package com.example.student.common.aop;

import com.example.student.common.utils.LogUtils;

import com.fasterxml.jackson.databind.ObjectMapper;

import jakarta.servlet.http.HttpServletRequest;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.\*;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.web.context.request.RequestContextHolder;

import org.springframework.web.context.request.ServletRequestAttributes;

import java.lang.reflect.Method;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;

@Aspect

@Component

public class LogAspect {

private static final Logger log = LoggerFactory.getLogger(LogAspect.class);

private static final ThreadLocal<String> TRACE\_ID = new ThreadLocal<>();

private static final ThreadLocal<Long> START\_TIME = new ThreadLocal<>();

@Autowired

private ObjectMapper objectMapper;

@Pointcut("execution(\* com.example.student.controller..\*.\*(..))")

public void controllerLog() {

}

@Before("controllerLog()")

public void doBefore(JoinPoint joinPoint) {

try {

TRACE\_ID.set(UUID.randomUUID().toString().replace("-", ""));

START\_TIME.set(System.currentTimeMillis());

ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();

if (attributes != null) {

HttpServletRequest request = attributes.getRequest();

Map<String, Object> requestInfo = new HashMap<>(8);

requestInfo.put("traceId", TRACE\_ID.get());

requestInfo.put("url", request.getRequestURL().toString());

requestInfo.put("method", request.getMethod());

requestInfo.put("ip", getClientIp(request));

requestInfo.put("class", joinPoint.getSignature().getDeclaringTypeName());

requestInfo.put("function", joinPoint.getSignature().getName());

requestInfo.put("args", Arrays.toString(joinPoint.getArgs()));

LogUtils.info(log, "接口请求: {}", objectMapper.writeValueAsString(requestInfo));

}

} catch (Exception e) {

LogUtils.error(log, "记录请求日志异常", e);

}

}

@Around("controllerLog()")

public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

try {

Object result = proceedingJoinPoint.proceed();

long endTime = System.currentTimeMillis();

long startTime = START\_TIME.get();

long executeTime = endTime - startTime;

if (executeTime > 500) {

LogUtils.warn(log, "接口 [{}] 执行时间: {}ms, 请检查性能问题",

proceedingJoinPoint.getSignature().getName(), executeTime);

} else {

LogUtils.info(log, "接口 [{}] 执行时间: {}ms",

proceedingJoinPoint.getSignature().getName(), executeTime);

}

return result;

} finally {

START\_TIME.remove();

}

}

@AfterReturning(returning = "result", pointcut = "controllerLog()")

public void doAfterReturning(Object result) {

try {

Map<String, Object> responseInfo = new HashMap<>(4);

responseInfo.put("traceId", TRACE\_ID.get());

responseInfo.put("result", result);

LogUtils.info(log, "接口响应: {}", objectMapper.writeValueAsString(responseInfo));

} catch (Exception e) {

LogUtils.error(log, "记录响应日志异常", e);

} finally {

TRACE\_ID.remove();

}

}

@AfterThrowing(pointcut = "controllerLog()", throwing = "e")

public void doAfterThrowing(Throwable e) {

LogUtils.error(log, "接口异常 [traceId:{}]: {}", TRACE\_ID.get(), e.getMessage(), e);

TRACE\_ID.remove();

START\_TIME.remove();

}

private String getClientIp(HttpServletRequest request) {

String ip = request.getHeader("X-Forwarded-For");

if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {

ip = request.getHeader("Proxy-Client-IP");

}

if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {

ip = request.getHeader("WL-Proxy-Client-IP");

}

if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {

ip = request.getHeader("HTTP\_CLIENT\_IP");

}

if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {

ip = request.getHeader("HTTP\_X\_FORWARDED\_FOR");

}

if (ip == null || ip.isEmpty() || "unknown".equalsIgnoreCase(ip)) {

ip = request.getRemoteAddr();

}

return ip;

}

}

package com.example.student.common.exception;

import com.example.student.common.ResultCode;

import lombok.Data;

import lombok.EqualsAndHashCode;

@Data

@EqualsAndHashCode(callSuper = true)

public class BusinessException extends RuntimeException {

private static final long serialVersionUID = 1L;

private Integer code;

private String message;

public BusinessException(String message) {

super(message);

this.code = ResultCode.ERROR.getCode();

this.message = message;

}

public BusinessException(Integer code, String message) {

super(message);

this.code = code;

this.message = message;

}

public BusinessException(ResultCode resultCode) {

super(resultCode.getMessage());

this.code = resultCode.getCode();

this.message = resultCode.getMessage();

}

public BusinessException(ResultCode resultCode, String message) {

super(message);

this.code = resultCode.getCode();

this.message = message;

}

public Integer getCode() {

return code;

}

@Override

public String getMessage() {

return message;

}

}

package com.example.student.common.exception;

import com.example.student.common.utils.LogUtils;

import com.example.student.common.Result;

import com.example.student.common.ResultCode;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.dao.DuplicateKeyException;

import org.springframework.validation.BindException;

import org.springframework.validation.FieldError;

import org.springframework.web.HttpRequestMethodNotSupportedException;

import org.springframework.web.bind.MethodArgumentNotValidException;

import org.springframework.web.bind.MissingServletRequestParameterException;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.RestControllerAdvice;

import org.springframework.web.method.annotation.MethodArgumentTypeMismatchException;

import java.nio.file.AccessDeniedException;

import java.sql.SQLSyntaxErrorException;

import java.util.List;

@RestControllerAdvice

public class CommonGlobalExceptionHandler {

private static final Logger log = LoggerFactory.getLogger(CommonGlobalExceptionHandler.class);

@ExceptionHandler(BusinessException.class)

public Result<?> handleBusinessException(BusinessException e) {

LogUtils.error(log, "业务异常: {}", e.getMessage());

return Result.error(e.getCode(), e.getMessage());

}

@ExceptionHandler(MethodArgumentNotValidException.class)

public Result<?> handleMethodArgumentNotValidException(MethodArgumentNotValidException e) {

List<FieldError> fieldErrors = e.getBindingResult().getFieldErrors();

StringBuilder errorMsg = new StringBuilder();

for (FieldError error : fieldErrors) {

errorMsg.append(error.getField()).append(": ").append(error.getDefaultMessage()).append(", ");

}

String message = errorMsg.toString();

LogUtils.warn(log, "参数校验异常: {}", message);

return Result.error(message);

}

@ExceptionHandler(BindException.class)

public Result<?> handleBindException(BindException e) {

StringBuilder errorMsg = new StringBuilder();

List<FieldError> fieldErrors = e.getBindingResult().getFieldErrors();

for (FieldError error : fieldErrors) {

errorMsg.append(error.getField()).append(": ").append(error.getDefaultMessage()).append(", ");

}

String message = errorMsg.toString();

LogUtils.warn(log, "绑定异常: {}", message);

return Result.error(message);

}

@ExceptionHandler(MethodArgumentTypeMismatchException.class)

public Result<?> handleMethodArgumentTypeMismatchException(MethodArgumentTypeMismatchException e) {

String message = "参数类型不匹配: " + e.getName() + " 应该为 " + e.getRequiredType().getName();

LogUtils.warn(log, message);

return Result.error(message);

}

@ExceptionHandler(MissingServletRequestParameterException.class)

public Result<?> handleMissingServletRequestParameterException(MissingServletRequestParameterException e) {

String message = "缺少必要参数: " + e.getParameterName();

LogUtils.warn(log, message);

return Result.error(message);

}

@ExceptionHandler(HttpRequestMethodNotSupportedException.class)

public Result<?> handleHttpRequestMethodNotSupportedException(HttpRequestMethodNotSupportedException e) {

String message = "不支持的请求方法: " + e.getMethod();

LogUtils.warn(log, message);

return Result.error(message);

}

@ExceptionHandler(AccessDeniedException.class)

public Result<?> handleAccessDeniedException(AccessDeniedException e) {

String message = "没有访问权限: " + e.getMessage();

LogUtils.warn(log, message);

return Result.error(403, message);

}

@ExceptionHandler(DuplicateKeyException.class)

public Result<?> handleDuplicateKeyException(DuplicateKeyException e) {

String message = "数据已存在，请勿重复添加";

LogUtils.error(log, "唯一键冲突: {}", e.getMessage());

return Result.error(message);

}

@ExceptionHandler(SQLSyntaxErrorException.class)

public Result<?> handleSQLSyntaxErrorException(SQLSyntaxErrorException e) {

String message = "SQL语法错误";

LogUtils.error(log, "SQL语法错误: {}", e.getMessage());

return Result.error(message);

}

@ExceptionHandler(Exception.class)

public Result<?> handleException(Exception e) {

LogUtils.error(log, "系统异常: ", e);

return Result.error("系统异常，请联系管理员");

}

}

package com.example.student.common;

import com.baomidou.mybatisplus.core.metadata.IPage;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

import java.util.List;

public class CommonPage<T> {

private Long pageNum;

private Long pageSize;

private Long totalPage;

private Long total;

private List<T> list;

public static <T> CommonPage<T> restPage(IPage<T> pageResult) {

CommonPage<T> result = new CommonPage<T>();

result.setPageNum(pageResult.getCurrent());

result.setPageSize(pageResult.getSize());

result.setTotal(pageResult.getTotal());

result.setTotalPage(pageResult.getPages());

result.setList(pageResult.getRecords());

return result;

