create table user_like
28 (
29     id      varchar(32) not null
30         primary key,
31     u_id    varchar(32) null,
32     address varchar(64) null
33 )
34
35     charset = utf8mb3
36     row_format = DYNAMIC;

```

API接口

UserController 的API接口：

- **用户认证**
 - POST /api/user/login - 用户登录，返回用户信息及其喜欢的信息。
- **用户信息管理**
 - POST /api/user/editPwd - 修改用户密码。
 - POST /api/user/editInfo - 编辑用户信息。
 - POST /api/user/add - 添加新用户。
- **用户喜好管理**
 - GET /api/user/like/{id} - 获取特定用户的喜好信息。
 - GET /api/user/addLike/{name}/{userId} - 为用户添加一个喜好。
 - GET /api/user/delLike/{name}/{userId} - 删除用户的一个喜好。
- **用户列表管理**
 - GET /api/user/getAll - 获取所有用户的信息。
- **用户删除**
 - GET /api/user/del - 删除一个用户（此接口似乎应该是@DeleteMapping和应包含@PathVariable注解）。
- **错误测试**
 - GET /api/user/error - 一个测试用的接口，用来模拟错误情况。

WeatherController 的API接口：

- **天气数据获取**
 - GET /api/weather/get - 获取特定城市的天气数据，默认为"广州"。
- **地区信息管理**
 - GET /api/weather/getAddress - 查询所有地区信息。
 - DELETE /api/weather/delAddress/{id} - 删除一个地区。
 - PUT /api/weather/putAddress - 修改地区信息。
 - POST /api/weather/addAddress - 增加新地区。

每个API提供了特定的功能，如用户的认证、信息管理、添加喜好、删除用户，以及天气数据和地区信息的管理。

功能演示

如何启动

1. 配置MySQL和Redis, 分别设置好端口和地址
2. 导入sql文件

```
1  SET NAMES utf8mb4;
2  SET FOREIGN_KEY_CHECKS = 0;
3
4  -----
5  -- Table structure for address
6  -----
7  DROP TABLE IF EXISTS `address`;
8  CREATE TABLE `address` (
9    `id` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
10   `address` varchar(64) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
11   PRIMARY KEY (`id`) USING BTREE
12 ) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Dynamic;
13
14  -----
15  -- Records of address
16  -----
17  INSERT INTO `address` VALUES ('153b8e3cdaa255fc82c67d71e1c4e78e', '武汉');
18  INSERT INTO `address` VALUES ('2', '天津');
19  INSERT INTO `address` VALUES ('d7163b083062488df5a12a1f8e17b775', '孝感');
20  INSERT INTO `address` VALUES ('e22def2dc29063c732516e07f9244f35', '北京');
21
22  -----
23  -- Table structure for user
24  -----
25  DROP TABLE IF EXISTS `user`;
26  CREATE TABLE `user` (
27    `id` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
28    `username` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
29    `password` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
30    `email` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
31    `nickname` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
32    `sex` varchar(8) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
33    `age` int(11) NULL DEFAULT NULL,
34    `role` varchar(4) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT '0'
COMMENT '1',
35    PRIMARY KEY (`id`) USING BTREE
36 ) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Dynamic;
37
38  -----
39  -- Records of user
40  -----
```