}

public static <T> CommonPage<T> restPage(Page<T> pageResult) {

CommonPage<T> result = new CommonPage<T>();

result.setPageNum(pageResult.getCurrent());

result.setPageSize(pageResult.getSize());

result.setTotal(pageResult.getTotal());

result.setTotalPage(pageResult.getPages());

result.setList(pageResult.getRecords());

return result;

}

public Long getPageNum() {

return pageNum;

}

public void setPageNum(Long pageNum) {

this.pageNum = pageNum;

}

public Long getPageSize() {

return pageSize;

}

public void setPageSize(Long pageSize) {

this.pageSize = pageSize;

}

public Long getTotalPage() {

return totalPage;

}

public void setTotalPage(Long totalPage) {

this.totalPage = totalPage;

}

public Long getTotal() {

return total;

}

public void setTotal(Long total) {

this.total = total;

}

public List<T> getList() {

return list;

}

public void setList(List<T> list) {

this.list = list;

}

}

package com.example.student.common;

import java.io.Serializable;

public class CommonResult<T> implements Serializable {

private static final long serialVersionUID = 1L;

private Integer code;

private String message;

private T data;

protected CommonResult() {

}

protected CommonResult(Integer code, String message, T data) {

this.code = code;

this.message = message;

this.data = data;

}

public static <T> CommonResult<T> success(T data) {

return new CommonResult<>(200, "操作成功", data);

}

public static <T> CommonResult<T> success(T data, String message) {

return new CommonResult<>(200, message, data);

}

public static <T> CommonResult<T> failed(String message) {

return new CommonResult<>(500, message, null);

}

public static <T> CommonResult<T> validateFailed(String message) {

return new CommonResult<>(400, message, null);

}

public static <T> CommonResult<T> unauthorized() {

return new CommonResult<>(401, "暂未登录或token已经过期", null);

}

public static <T> CommonResult<T> forbidden() {

return new CommonResult<>(403, "没有相关权限", null);

}

public Integer getCode() {

return code;

}

public void setCode(Integer code) {

this.code = code;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

}

package com.example.student.common;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.Max;

import jakarta.validation.constraints.Min;

import lombok.Data;

@Data

@Schema(description = "分页请求参数")

public class PageRequest {

@Schema(description = "页码", example = "1")

@Min(value = 1, message = "页码最小为1")

private Integer pageNum = 1;

@Schema(description = "每页数量", example = "10")

@Min(value = 1, message = "每页数量最小为1")

@Max(value = 1000, message = "每页数量最大为1000")

private Integer pageSize = 10;

}

package com.example.student.common;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.util.Collections;

import java.util.List;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Schema(description = "分页结果")

public class PageResult<T> {

@Schema(description = "数据列表")

private List<T> rows;

@Schema(description = "总记录数")

private long total;

@Schema(description = "当前页码")

private int pageNum;

@Schema(description = "每页大小")

private int pageSize;

public PageResult(List<T> rows, long total) {

this.rows = rows;

this.total = total;

}

public static <T> PageResult<T> build(List<T> list, long total, int pageNum, int pageSize) {

return new PageResult<T>(list, total, pageNum, pageSize);

}

public static <T> PageResult<T> empty() {

return new PageResult<T>(Collections.emptyList(), 0L, 1, 10);

}

public static <T> PageResult<T> empty(Class<T> clazz) {

return new PageResult<T>(Collections.emptyList(), 0L, 1, 10);

}

public List<T> getList() {

return this.rows;

}

}

package com.example.student.common;

import com.fasterxml.jackson.annotation.JsonInclude;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import java.io.Serializable;

@Data

@JsonInclude(JsonInclude.Include.NON\_NULL)

@Schema(description = "统一返回结果")

public class Result<T> implements Serializable {

private static final long serialVersionUID = 1L;

@Schema(description = "状态码")

private Integer code;

@Schema(description = "消息")

private String message;

@Schema(description = "数据")

private T data;

private Result() {

}

public static <T> Result<T> success() {

return success("操作成功");

}

public static <T> Result<T> success(String message) {

return success(message, null);

}

public static <T> Result<T> success(T data) {

return success("操作成功", data);

}

public static <T> Result<T> success(String message, T data) {

Result<T> result = new Result<>();

result.setCode(ResultCode.SUCCESS.getCode());

result.setMessage(message);

result.setData(data);

return result;

}

public static <T> Result<T> error() {

return error("操作失败");

}

public static <T> Result<T> error(String message) {

return error(ResultCode.ERROR.getCode(), message);

}

public static <T> Result<T> error(Integer code, String message) {

Result<T> result = new Result<>();

result.setCode(code);

result.setMessage(message);

return result;

}

public static <T> Result<T> error(ResultCode resultCode) {

return error(resultCode.getCode(), resultCode.getMessage());

}

public static <T> Result<T> error(ResultCode resultCode, String message) {

return error(resultCode.getCode(), message);

}

}

package com.example.student.common;

import lombok.Getter;

@Getter

public enum ResultCode {

SUCCESS(200, "操作成功"),

ERROR(500, "操作失败"),

VALIDATE\_FAILED(400, "参数检验失败"),

UNAUTHORIZED(401, "未登录或token已过期"),

FORBIDDEN(403, "没有相关权限");

private final Integer code;

private final String message;

ResultCode(Integer code, String message) {

this.code = code;

this.message = message;

}

}

package com.example.student.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.cors.CorsConfiguration;

import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import org.springframework.web.filter.CorsFilter;

@Configuration

public class CorsConfig {

@Bean

public CorsFilter corsFilter() {

CorsConfiguration config = new CorsConfiguration();

config.addAllowedOriginPattern("\*");

config.addAllowedHeader("\*");

config.addAllowedMethod("\*");

config.setAllowCredentials(true);

config.addExposedHeader("\*");

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("

@Data

@Component

@ConfigurationProperties(prefix = "jwt")

public class JwtConfig {

private String secret;

private Long expire;

private String header;

private String tokenPrefix;

}

package com.example.student.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.io.Resource;

import org.springframework.core.io.ResourceLoader;

import org.springframework.beans.factory.annotation.Autowired;

import io.swagger.v3.oas.models.Components;

import io.swagger.v3.oas.models.OpenAPI;

import io.swagger.v3.oas.models.info.Contact;

import io.swagger.v3.oas.models.info.Info;

import io.swagger.v3.oas.models.info.License;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.nio.charset.StandardCharsets;

import java.util.stream.Collectors;

@Configuration

public class Knife4jConfig {

@Autowired

private ResourceLoader resourceLoader;

private String readResourceContent(Resource resource) throws IOException {

try (InputStream inputStream = resource.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream, StandardCharsets.UTF\_8))) {

return reader.lines().collect(Collectors.joining("\n"));

}

}

@Bean

public OpenAPI customOpenAPI() {

return new OpenAPI()

.components(new Components())

.info(new Info()

.title("学生信息管理系统API文档")

.description("学生信息管理系统接口文档")

.version("1.0.0")

.contact(new Contact().name("管理员").email("admin@example.com"))

.license(new License().name("Apache 2.0").url("https:

);

}

}

package com.example.student.config;

import com.baomidou.mybatisplus.annotation.DbType;

import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;

import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;

import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;

import org.apache.ibatis.reflection.MetaObject;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.time.LocalDateTime;

@Configuration

public class MybatisPlusConfig {

@Bean

public MybatisPlusInterceptor mybatisPlusInterceptor() {

MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();

interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.MYSQL));

return interceptor;

}

@Bean

public MetaObjectHandler metaObjectHandler() {

return new MetaObjectHandler() {

@Override

public void insertFill(MetaObject metaObject) {

LocalDateTime now = LocalDateTime.now();

this.strictInsertFill(metaObject, "createTime", LocalDateTime.class, now);

this.strictInsertFill(metaObject, "updateTime", LocalDateTime.class, now);

}

@Override

public void updateFill(MetaObject metaObject) {

this.strictUpdateFill(metaObject, "updateTime", LocalDateTime.class, LocalDateTime.now());

}

};

}

}

package com.example.student.config;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.example.student.entity.SysRoleMenu;

import com.example.student.mapper.SysRoleMenuMapper;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

@Component

@RequiredArgsConstructor

@Slf4j

public class RolePermissionInitializer implements CommandLineRunner {

private final SysRoleMenuMapper sysRoleMenuMapper;

@Override

@Transactional

public void run(String... args) throws Exception {

log.info("开始初始化角色权限...");

initStudentSelectionPermissions();

log.info("如果你是买来的，那你就是大笨蛋哈哈哈哈哈哈因为这是免费的\n" +

"联系方式:1610494022@qq.com,为大学生提供免费支持");

}

private void initStudentSelectionPermissions() {

Long studentRoleId = 3L;

List<Long> selectionMenuIds = new ArrayList<>();

selectionMenuIds.add(6511L);

selectionMenuIds.add(6512L);

selectionMenuIds.add(6513L);

selectionMenuIds.add(6514L);

log.info("为学生角色(ID={})初始化选课相关权限...", studentRoleId);

try {

List<SysRoleMenu> existingPermissions = sysRoleMenuMapper.selectList(

new LambdaQueryWrapper<SysRoleMenu>()

.eq(SysRoleMenu::getRoleId, studentRoleId)

.in(SysRoleMenu::getMenuId, selectionMenuIds)

);

List<Long> existingMenuIds = existingPermissions.stream()

.map(SysRoleMenu::getMenuId)

.collect(Collectors.toList());

List<Long> newMenuIds = selectionMenuIds.stream()

.filter(menuId -> !existingMenuIds.contains(menuId))

.collect(Collectors.toList());

if (newMenuIds.isEmpty()) {

log.info("如果你是买来的，那你就是大笨蛋哈哈哈哈哈哈因为这是免费的\n" +

" 联系方式:1610494022@qq.com,为大学生提供免费支持");

return;

}

List<SysRoleMenu> roleMenuList = new ArrayList<>();

for (Long menuId : newMenuIds) {

SysRoleMenu roleMenu = new SysRoleMenu();

roleMenu.setRoleId(studentRoleId);

roleMenu.setMenuId(menuId);

roleMenuList.add(roleMenu);

}

if (!roleMenuList.isEmpty()) {

int insertCount = sysRoleMenuMapper.batchInsert(roleMenuList);

log.info("为学生角色添加了{}个选课相关权限: {}", insertCount, newMenuIds);

}

} catch (Exception e) {

log.warn("添加学生选课权限时发生异常, 将尝试单条插入: {}", e.getMessage());

for (Long menuId : selectionMenuIds) {

try {

SysRoleMenu roleMenu = new SysRoleMenu();

roleMenu.setRoleId(studentRoleId);

roleMenu.setMenuId(menuId);

sysRoleMenuMapper.insert(roleMenu);

log.info("为学生角色添加选课权限菜单ID: {}", menuId);

} catch (Exception ex) {

log.debug("菜单权限可能已存在，跳过: roleId={}, menuId={}", studentRoleId, menuId);

}

}

}

}

}

package com.example.student.config;

import com.example.student.filter.JwtAuthenticationFilter;

import com.example.student.handler.AccessDeniedHandlerImpl;

import com.example.student.handler.AuthenticationEntryPointImpl;

import jakarta.annotation.Resource;

import lombok.extern.slf4j.Slf4j;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.ProviderManager;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

@EnableMethodSecurity(prePostEnabled = true)

@Slf4j

public class SecurityConfig {

@Resource

private JwtAuthenticationFilter jwtAuthenticationFilter;

@Resource

private UserDetailsService userDetailsService;

@Resource

private AuthenticationEntryPointImpl authenticationEntryPoint;

@Resource

private AccessDeniedHandlerImpl accessDeniedHandler;

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManager() {

log.info("创建认证管理器，使用UserDetailsService: {}", userDetailsService.getClass().getName());

DaoAuthenticationProvider provider = new DaoAuthenticationProvider();

provider.setUserDetailsService(userDetailsService);

provider.setPasswordEncoder(passwordEncoder());

return new ProviderManager(provider);

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

return http

.csrf(AbstractHttpConfigurer::disable)

.sessionManagement(sessionManagement -> sessionManagement

.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

.authorizeHttpRequests(authorizeRequests -> authorizeRequests

.requestMatchers(HttpMethod.OPTIONS).permitAll()

.requestMatchers("/", "/index.html").permitAll()

.requestMatchers("/favicon.ico").permitAll()

.requestMatchers("/api/favicon.ico").permitAll()

.requestMatchers("/static

@Configuration

public class SwaggerConfig {

@Bean

public OpenAPI openAPI() {

final String securitySchemeName = "Bearer认证";

return new OpenAPI()

.info(new Info()

.title("学生信息管理系统API文档")

.description("学生信息管理系统API文档，包含系统所有接口")

.version("1.0.0")

.contact(new Contact()

.name("example")

.email("example@example.com")

.url("https:

.license(new License()

.name("Apache 2.0")

.url("https:

.addSecurityItem(new SecurityRequirement().addList(securitySchemeName))

.components(new Components()

.addSecuritySchemes(securitySchemeName, new SecurityScheme()

.name(securitySchemeName)

.type(SecurityScheme.Type.HTTP)

.scheme("bearer")

.bearerFormat("JWT")

.description("JWT认证，请在下方输入框输入Bearer {token}"))

);

}

}

package com.example.student.config;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class WebMvcConfig implements WebMvcConfigurer {

@Value("${file.upload.path:/uploads}")

private String uploadPath;

@Value("${file.access.path:/uploads}")

private String accessPath;

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler(accessPath + "

@Tag(name = "管理员管理", description = "管理员管理相关接口")

@RestController

@RequestMapping("/admin")

public class AdminController {

@Resource

private AdminService adminService;

@Operation(summary = "分页查询管理员", description = "分页查询管理员接口")

@PreAuthorize("@ss.hasPermission('system:user:list')")

@PostMapping("/list")

public Result<PageResult<AdminVO>> list(@Valid @RequestBody AdminQueryDTO queryDTO) {

PageResult<AdminVO> pageResult = adminService.pageAdmin(queryDTO);

return Result.success(pageResult);

}

@Operation(summary = "获取管理员详情", description = "获取管理员详情接口")

@PreAuthorize("@ss.hasPermission('system:user:query')")

@GetMapping("/{id}")

public Result<AdminVO> getInfo(@Parameter(description = "管理员ID") @PathVariable Long id) {

AdminVO admin = adminService.getAdminById(id);

return Result.success(admin);

}

@Operation(summary = "新增管理员", description = "新增管理员接口")

@PreAuthorize("@ss.hasPermission('system:user:add')")

@PostMapping

public Result<Boolean> add(@Valid @RequestBody AdminDTO adminDTO) {

boolean result = adminService.addAdmin(adminDTO);

return Result.success(result);

}

@Operation(summary = "修改管理员", description = "修改管理员接口")

@PreAuthorize("@ss.hasPermission('system:user:edit')")

@PutMapping

public Result<Boolean> update(@Valid @RequestBody AdminDTO adminDTO) {

boolean result = adminService.updateAdmin(adminDTO);

return Result.success(result);

}

@Operation(summary = "删除管理员", description = "删除管理员接口")

@PreAuthorize("@ss.hasPermission('system:user:delete')")

@DeleteMapping("/{id}")

public Result<Boolean> delete(@Parameter(description = "管理员ID") @PathVariable Long id) {

boolean result = adminService.deleteAdmin(id);

return Result.success(result);

}

@Operation(summary = "重置密码", description = "重置密码接口")

@PreAuthorize("@ss.hasPermission('system:user:resetPwd')")

@PutMapping("/resetPassword/{id}")

public Result<Boolean> resetPassword(@Parameter(description = "管理员ID") @PathVariable Long id) {

boolean result = adminService.resetPassword(id);

return Result.success(result);

}

@Operation(summary = "更新状态", description = "更新状态接口")

@PreAuthorize("@ss.hasPermission('system:user:edit')")

@PutMapping("/status/{id}/{status}")

public Result<Boolean> updateStatus(

@Parameter(description = "管理员ID") @PathVariable Long id,

@Parameter(description = "状态") @PathVariable Integer status) {

boolean result = adminService.updateStatus(id, status);

return Result.success(result);

}

}

package com.example.student.controller;

import com.baomidou.mybatisplus.core.metadata.IPage;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

import com.example.student.common.CommonPage;

import com.example.student.common.CommonResult;

import com.example.student.entity.Attendance;

import com.example.student.service.AttendanceService;

import com.example.student.vo.AttendanceStatisticsVO;

import com.example.student.vo.AttendanceVO;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import io.swagger.v3.oas.annotations.Operation;

import jakarta.servlet.http.HttpServletResponse;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

@RestController

@RequestMapping("/attendance")

@Slf4j

public class AttendanceController {

@Autowired

private AttendanceService attendanceService;

@Operation(summary = "查询考勤记录列表")

@PostMapping("/list")

@PreAuthorize("@ss.hasPermission('attendance:record:list')")

public CommonResult<CommonPage<AttendanceVO>> list(@RequestBody Map<String, Object> params) {

log.info("查询考勤记录列表参数: {}", params);

Integer pageNum = (Integer) params.getOrDefault("pageNum", 1);

Integer pageSize = (Integer) params.getOrDefault("pageSize", 10);

if (params.containsKey("courseId") && params.get("courseId") != null) {

log.info("检测到课程ID参数: {}", params.get("courseId"));

}

Page<Attendance> page = new Page<>(pageNum, pageSize);

Page<AttendanceVO> resultPage = attendanceService.listAttendanceRecords(page, params);

return CommonResult.success(CommonPage.restPage(resultPage));

}

@PostMapping("/statistics")

@PreAuthorize("@ss.hasPermission('attendance:stats:list')")

public CommonResult<CommonPage<AttendanceStatisticsVO>> statistics(@RequestBody Map<String, Object> params) {

Integer pageNum = (Integer) params.getOrDefault("pageNum", 1);

Integer pageSize = (Integer) params.getOrDefault("pageSize", 10);

Page<AttendanceStatisticsVO> page = new Page<>(pageNum, pageSize);

Page<AttendanceStatisticsVO> resultPage = attendanceService.getAttendanceStatistics(page, params);

return CommonResult.success(CommonPage.restPage(resultPage));

}

@GetMapping("/course/{courseId}/statistics")

@PreAuthorize("@ss.hasPermission('attendance:stats:list')")

public CommonResult<AttendanceStatisticsVO> getCourseStatistics(@PathVariable("courseId") Long courseId) {

AttendanceStatisticsVO statistics = attendanceService.getCourseAttendanceStatistics(courseId);

return CommonResult.success(statistics);

}

@GetMapping("/class/{classId}/statistics")

@PreAuthorize("@ss.hasPermission('attendance:stats:list')")

public CommonResult<AttendanceStatisticsVO> getClassStatistics(@PathVariable("classId") Long classId) {