```

41 INSERT INTO `user` VALUES ('1', 'admin', 'admin', '3434565548@qq.com', '张二',
    '男', 22, '1');
42 INSERT INTO `user` VALUES ('1072bde7b194f190b5252e7d321ab175', 'wangwu', 'wangwu',
    '123@qq.com', '王五', '男', 18, '0');
43 INSERT INTO `user` VALUES ('2a5d884243189d07b3e8893caa01a3b4', 'lisi', '123123',
    '123123@qq.com', '李四', '男', 22, '1');
44 INSERT INTO `user` VALUES ('f459ecef231d26761658c41384625467', '123', '123',
    '123', '123', '123', 123, '0');
45
46 -----
47 -- Table structure for user_like
48 -----
49 DROP TABLE IF EXISTS `user_like`;
50 CREATE TABLE `user_like` (
51   `id` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
52   `u_id` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
53   `address` varchar(64) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
    NULL,
54   PRIMARY KEY (`id`) USING BTREE
55 ) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
    Dynamic;
56
57 -----
58 -- Records of user_like
59 -----
60 INSERT INTO `user_like` VALUES ('7af069aa7f64f33c9961106220d6cb9c',
    '1072bde7b194f190b5252e7d321ab175', '上海');
61 INSERT INTO `user_like` VALUES ('a77b57ff5eba78e64824ddd27a0b579d', '1', '北京');
62 INSERT INTO `user_like` VALUES ('b24047deec3381f51b2468a383460d3f',
    '1072bde7b194f190b5252e7d321ab175', '北京');
63 INSERT INTO `user_like` VALUES ('cfdde3091b50921bec4549af281ef759', '1', '天津');
64
65 SET FOREIGN_KEY_CHECKS = 1;
66

```

3. 在application.properties文件里面同步好配置，修改静态资源路径

```

1 # 静态资源
2 spring.resources.static-
    locations=file:C://Users/lth/Downloads/weather_hanger/weather/weather/src/main/reso
    urces/static/