AttendanceStatisticsVO statistics = attendanceService.getClassAttendanceStatistics(classId);

return CommonResult.success(statistics);

}

@PostMapping("/batch")

@PreAuthorize("@ss.hasPermission('attendance:record:add') or @ss.hasPermission('attendance:record:edit') or @ss.hasPermission('attendance:record:remove')")

public CommonResult<Boolean> batch(@RequestBody Map<String, Object> params) {

if ("delete".equals(params.get("action"))) {

Object ids = params.get("ids");

if (ids instanceof List) {

List<Long> idList = new ArrayList<>();

for (Object id : (List<?>) ids) {

if (id instanceof Integer) {

idList.add(((Integer) id).longValue());

} else if (id instanceof Long) {

idList.add((Long) id);

}

}

boolean success = attendanceService.removeAttendanceByIds(idList);

return CommonResult.success(success);

}

return CommonResult.failed("无效的ID列表");

}

Attendance attendance = new Attendance();

Object id = params.get("id");

attendance.setId(id != null ? Long.valueOf(id.toString()) : null);

attendance.setCourseOfferingId(Long.valueOf(params.get("courseOfferingId").toString()));

attendance.setStudentId(Long.valueOf(params.get("studentId").toString()));

attendance.setAttendanceDate(java.time.LocalDate.parse(params.get("attendanceDate").toString()));

attendance.setStatus(Integer.valueOf(params.get("status").toString()));

attendance.setRemark((String) params.get("remark"));

boolean success = true;

if (attendance.getId() != null) {

success = attendanceService.updateById(attendance);

} else {

success = attendanceService.save(attendance);

}

return CommonResult.success(success);

}

@PostMapping("/export")

@PreAuthorize("@ss.hasPermission('attendance:record:export')")

public void export(HttpServletResponse response, @RequestBody Map<String, Object> params) {

List<AttendanceVO> list = attendanceService.exportAttendance(params);

}

}

package com.example.student.controller;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

import com.example.student.common.Result;

import com.example.student.entity.Class;

import com.example.student.entity.Teacher;

import com.example.student.mapper.ClassMapper;

import com.example.student.mapper.TeacherMapper;

import com.example.student.vo.ClassVO;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDateTime;

import java.util.List;

import java.util.Map;

import java.util.HashMap;

@Tag(name = "班级管理", description = "班级管理相关接口")

@RestController

@RequestMapping("/student/class")

@Slf4j

public class ClassController {

@Resource

private ClassMapper classMapper;

@Resource

private TeacherMapper teacherMapper;

@Operation(summary = "获取班级列表", description = "分页获取班级列表")

@GetMapping("/list")

@PreAuthorize("hasAuthority('education:class:list')")

public Result<Page<ClassVO>> getClassList(

@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer size,

@RequestParam(required = false) String name,

@RequestParam(required = false) String code,

@RequestParam(required = false) Long majorId,

@RequestParam(required = false) String grade) {

log.info("查询班级列表，参数：current={}, size={}, name={}, code={}, majorId={}, grade={}",

current, size, name, code, majorId, grade);

try {

Page<Class> page = new Page<>(current, size);

LambdaQueryWrapper<Class> queryWrapper = new LambdaQueryWrapper<>();

if (name != null && !name.isEmpty()) {

queryWrapper.like(Class::getName, name);

}

if (code != null && !code.isEmpty()) {

queryWrapper.like(Class::getCode, code);

}

if (majorId != null) {

queryWrapper.eq(Class::getMajorId, majorId);

}

if (grade != null && !grade.isEmpty()) {

queryWrapper.eq(Class::getGrade, grade);

}

queryWrapper.orderByDesc(Class::getUpdateTime);

Page<Class> classPage = classMapper.selectPage(page, queryWrapper);

Page<ClassVO> voPage = convertToVoPage(classPage);

log.info("查询到班级数量：{}", voPage.getTotal());

return Result.success(voPage);

} catch (Exception e) {

log.error("查询班级列表失败", e);

return Result.error("获取班级列表失败：" + e.getMessage());

}

}

@Operation(summary = "获取班级详情", description = "根据ID获取班级详情")

@GetMapping("/{id}")

@PreAuthorize("hasAuthority('education:class:list')")

public Result<ClassVO> getClassDetail(@PathVariable @Parameter(description = "班级ID") Long id) {

log.info("查询班级详情，ID：{}", id);

try {

ClassVO classVO = classMapper.selectClassVOById(id);

if (classVO == null) {

log.warn("班级不存在，ID：{}", id);

return Result.error("班级不存在");

}

log.info("查询班级详情成功，ID：{}", id);

return Result.success(classVO);

} catch (Exception e) {

log.error("查询班级详情失败", e);

return Result.error("获取班级详情失败：" + e.getMessage());

}

}

@Operation(summary = "添加班级", description = "添加班级信息")

@PostMapping

@PreAuthorize("hasAuthority('education:class:add')")

public Result<Void> addClass(@RequestBody Class classInfo) {

log.info("添加班级：{}", classInfo);

try {

LocalDateTime now = LocalDateTime.now();

classInfo.setCreateTime(now);

classInfo.setUpdateTime(now);

int rows = classMapper.insert(classInfo);

if (rows > 0) {

log.info("添加班级成功，ID：{}", classInfo.getId());

return Result.success("添加班级成功");

} else {

log.warn("添加班级失败");

return Result.error("添加班级失败");

}

} catch (Exception e) {

log.error("添加班级失败", e);

return Result.error("添加班级失败：" + e.getMessage());

}

}

@Operation(summary = "更新班级", description = "更新班级信息")

@PutMapping

@PreAuthorize("hasAuthority('education:class:edit')")

public Result<Void> updateClass(@RequestBody Class classInfo) {

log.info("更新班级：{}", classInfo);

try {

if (classInfo.getId() == null) {

log.warn("更新班级失败：ID不能为空");

return Result.error("班级ID不能为空");

}

Class existingClass = classMapper.selectById(classInfo.getId());

if (existingClass == null) {

log.warn("更新班级失败：班级不存在，ID：{}", classInfo.getId());

return Result.error("班级不存在");

}

classInfo.setUpdateTime(LocalDateTime.now());

classInfo.setCreateTime(existingClass.getCreateTime());

int rows = classMapper.updateById(classInfo);

if (rows > 0) {

log.info("更新班级成功，ID：{}", classInfo.getId());

return Result.success("更新班级成功");

} else {

log.warn("更新班级失败，ID：{}", classInfo.getId());

return Result.error("更新班级失败");

}

} catch (Exception e) {

log.error("更新班级失败", e);

return Result.error("更新班级失败：" + e.getMessage());

}

}

@Operation(summary = "删除班级", description = "根据ID删除班级")

@DeleteMapping("/{id}")

@PreAuthorize("hasAuthority('education:class:remove')")

public Result<Void> deleteClass(@PathVariable @Parameter(description = "班级ID") Long id) {

log.info("删除班级，ID：{}", id);

try {

Class existingClass = classMapper.selectById(id);

if (existingClass == null) {

log.warn("删除班级失败：班级不存在，ID：{}", id);

return Result.error("班级不存在");

}

int rows = classMapper.deleteById(id);

if (rows > 0) {

log.info("删除班级成功，ID：{}", id);

return Result.success("删除班级成功");

} else {

log.warn("删除班级失败，ID：{}", id);

return Result.error("删除班级失败");

}

} catch (Exception e) {

log.error("删除班级失败", e);

return Result.error("删除班级失败：" + e.getMessage());

}

}

@Operation(summary = "获取所有班级", description = "获取所有班级，不分页")

@GetMapping("/all")

public Result<List<ClassVO>> getAllClasses() {

log.info("开始查询所有班级信息");

try {

List<ClassVO> classList = classMapper.selectClassVOList();

log.info("查询到{}个班级信息", classList.size());

return Result.success(classList);

} catch (Exception e) {

log.error("查询班级列表失败", e);

return Result.error("获取班级列表失败：" + e.getMessage());

}

}

private Page<ClassVO> convertToVoPage(Page<Class> classPage) {

Page<ClassVO> voPage = new Page<>(classPage.getCurrent(), classPage.getSize(), classPage.getTotal());

List<ClassVO> voList = classMapper.selectClassVOListByIds(classPage.getRecords());

voPage.setRecords(voList);

return voPage;

}

@Operation(summary = "获取所有教师", description = "获取所有教师信息，简化版")

@GetMapping("/all/teachers")

public Result<List<Map<String, Object>>> getAllTeachers() {

log.info("开始查询所有教师基本信息");

try {

List<Map<String, Object>> teachers = teacherMapper.selectTeacherOptions();

log.info("查询到{}个教师信息", teachers.size());

return Result.success(teachers);

} catch (Exception e) {

log.error("查询教师列表失败", e);

return Result.error("获取教师列表失败：" + e.getMessage());

}

}

@Operation(summary = "根据姓名搜索教师", description = "根据姓名搜索教师信息")

@GetMapping("/list/teachers")

public Result<List<Map<String, Object>>> searchTeachers(@RequestParam(required = false) String name) {

log.info("开始根据姓名搜索教师，name: {}", name);

try {

List<Map<String, Object>> teachers = teacherMapper.searchTeachersByName(name);

log.info("搜索到{}个教师信息", teachers.size());

return Result.success(teachers);

} catch (Exception e) {

log.error("搜索教师列表失败", e);

return Result.error("搜索教师列表失败：" + e.getMessage());

}

}

}

package com.example.student.controller;

import com.example.student.common.PageResult;

import com.example.student.common.Result;

import com.example.student.dto.CourseDTO;

import com.example.student.dto.CourseQueryDTO;

import com.example.student.entity.Course;

import com.example.student.service.CourseService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.validation.Valid;

import jakarta.validation.constraints.NotNull;

import java.io.IOException;

import java.util.List;

import java.util.Map;

@Tag(name = "课程管理接口")

@RestController

@RequestMapping("/course")

@RequiredArgsConstructor

@Validated

@Slf4j

public class CourseController {

private final CourseService courseService;

@Operation(summary = "获取课程列表")

@GetMapping("/list")

@PreAuthorize("hasAuthority('course:course:list')")

public Result<PageResult<Course>> list(CourseQueryDTO queryDTO) {

PageResult<Course> pageResult = courseService.getCoursePage(queryDTO);

return Result.success(pageResult);

}

@Operation(summary = "获取课程详情")

@GetMapping("/{id}")

@PreAuthorize("hasAnyAuthority('course:course:query', 'course:my-courses:detail')")

public Result<Course> getInfo(@PathVariable("id") @NotNull Long id) {

log.info("获取课程详情, id: {}", id);

Course course = courseService.getCourseById(id);

return Result.success(course);

}

@Operation(summary = "新增课程")

@PostMapping

@PreAuthorize("hasAuthority('course:course:add')")

public Result<?> add(@RequestBody @Valid CourseDTO courseDTO) {

courseService.addCourse(courseDTO);

return Result.success();

}

@Operation(summary = "修改课程")

@PutMapping

@PreAuthorize("hasAuthority('course:course:edit')")

public Result<?> update(@RequestBody @Valid CourseDTO courseDTO) {

courseService.updateCourse(courseDTO);

return Result.success();

}

@Operation(summary = "删除课程")

@DeleteMapping("/{id}")

@PreAuthorize("hasAuthority('course:course:delete')")

public Result<?> delete(@PathVariable("id") @NotNull Long id) {

courseService.deleteCourse(id);

return Result.success();

}

@Operation(summary = "导出课程数据")

@PostMapping("/export")

@PreAuthorize("hasAuthority('course:course:export')")

public void export(CourseQueryDTO queryDTO, HttpServletResponse response) throws IOException {

courseService.exportCourse(queryDTO, response);

}

@Operation(summary = "导入课程数据")

@PostMapping("/import")

@PreAuthorize("hasAuthority('course:course:import')")

public Result<?> importData(MultipartFile file) throws IOException {

courseService.importCourse(file);

return Result.success();

}

@Operation(summary = "根据院系ID获取课程列表")

@GetMapping("/department/{departmentId}")

@PreAuthorize("hasAuthority('course:course:list')")

public Result<List<Course>> getCoursesByDepartment(@PathVariable("departmentId") @NotNull Long departmentId) {

List<Course> courses = courseService.getCoursesByDepartmentId(departmentId);

return Result.success(courses);

}

@Operation(summary = "获取课程统计数据")

@GetMapping("/stats")

@PreAuthorize("hasAuthority('course:course:query')")

public Result<?> getCourseStats() {

return Result.success(courseService.getCourseStats());

}

@Operation(summary = "获取考勤记录模块可用的课程列表", description = "支持按课程名称模糊搜索，供考勤记录模块使用")

@GetMapping("/listForAttendance")

@PreAuthorize("hasAuthority('attendance:record:list')")

public Result<PageResult<Course>> listForAttendance(

@RequestParam(required = false) String keyword,

@RequestParam(defaultValue = "1") Integer pageNum,

@RequestParam(defaultValue = "10") Integer pageSize) {

log.info("考勤记录模块查询课程列表，关键字: {}", keyword);

PageResult<Course> pageResult = courseService.getCoursesForAttendance(keyword, pageNum, pageSize);

log.info("考勤记录模块查询课程列表结果数量: {}",

pageResult != null && pageResult.getList() != null ? pageResult.getList().size() : 0);

return Result.success(pageResult);

}

@Operation(summary = "检查课程时间冲突")

@PostMapping("/check-conflict")

@PreAuthorize("hasAnyAuthority('course:selection:select', 'course:selection:available', 'course:selection:selected')")

public Result<Boolean> checkTimeConflict(@RequestBody Map<String, String> params) {

String courseTime1 = params.get("courseTime1");

String courseTime2 = params.get("courseTime2");

log.info("检查课程时间冲突: courseTime1={}, courseTime2={}", courseTime1, courseTime2);

if (courseTime1 == null || courseTime2 == null ||

courseTime1.isEmpty() || courseTime2.isEmpty()) {

return Result.success(false);

}

try {

boolean hasConflict = courseService.checkTimeConflict(courseTime1, courseTime2);

return Result.success(hasConflict);

} catch (Exception e) {

log.error("检查课程时间冲突出错", e);

return Result.error("检查时间冲突失败: " + e.getMessage());

}

}

}

package com.example.student.controller;

import com.example.student.common.PageResult;

import com.example.student.common.Result;

import com.example.student.dto.CourseDTO;

import com.example.student.dto.CourseQueryDTO;

import com.example.student.entity.Course;

import com.example.student.service.CourseService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.validation.Valid;

import jakarta.validation.constraints.NotNull;

import java.io.IOException;

import java.util.List;

import java.util.Map;

@Tag(name = "课程管理接口")

@RestController

@RequestMapping("/course")

@RequiredArgsConstructor

@Validated

@Slf4j

public class CourseController {

private final CourseService courseService;

@Operation(summary = "获取课程列表")

@GetMapping("/list")

@PreAuthorize("hasAuthority('course:course:list')")

public Result<PageResult<Course>> list(CourseQueryDTO queryDTO) {

PageResult<Course> pageResult = courseService.getCoursePage(queryDTO);

return Result.success(pageResult);

}

@Operation(summary = "获取课程详情")

@GetMapping("/{id}")

@PreAuthorize("hasAnyAuthority('course:course:query', 'course:my-courses:detail')")

public Result<Course> getInfo(@PathVariable("id") @NotNull Long id) {

log.info("获取课程详情, id: {}", id);

Course course = courseService.getCourseById(id);

return Result.success(course);

}

@Operation(summary = "新增课程")

@PostMapping

@PreAuthorize("hasAuthority('course:course:add')")

public Result<?> add(@RequestBody @Valid CourseDTO courseDTO) {

courseService.addCourse(courseDTO);

return Result.success();

}

@Operation(summary = "修改课程")

@PutMapping

@PreAuthorize("hasAuthority('course:course:edit')")

public Result<?> update(@RequestBody @Valid CourseDTO courseDTO) {

courseService.updateCourse(courseDTO);

return Result.success();

}

@Operation(summary = "删除课程")

@DeleteMapping("/{id}")

@PreAuthorize("hasAuthority('course:course:delete')")

public Result<?> delete(@PathVariable("id") @NotNull Long id) {

courseService.deleteCourse(id);

return Result.success();

}

@Operation(summary = "导出课程数据")

@PostMapping("/export")

@PreAuthorize("hasAuthority('course:course:export')")

public void export(CourseQueryDTO queryDTO, HttpServletResponse response) throws IOException {

courseService.exportCourse(queryDTO, response);

}

@Operation(summary = "导入课程数据")

@PostMapping("/import")

@PreAuthorize("hasAuthority('course:course:import')")

public Result<?> importData(MultipartFile file) throws IOException {

courseService.importCourse(file);

return Result.success();

}

@Operation(summary = "根据院系ID获取课程列表")

@GetMapping("/department/{departmentId}")

@PreAuthorize("hasAuthority('course:course:list')")

public Result<List<Course>> getCoursesByDepartment(@PathVariable("departmentId") @NotNull Long departmentId) {

List<Course> courses = courseService.getCoursesByDepartmentId(departmentId);

return Result.success(courses);

}

@Operation(summary = "获取课程统计数据")

@GetMapping("/stats")

@PreAuthorize("hasAuthority('course:course:query')")

public Result<?> getCourseStats() {

return Result.success(courseService.getCourseStats());

}

@Operation(summary = "获取考勤记录模块可用的课程列表", description = "支持按课程名称模糊搜索，供考勤记录模块使用")

@GetMapping("/listForAttendance")

@PreAuthorize("hasAuthority('attendance:record:list')")

public Result<PageResult<Course>> listForAttendance(

@RequestParam(required = false) String keyword,

@RequestParam(defaultValue = "1") Integer pageNum,

@RequestParam(defaultValue = "10") Integer pageSize) {

log.info("考勤记录模块查询课程列表，关键字: {}", keyword);

PageResult<Course> pageResult = courseService.getCoursesForAttendance(keyword, pageNum, pageSize);

log.info("考勤记录模块查询课程列表结果数量: {}",

pageResult != null && pageResult.getList() != null ? pageResult.getList().size() : 0);

return Result.success(pageResult);

}

@Operation(summary = "检查课程时间冲突")

@PostMapping("/check-conflict")

@PreAuthorize("hasAnyAuthority('course:selection:select', 'course:selection:available', 'course:selection:selected')")

public Result<Boolean> checkTimeConflict(@RequestBody Map<String, String> params) {

String courseTime1 = params.get("courseTime1");

String courseTime2 = params.get("courseTime2");

log.info("检查课程时间冲突: courseTime1={}, courseTime2={}", courseTime1, courseTime2);

if (courseTime1 == null || courseTime2 == null ||

courseTime1.isEmpty() || courseTime2.isEmpty()) {

return Result.success(false);