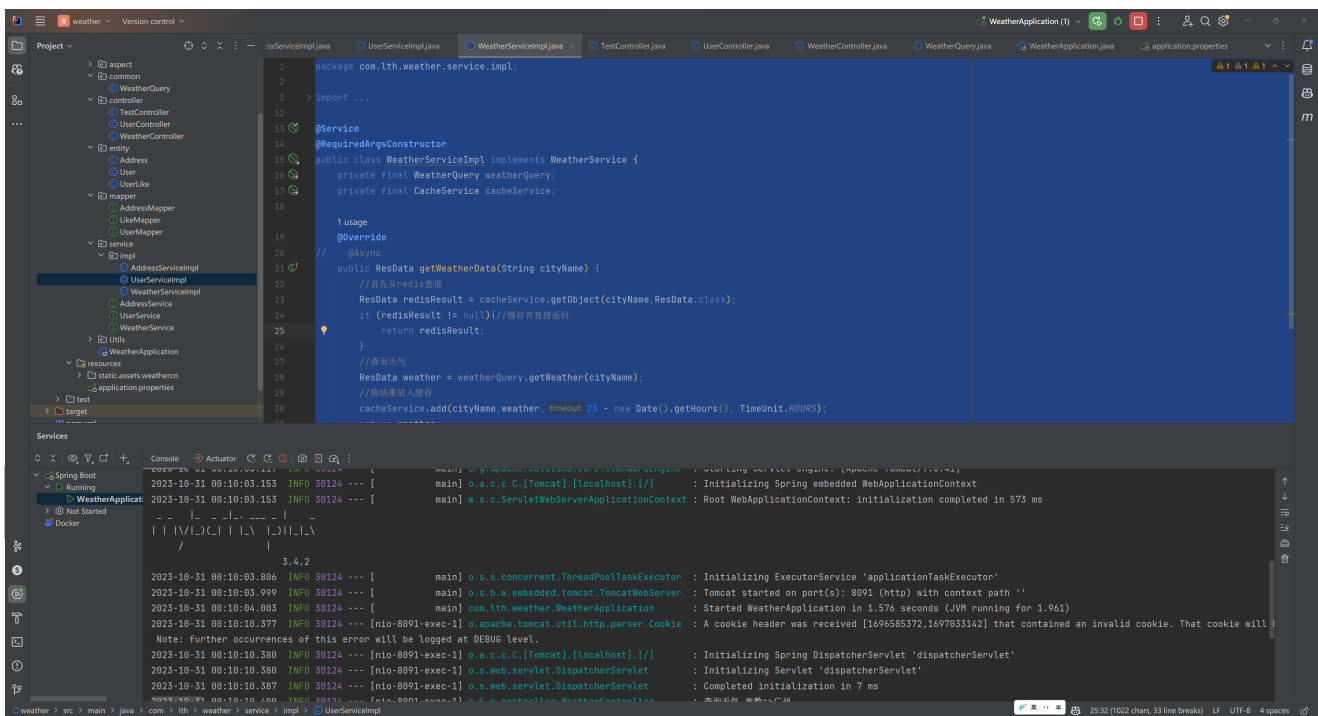
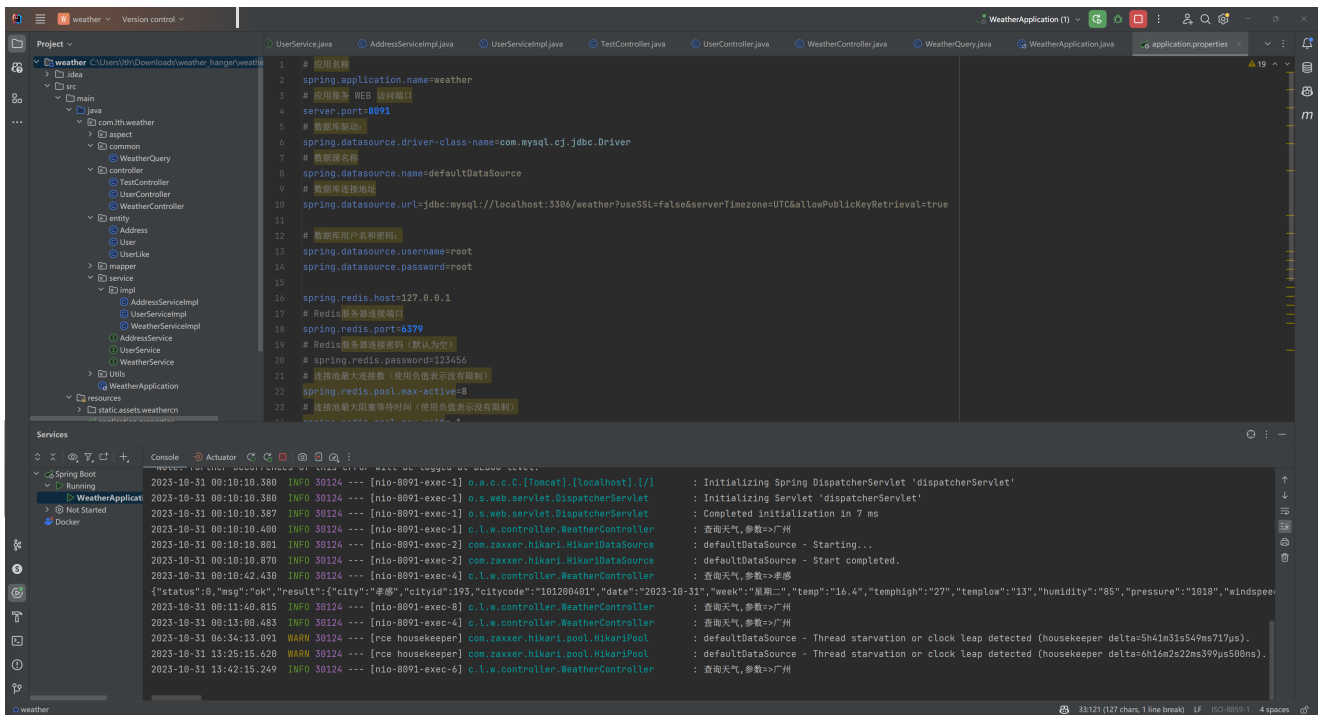
```

4. 启动nginx，在对应文件下面启动cmd窗口，使用命令start nginx.exe

5. 打开项目文件，启动项目，打开localhost:10088，将能访问到最后的页面


用户认证与管理

控制台输出实例：



1. 用户登录：



 admin



登录

[免费注册](#)

- 用户可以创建账户，需要输入基本信息，如用户名、密码、邮箱等。
- 注册信息通过前端Vue应用收集，然后通过Spring Boot后端进行处理。

个人中心



昵称:





[去登录](#)

3. 个人信息管理:

- 允许用户查看和更新个人信息。
- 实现CRUD（创建、读取、更新、删除）操作，通过Vue.js前端与用户交互。