}

try {

boolean hasConflict = courseService.checkTimeConflict(courseTime1, courseTime2);

return Result.success(hasConflict);

} catch (Exception e) {

log.error("检查课程时间冲突出错", e);

return Result.error("检查时间冲突失败: " + e.getMessage());

}

}

}

package com.example.student.controller;

import com.example.student.common.PageResult;

import com.example.student.common.Result;

import com.example.student.dto.CourseSelectionDTO;

import com.example.student.dto.CourseSelectionQueryDTO;

import com.example.student.service.CourseSelectionService;

import com.example.student.service.StudentService;

import com.example.student.util.SecurityUtils;

import com.example.student.vo.CourseSelectionStatisticsVO;

import com.example.student.vo.CourseTypeStatisticsVO;

import com.example.student.vo.SemesterStatisticsVO;

import com.example.student.vo.SemesterVO;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.\*;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.validation.Valid;

import jakarta.validation.constraints.NotNull;

import java.io.IOException;

import java.util.List;

import java.util.Map;

@Tag(name = "选课管理接口")

@RestController

@RequestMapping("/selection/manage")

@RequiredArgsConstructor

@Validated

@Slf4j

public class CourseSelectionController {

private final CourseSelectionService courseSelectionService;

private final StudentService studentService;

@Operation(summary = "获取学生已选课程")

@GetMapping("/{studentId}/selected-courses")

@PreAuthorize("hasAuthority('course:my-courses:list')")

public Result<PageResult<?>> getStudentSelectedCourses(

@Parameter(description = "学生ID") @PathVariable Long studentId,

@RequestParam Map<String, Object> params) {

log.info("获取学生已选课程, 学生ID: {}, 参数: {}", studentId, params);

PageResult<?> pageResult = courseSelectionService.getStudentSelectedCourses(studentId, params);

return Result.success(pageResult);

}

@Operation(summary = "获取课程详情")

@GetMapping("/detail/{courseId}")

@PreAuthorize("hasAuthority('selection:detail')")

public Result<?> getCourseDetail(@PathVariable Long courseId) {

return Result.success(courseSelectionService.getCourseDetail(courseId));

}

@Operation(summary = "获取课程教材信息")

@GetMapping("/materials/{courseId}")

@PreAuthorize("hasAuthority('selection:detail')")

public Result<?> getCourseMaterials(@PathVariable Long courseId) {

return Result.success(courseSelectionService.getCourseMaterials(courseId));

}

@Operation(summary = "获取学生可选课程列表")

@GetMapping("/{studentId}/available-courses")

@PreAuthorize("hasAuthority('course:selection:available')")

public Result<PageResult<?>> getAvailableCourses(

@Parameter(description = "学生ID") @PathVariable Long studentId,

@RequestParam Map<String, Object> params) {

log.info("获取学生可选课程, 学生ID: {}, 参数: {}", studentId, params);

PageResult<?> pageResult = courseSelectionService.getAvailableCourses(studentId, params);

return Result.success(pageResult);

}

@Operation(summary = "学生选课")

@PostMapping("/select/{courseOfferingId}")

@PreAuthorize("hasAuthority('selection:select')")

public Result<?> selectCourse(

@Parameter(description = "课程开设ID", required = true) @PathVariable Long courseOfferingId,

@Parameter(description = "学生ID", required = false) @RequestParam(required = false) Long studentId) {

if (studentId == null) {

Long userId = SecurityUtils.getUserId();

studentId = studentService.getStudentIdByUserId(userId);

if (studentId == null) {

return Result.error("当前用户没有关联的学生账号");

}

}

boolean result = courseSelectionService.selectCourse(studentId, courseOfferingId);

return Result.success(result);

}

@Operation(summary = "学生退选课程")

@PostMapping("/withdraw/{courseOfferingId}")

@PreAuthorize("hasAuthority('selection:withdraw')")

public Result<?> withdrawCourse(

@Parameter(description = "课程开设ID", required = true) @PathVariable Long courseOfferingId,

@Parameter(description = "学生ID", required = false) @RequestParam(required = false) Long studentId) {

if (studentId == null) {

Long userId = SecurityUtils.getUserId();

studentId = studentService.getStudentIdByUserId(userId);

if (studentId == null) {

return Result.error("当前用户没有关联的学生账号");

}

}

boolean result = courseSelectionService.withdrawCourse(studentId, courseOfferingId);

return Result.success(result);

}

@Operation(summary = "获取选课设置")

@GetMapping("/settings")

@PreAuthorize("hasAnyAuthority('course:selection:available', 'course:selection:selected')")

public Result<?> getSelectionSettings() {

return Result.success(courseSelectionService.getSelectionSettings());

}

@Operation(summary = "检查课程时间冲突")

@PostMapping("/check-conflict")

@PreAuthorize("hasAuthority('course:selection:select')")

public Result<Boolean> checkTimeConflict(@RequestBody Map<String, String> params) {

String courseTime1 = params.get("courseTime1");

String courseTime2 = params.get("courseTime2");

boolean hasConflict = courseSelectionService.checkTimeConflict(courseTime1, courseTime2);

return Result.success(hasConflict);

}

@Operation(summary = "查询选课列表")

@GetMapping("/list")

@PreAuthorize("@auth.hasPermi('course:selection:list')")

public Result<PageResult<?>> list(CourseSelectionQueryDTO queryDTO) {

return Result.success(courseSelectionService.getSelectionList(queryDTO));

}

@Operation(summary = "获取选课详细信息")

@GetMapping("/{id}")

@PreAuthorize("@auth.hasPermi('course:selection:query')")

public Result<?> getInfo(@PathVariable("id") Long id) {

return Result.success(courseSelectionService.getSelectionById(id));

}

@Operation(summary = "修改选课信息")

@PutMapping("")

@PreAuthorize("@auth.hasPermi('course:selection:operate')")

public Result<?> update(@RequestBody @Valid CourseSelectionDTO courseSelectionDTO) {

return Result.success(courseSelectionService.updateSelection(courseSelectionDTO));

}

@Operation(summary = "批量退课操作")

@PutMapping("/batch-withdraw")

@PreAuthorize("@auth.hasPermi('course:selection:operate')")

public Result<?> batchWithdraw(@RequestBody List<Long> ids) {

return Result.success(courseSelectionService.batchWithdrawCourses(ids));

}

@Operation(summary = "退课操作")

@PutMapping("/withdraw/{id}")

@PreAuthorize("@auth.hasPermi('course:selection:operate')")

public Result<?> withdrawById(@PathVariable("id") Long id) {

return Result.success(courseSelectionService.withdrawCourse(id));

}

@Operation(summary = "导出选课数据")

@GetMapping("/export")

@PreAuthorize("@auth.hasPermi('course:selection:export')")

public void export(CourseSelectionQueryDTO queryDTO, HttpServletResponse response) throws IOException {

courseSelectionService.exportSelection(queryDTO, response);

}

@Operation(summary = "获取选课统计数据")

@GetMapping("/statistics")

@PreAuthorize("@auth.hasPermi('course:selection:list')")

public Result<CourseSelectionStatisticsVO> getStatistics() {

return Result.success(courseSelectionService.getSelectionStatistics());

}

@Operation(summary = "获取课程类型选课比例数据")

@GetMapping("/statistics/course-type")

@PreAuthorize("@auth.hasPermi('course:selection:list')")

public Result<List<CourseTypeStatisticsVO>> getCourseTypeStatistics() {

return Result.success(courseSelectionService.getCourseTypeStatistics());

}

@Operation(summary = "获取学期选课统计数据")

@GetMapping("/statistics/semester")

@PreAuthorize("@auth.hasPermi('course:selection:list')")

public Result<List<SemesterStatisticsVO>> getSemesterStatistics() {

return Result.success(courseSelectionService.getSemesterStatistics());

}

@Operation(summary = "获取学期列表")

@GetMapping("/semesters")

@PreAuthorize("@auth.hasPermi('course:selection:list')")

public Result<List<SemesterVO>> getSemesters() {

return Result.success(courseSelectionService.getSemesters());

}

@Operation(summary = "生成选课报表")

@PostMapping("/report")

@PreAuthorize("@auth.hasPermi('course:selection:export')")

public Result<?> generateReport(@RequestBody Map<String, Object> params) {

return Result.success(courseSelectionService.generateSelectionReport(params));

}

}

package com.example.student.controller;

import com.example.student.common.Result;

import com.example.student.entity.ChartData;

import com.example.student.entity.Notice;

import com.example.student.entity.StatisticsData;

import com.example.student.entity.TodoItem;

import com.example.student.service.DashboardService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@Tag(name = "仪表盘管理", description = "仪表盘管理相关接口")

@RestController

@RequestMapping("/dashboard")

@Slf4j

public class DashboardController {

@Autowired

private DashboardService dashboardService;

@Operation(summary = "获取统计数据", description = "获取统计数据")

@GetMapping("/statistics")

@PreAuthorize("hasAnyAuthority('admin', 'teacher', 'student')")

public Result<StatisticsData> getStatistics() {

logCurrentUserAuthorities("getStatistics");

return Result.success(dashboardService.getStatisticsData());

}

@Operation(summary = "获取待办事项", description = "获取待办事项")

@GetMapping("/todo")

@PreAuthorize("hasAnyAuthority('admin', 'teacher', 'student')")

public Result<List<TodoItem>> getTodoList() {

logCurrentUserAuthorities("getTodoList");

return Result.success(dashboardService.getTodoList());

}

@Operation(summary = "获取系统公告", description = "获取系统公告")

@GetMapping("/notice")

@PreAuthorize("hasAnyAuthority('admin', 'teacher', 'student')")

public Result<List<Notice>> getNoticeList() {

logCurrentUserAuthorities("getNoticeList");

return Result.success(dashboardService.getNoticeList());

}

@Operation(summary = "获取学生按学期分布数据", description = "获取学生按学期分布数据")

@GetMapping("/student/semester")

@PreAuthorize("hasAnyAuthority('admin', 'teacher', 'student')")

public Result<ChartData> getStudentSemesterData(@RequestParam(required = false, defaultValue = "近半年") String timeRange) {

logCurrentUserAuthorities("getStudentSemesterData");

log.info("获取学生按学期分布数据，时间范围: {}", timeRange);

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

if (authentication != null && authentication.getAuthorities().stream()

.anyMatch(a -> a.getAuthority().equals("student"))) {

log.info("学生用户访问学期分布数据，返回成绩分布数据");

return Result.success(dashboardService.getStudentGradeDistribution());

}

return Result.success(dashboardService.getStudentSemesterData(timeRange));

}

@Operation(summary = "获取院系分布数据", description = "获取院系分布数据")

@GetMapping("/department/distribution")

@PreAuthorize("hasAnyAuthority('admin', 'teacher', 'student')")

public Result<ChartData> getDepartmentDistribution(@RequestParam(required = false, defaultValue = "今年") String year) {

logCurrentUserAuthorities("getDepartmentDistribution");

log.info("获取院系分布数据，年份: {}", year);

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

if (authentication != null && authentication.getAuthorities().stream()

.anyMatch(a -> a.getAuthority().equals("student"))) {

log.info("学生用户访问院系分布数据，返回个人学习统计数据");

return Result.success(dashboardService.getStudentPersonalStats());

}

return Result.success(dashboardService.getDepartmentDistribution(year));

}

@Operation(summary = "获取学生个人学习统计数据", description = "获取学生个人学习统计数据")

@GetMapping("/student/personal/stats")

@PreAuthorize("hasAuthority('student')")

public Result<ChartData> getStudentPersonalStats() {

logCurrentUserAuthorities("getStudentPersonalStats");

log.info("获取学生个人学习统计数据");

return Result.success(dashboardService.getStudentPersonalStats());

}

@Operation(summary = "获取学生成绩分布数据", description = "获取学生成绩分布数据")

@GetMapping("/student/grade/distribution")

@PreAuthorize("hasAuthority('student')")

public Result<ChartData> getStudentGradeDistribution() {

logCurrentUserAuthorities("getStudentGradeDistribution");

log.info("获取学生成绩分布数据");

return Result.success(dashboardService.getStudentGradeDistribution());

}

@Operation(summary = "获取学生专用的院系分布数据", description = "获取学生专用的院系分布数据")

@GetMapping("/student/department/distribution")

@PreAuthorize("hasAuthority('student')")

public Result<ChartData> getStudentDepartmentDistribution() {

logCurrentUserAuthorities("getStudentDepartmentDistribution");

log.info("获取学生专用的院系分布数据");

return Result.success(dashboardService.getStudentPersonalStats());

}

@Operation(summary = "获取学生专用的学期数据", description = "获取学生专用的学期数据")

@GetMapping("/student/semester/data")

@PreAuthorize("hasAuthority('student')")

public Result<ChartData> getStudentSemesterData() {

logCurrentUserAuthorities("getStudentSemesterData");

log.info("获取学生专用的学期数据");

return Result.success(dashboardService.getStudentGradeDistribution());

}

private void logCurrentUserAuthorities(String methodName) {

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

if (authentication != null) {

log.info("用户 {} 访问 {} 方法，拥有权限: {}",

authentication.getName(),

methodName,

authentication.getAuthorities());

} else {

log.warn("无认证信息访问 {} 方法", methodName);

}

}

}

package com.example.student.controller;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

import com.example.student.common.Result;

import com.example.student.entity.Department;

import com.example.student.mapper.DepartmentMapper;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDateTime;

import java.util.List;

@Tag(name = "院系管理", description = "院系管理相关接口")

@RestController

@RequestMapping("/education/department")

@Slf4j

public class DepartmentController {

@Resource

private DepartmentMapper departmentMapper;

@Operation(summary = "获取院系列表", description = "分页获取院系列表")

@GetMapping("/list")

@PreAuthorize("hasAuthority('education:department:list')")

public Result<Page<Department>> getDepartmentList(

@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer size,

@RequestParam(required = false) String name,

@RequestParam(required = false) String code) {

log.info("查询院系列表，参数：current={}, size={}, name={}, code={}", current, size, name, code);

try {

Page<Department> page = new Page<>(current, size);

LambdaQueryWrapper<Department> queryWrapper = new LambdaQueryWrapper<>();

if (name != null && !name.isEmpty()) {

queryWrapper.like(Department::getName, name);

}

if (code != null && !code.isEmpty()) {

queryWrapper.like(Department::getCode, code);

}

queryWrapper.orderByDesc(Department::getUpdateTime);

Page<Department> departmentPage = departmentMapper.selectPage(page, queryWrapper);

log.info("查询到院系数量：{}", departmentPage.getTotal());

return Result.success(departmentPage);

} catch (Exception e) {

log.error("查询院系列表失败", e);

return Result.error("获取院系列表失败：" + e.getMessage());

}

}

@Operation(summary = "获取院系详情", description = "根据ID获取院系详情")

@GetMapping("/{id}")

@PreAuthorize("hasAuthority('education:department:query') or hasAuthority('education:department:edit')")

public Result<Department> getDepartmentById(@PathVariable @Parameter(description = "院系ID") Long id) {

log.info("获取院系详情，ID：{}", id);

try {

Department department = departmentMapper.selectById(id);

if (department == null) {

log.warn("获取院系详情失败：院系不存在，ID：{}", id);

return Result.error("院系不存在");

}

return Result.success(department);

} catch (Exception e) {

log.error("获取院系详情失败", e);

return Result.error("获取院系详情失败：" + e.getMessage());

}

}

@Operation(summary = "添加院系", description = "添加院系信息")

@PostMapping

@PreAuthorize("hasAuthority('education:department:add')")

public Result<Void> addDepartment(@RequestBody Department department) {

log.info("添加院系：{}", department);

try {

LambdaQueryWrapper<Department> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.eq(Department::getCode, department.getCode());

if (departmentMapper.selectCount(queryWrapper) > 0) {

log.warn("添加院系失败：院系编码已存在，code={}", department.getCode());

return Result.error("院系编码已存在");

}

LocalDateTime now = LocalDateTime.now();

department.setCreateTime(now);

department.setUpdateTime(now);

int rows = departmentMapper.insert(department);

if (rows > 0) {

log.info("添加院系成功，ID：{}", department.getId());

return Result.success("添加院系成功");

} else {

log.warn("添加院系失败");

return Result.error("添加院系失败");

}

} catch (Exception e) {

log.error("添加院系失败", e);

return Result.error("添加院系失败：" + e.getMessage());

}

}

@Operation(summary = "更新院系", description = "更新院系信息")

@PutMapping

@PreAuthorize("hasAuthority('education:department:edit')")

public Result<Void> updateDepartment(@RequestBody Department department) {

log.info("更新院系：{}", department);

try {

if (department.getId() == null) {

log.warn("更新院系失败：ID不能为空");

return Result.error("院系ID不能为空");

}

Department existingDepartment = departmentMapper.selectById(department.getId());

if (existingDepartment == null) {

log.warn("更新院系失败：院系不存在，ID：{}", department.getId());

return Result.error("院系不存在");

}

if (!existingDepartment.getCode().equals(department.getCode())) {

LambdaQueryWrapper<Department> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.eq(Department::getCode, department.getCode());

if (departmentMapper.selectCount(queryWrapper) > 0) {

log.warn("更新院系失败：院系编码已存在，code={}", department.getCode());

return Result.error("院系编码已存在");

}

}

department.setUpdateTime(LocalDateTime.now());

department.setCreateTime(existingDepartment.getCreateTime());

int rows = departmentMapper.updateById(department);

if (rows > 0) {

log.info("更新院系成功，ID：{}", department.getId());

return Result.success("更新院系成功");

} else {

log.warn("更新院系失败，ID：{}", department.getId());

return Result.error("更新院系失败");

}

} catch (Exception e) {

log.error("更新院系失败", e);

return Result.error("更新院系失败：" + e.getMessage());

}

}

@Operation(summary = "删除院系", description = "根据ID删除院系")

@DeleteMapping("/{id}")

@PreAuthorize("hasAuthority('education:department:delete') or hasAuthority('education:department:remove')")

public Result<Void> deleteDepartment(@PathVariable @Parameter(description = "院系ID") Long id) {

log.info("删除院系，ID：{}", id);

try {

Department existingDepartment = departmentMapper.selectById(id);

if (existingDepartment == null) {

log.warn("删除院系失败：院系不存在，ID：{}", id);

return Result.error("院系不存在");

}

int rows = departmentMapper.deleteById(id);

if (rows > 0) {

log.info("删除院系成功，ID：{}", id);

return Result.success("删除院系成功");

} else {

log.warn("删除院系失败，ID：{}", id);

return Result.error("删除院系失败");