- 后端Spring Boot处理数据的持久化到MySQL数据库。



张二

基本信息

ID 1...
账号 admin
邮箱 1914359773@qq.com
姓名 张二
性别 男
年龄 22

[修改密码](#)[退出登录](#)

4. 管理员管理：

- 管理员可以管理用户账户，如重置密码、修改用户权限、删除用户。
- 管理界面可以为管理员提供一个总览，显示系统内所有用户的状态。

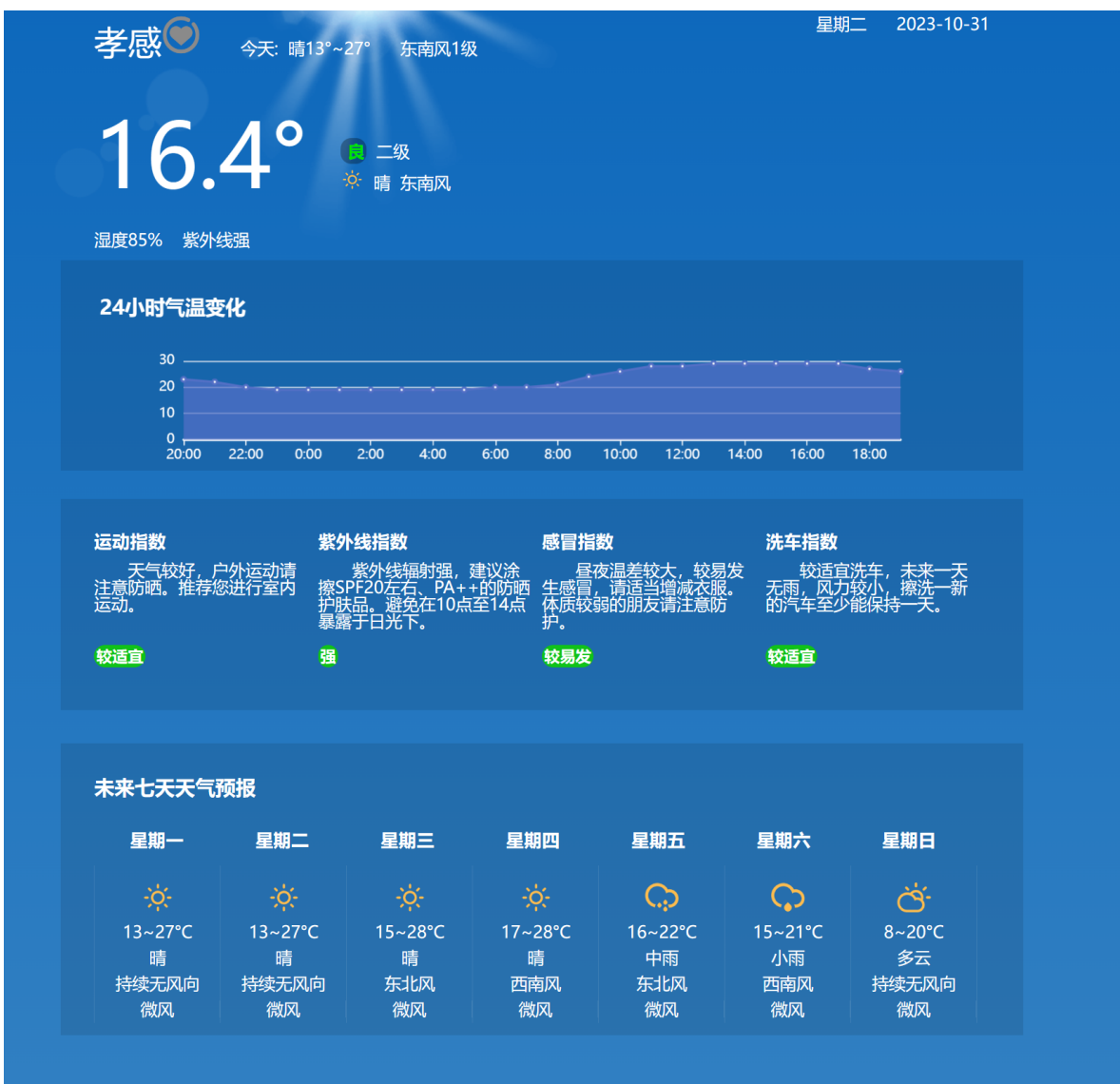
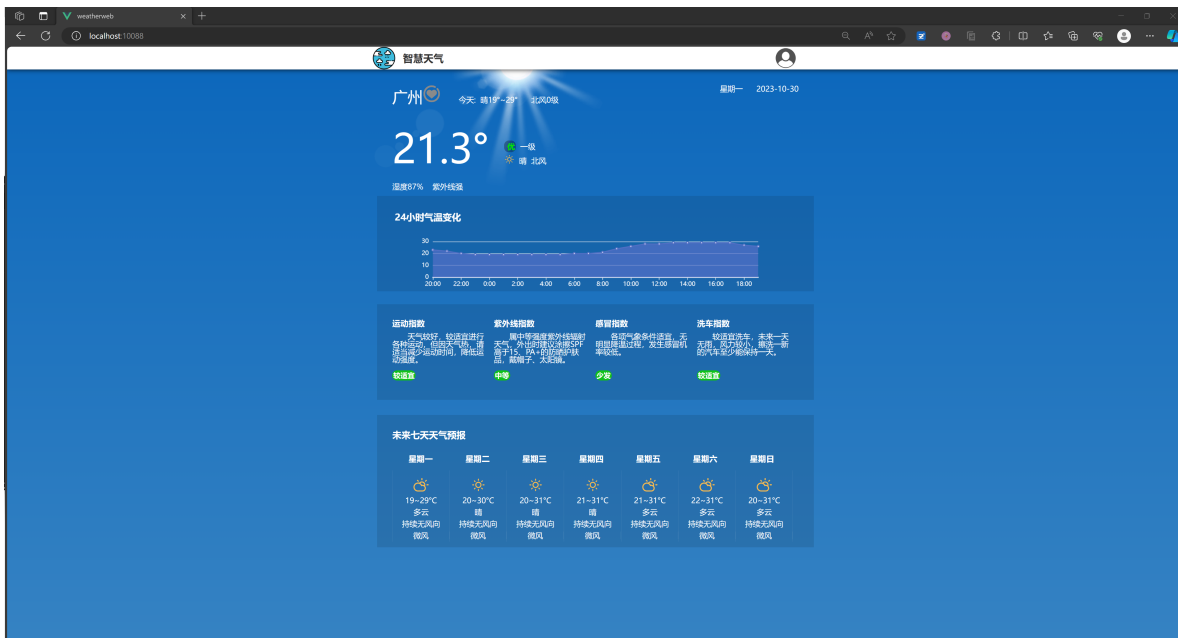
天气管理后台						
用户管理						
id	姓名	账号	密码	邮箱	操作	
1	张二	admin	admin	1914359773@qq.com	编辑	删除
1072bde7b194f1...	王五	wangwu	wangwu	123@qq.com	编辑	删除
2a5d884243189d...	李四	lisi	123123	123123@qq.com	编辑	删除
c01b1a33a16f95...	lth	hanger	hanger		编辑	删除
f459eece231d057...	123	123	123	123	编辑	删除

界面和交互

1. 基础界面显示：

- 显示天气查询入口、地区选择和个人账户管理等功能链接或按钮。

2.



3. 地区选择:

- 允许用户选择或搜索他们想要查询的地区。
- 可以实现一个下拉列表或自动完成功能以提供用户友好的体验。



天气信息功能

1. 天气查询:

- 用户选择地区后，系统通过调用第三方天气API来获取天气数据。
- 显示当前天气情况、预报和可能的天气预警。
- 通过Redis实现数据缓存，提高频繁查询的响应速度。



service层

```
1 package com.lth.weather.service.impl;
2
3 import com.lth.weather.service.WeatherService;
4 import com.lth.weather.Utils.CacheService;
5 import com.lth.weather.Utils.ResData;
6 import com.lth.weather.common.WeatherQuery;
7 import lombok.RequiredArgsConstructor;
8 import org.springframework.stereotype.Service;
9
10 import java.util.Date;
11 import java.util.concurrent.TimeUnit;
12
13 @Service
14 @RequiredArgsConstructor
15 public class WeatherServiceImpl implements WeatherService {
```

```

16     private final WeatherQuery weatherQuery;
17     private final CacheService cacheService;
18
19     @Override
20     // @Async
21     public ResData getWeatherData(String cityName) {
22         //首先从redis查询
23         ResData redisResult = cacheService.getObject(cityName, ResData.class);
24         if (redisResult != null){//缓存有直接返回
25             return redisResult;
26         }
27         //查询天气
28         ResData weather = weatherQuery.getWeather(cityName);
29         //将结果放入缓存
30         cacheService.add(cityName, weather, 23 - new Date().getHours(), TimeUnit.HOURS);
31         return weather;
32     }
33 }
34

```

核心技术：调用阿里云的api商店来查询天气，统一封装返回对象。

```

1 package com.lth.weather.common;
2
3 import com.lth.weather.Utills.HttpUtils;
4 import com.lth.weather.Utills.ResData;
5 import org.apache.http.HttpResponse;
6 import org.apache.http.util.EntityUtils;
7 import org.springframework.stereotype.Component;
8
9 import java.util.Collections;
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13
14 @Component
15 public class WeatherQuery {
16
17     public ResData getWeather(String cityName){
18         String host = "https://jisutqybmf.market.alicloudapi.com";
19         String path = "/weather/query";
20         String method = "GET";//GET/POST 任意
21         String appcode = "34f91635b9ef40adb77196230317afae";
22         Map<String, String> headers = new HashMap<String, String>();
23         //最后在header中的格式(中间是英文空格)为Authorization:APPCODE
24         83359fd73fe94948385f570e3c139105
25         headers.put("Authorization", "APPCODE " + appcode);
26         //根据API的要求，定义相对应的Content-Type
27         headers.put("Content-Type", "application/json; charset=UTF-8");
28         Map<String, String> querys = new HashMap<String, String>();
29         querys.put("city", cityName);
30         /*querys.put("citycode", "citycode");
31         querys.put("cityid", "cityid");
32         querys.put("ip", "ip");

```

```

32         querys.put("location", "location");*/
33
34
35     try {
36         HttpResponse response = HttpUtils.doGet(host, path, method, headers, querys);
37         /*System.out.println(response.toString());
38         获取response的body
39         System.out.println(EntityUtils.toString(response.getEntity(), "utf-8"));*/
40         List<Object> entity =
Collections.singletonList(EntityUtils.toString(response.getEntity(), "utf-8"));
41         entity.forEach(System.out::println);
42
43         return ResData.ok(entity);
44     } catch (Exception e) {
45         e.printStackTrace();
46         return ResData.err();
47     }
48 }
49 }

```

通过集成阿里云的API商店，项目能够利用阿里云提供的强大基础设施和稳定的天气服务接口，确保了天气信息的准确性和及时性。

此外，该设计采用了合理的调用频率和缓存策略，能够有效地控制成本，因为从API商店获取数据通常是有限额和收费的。通过缓存最常请求的天气信息到Redis，减少了对阿里云API的频繁调用，从而降低了费用。

考虑到，在获取天气数据时可能会遇到各种异常情况，该设计通过异常捕获和日志记录机制，提供了强大的错误诊断和处理能力。返回标准化的错误信息，有利于前端进行相应的异常处理，提升了用户体验。

项目总结

这个项目经历了从需求分析、设计、开发到测试的完整过程，最终成功地构建了一个高效、稳定且用户友好的天气预报管理系统。

通过这个项目，我不仅为用户提供了一个直观、易用的接口来访问天气信息，还实现了高性能的特点，减少了数据库的读取次数，优化了静态资源的加载。这些成果都是我精心策划和辛勤努力所得到的。

在需求分析阶段，深入了解了用户的需求，发现用户需要一个能够实时获取天气信息、查看天气预报、查询历史数据以及管理多个地点的天气预报管理系统。为了满足这些需求，我采用了先进的技术栈，包括Spring Boot、Vue.js、Redis和MySQL等，这些技术使得系统开发更为便捷，数据处理更加高效，用户体验更加优秀。