}

} catch (Exception e) {

log.error("删除院系失败", e);

return Result.error("删除院系失败：" + e.getMessage());

}

}

@Operation(summary = "获取所有院系", description = "获取所有院系列表（不分页）")

@GetMapping("/all")

@PreAuthorize("hasAuthority('education:department:list')")

public Result<List<Department>> getAllDepartments() {

log.info("获取所有院系");

try {

LambdaQueryWrapper<Department> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.orderByAsc(Department::getCode);

List<Department> departmentList = departmentMapper.selectList(queryWrapper);

log.info("获取所有院系成功，数量：{}", departmentList.size());

return Result.success(departmentList);

} catch (Exception e) {

log.error("获取所有院系失败", e);

return Result.error("获取所有院系失败：" + e.getMessage());

}

}

}

package com.example.student.controller;

import com.example.student.common.Result;

import com.example.student.common.PageResult;

import com.example.student.dto.GradeQueryDTO;

import com.example.student.service.GradeService;

import com.example.student.vo.GradeStatisticsVO;

import com.example.student.vo.StudentGradeVO;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

import lombok.RequiredArgsConstructor;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.\*;

import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.util.List;

import java.util.Map;

@Tag(name = "成绩管理", description = "成绩管理相关接口")

@RestController

@RequestMapping("/grade")

@RequiredArgsConstructor

@Validated

public class GradeController {

private final GradeService gradeService;

@Operation(summary = "获取成绩统计数据")

@GetMapping("/stats")

@PreAuthorize("@auth.hasPermi('grade:stats')")

public Result<GradeStatisticsVO> getGradeStats(GradeQueryDTO queryDTO) {

return Result.success(gradeService.getGradeStatistics(queryDTO));

}

@Operation(summary = "导出成绩统计报告")

@GetMapping("/export-report")

@PreAuthorize("@auth.hasPermi('grade:export')")

public void exportGradeReport(GradeQueryDTO queryDTO, HttpServletResponse response) throws IOException {

if ("all".equals(queryDTO.getSemester())) {

queryDTO.setSemester(null);

}

gradeService.exportGradeReport(queryDTO, response);

}

@Operation(summary = "获取课程选项")

@GetMapping("/course-options")

@PreAuthorize("@auth.hasPermi('grade:query')")

public Result<List<Map<String, Object>>> getCourseOptions() {

return Result.success(gradeService.getCourseOptions());

}

@Operation(summary = "获取班级选项")

@GetMapping("/class-options")

@PreAuthorize("@auth.hasPermi('grade:query')")

public Result<List<Map<String, Object>>> getClassOptions() {

return Result.success(gradeService.getClassOptions());

}

@Operation(summary = "获取学期列表")

@GetMapping("/semester-list")

@PreAuthorize("@auth.hasPermi('grade:query')")

public Result<List<String>> getSemesterList() {

return Result.success(gradeService.getSemesterList());

}

@Operation(summary = "获取学生成绩列表")

@GetMapping("/student-list")

@PreAuthorize("@auth.hasPermi('grade:query')")

public Result<PageResult<StudentGradeVO>> getStudentGradesList(

@RequestParam Long courseId,

@RequestParam String semester,

@RequestParam(required = false) Long classId,

@RequestParam(required = false) String studentName,

@RequestParam(defaultValue = "1") Integer pageNum,

@RequestParam(defaultValue = "10") Integer pageSize) {

return Result.success(

gradeService.getStudentGradesList(courseId, semester, classId, studentName, pageNum, pageSize)

);

}

@Operation(summary = "批量提交学生成绩")

@PostMapping("/submit-grades")

@PreAuthorize("@auth.hasPermi('grade:input')")

public Result<Boolean> batchSubmitGrades(@RequestBody List<Map<String, Object>> grades) {

return Result.success(gradeService.batchSubmitGrades(grades));

}

}

package com.example.student.controller;

import com.example.student.common.Result;

import com.example.student.service.impl.LogDemoServiceImpl;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@Tag(name = "日志功能演示")

@RestController

@RequestMapping("/log")

public class LogDemoController {

@Autowired

private LogDemoServiceImpl logDemoService;

@Operation(summary = "正常日志记录演示")

@GetMapping("/demo")

public Result<String> logDemo(

@Parameter(description = "参数") @RequestParam(required = false) String param) {

String result = logDemoService.logDemo(param);

return Result.success(result);

}

@Operation(summary = "异常日志记录演示")

@GetMapping("/exception")

public Result<String> logExceptionDemo(

@Parameter(description = "参数(输入'error'触发异常，输入非数字触发格式异常)")

@RequestParam String param) {

String result = logDemoService.logExceptionDemo(param);

return Result.success(result);

}

}

package com.example.student.controller;

import com.example.student.common.Result;

import com.example.student.dto.LoginDTO;

import com.example.student.dto.LoginResponseDTO;

import com.example.student.entity.SysMenu;

import com.example.student.service.LoginService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import jakarta.validation.Valid;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RequestMapping("/auth")

@Tag(name = "登录管理", description = "登录相关接口")

@RestController

public class LoginController {

@Resource

private LoginService loginService;

@Operation(summary = "用户登录", description = "用户登录接口")

@PostMapping("/login")

public Result<LoginResponseDTO> login(@Valid @RequestBody LoginDTO loginDTO) {

LoginResponseDTO loginResponse = loginService.login(loginDTO);

return Result.success("登录成功", loginResponse);

}

@Operation(summary = "获取当前登录用户信息", description = "获取当前登录用户信息接口")

@GetMapping("/user/info")

public Result<LoginResponseDTO> getUserInfo() {

LoginResponseDTO userInfo = loginService.getUserInfo();

return Result.success(userInfo);

}

@Operation(summary = "用户登出", description = "用户登出接口")

@PostMapping("/logout")

public Result<Void> logout() {

loginService.logout();

return Result.success("登出成功");

}

@Operation(summary = "获取用户路由菜单", description = "获取当前用户可访问的路由菜单")

@GetMapping("/routes")

public Result<List<SysMenu>> getRoutes() {

List<SysMenu> menuList = loginService.getUserMenuList();

return Result.success(menuList);

}

}

package com.example.student.controller;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

import com.example.student.common.Result;

import com.example.student.entity.Major;

import com.example.student.mapper.MajorMapper;

import com.example.student.vo.MajorVO;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDateTime;

import java.util.List;

@Tag(name = "专业管理", description = "专业管理相关接口")

@RestController

@RequestMapping("/education/major")

@Slf4j

public class MajorController {

@Resource

private MajorMapper majorMapper;

@Operation(summary = "获取专业列表", description = "分页获取专业列表")

@GetMapping("/list")

@PreAuthorize("hasAuthority('education:major:list')")

public Result<Page<MajorVO>> getMajorList(

@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer size,

@RequestParam(required = false) String name,

@RequestParam(required = false) String code,

@RequestParam(required = false) Long departmentId) {

log.info("查询专业列表，参数：current={}, size={}, name={}, code={}, departmentId={}",

current, size, name, code, departmentId);

try {

Page<Major> page = new Page<>(current, size);

Major queryMajor = new Major();

queryMajor.setName(name);

queryMajor.setCode(code);

queryMajor.setDepartmentId(departmentId);

Page<MajorVO> majorVOPage = majorMapper.selectMajorVOPage(page, queryMajor);

log.info("查询到专业数量：{}", majorVOPage.getTotal());

return Result.success(majorVOPage);

} catch (Exception e) {

log.error("查询专业列表失败", e);

return Result.error("获取专业列表失败：" + e.getMessage());

}

}

@Operation(summary = "添加专业", description = "添加专业信息")

@PostMapping

@PreAuthorize("hasAuthority('education:major:add')")

public Result<Void> addMajor(@RequestBody Major major) {

log.info("添加专业：{}", major);

try {

LambdaQueryWrapper<Major> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.eq(Major::getCode, major.getCode());

if (majorMapper.selectCount(queryWrapper) > 0) {

log.warn("添加专业失败：专业编码已存在，code={}", major.getCode());

return Result.error("专业编码已存在");

}

LocalDateTime now = LocalDateTime.now();

major.setCreateTime(now);

major.setUpdateTime(now);

int rows = majorMapper.insert(major);

if (rows > 0) {

log.info("添加专业成功，ID：{}", major.getId());

return Result.success("添加专业成功");

} else {

log.warn("添加专业失败");

return Result.error("添加专业失败");

}

} catch (Exception e) {

log.error("添加专业失败", e);

return Result.error("添加专业失败：" + e.getMessage());

}

}

@Operation(summary = "更新专业", description = "更新专业信息")

@PutMapping

@PreAuthorize("hasAuthority('education:major:edit')")

public Result<Void> updateMajor(@RequestBody Major major) {

log.info("更新专业：{}", major);

try {

if (major.getId() == null) {

log.warn("更新专业失败：ID不能为空");

return Result.error("专业ID不能为空");

}

Major existingMajor = majorMapper.selectById(major.getId());

if (existingMajor == null) {

log.warn("更新专业失败：专业不存在，ID：{}", major.getId());

return Result.error("专业不存在");

}

if (!existingMajor.getCode().equals(major.getCode())) {

LambdaQueryWrapper<Major> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.eq(Major::getCode, major.getCode());

if (majorMapper.selectCount(queryWrapper) > 0) {

log.warn("更新专业失败：专业编码已存在，code={}", major.getCode());

return Result.error("专业编码已存在");

}

}

major.setUpdateTime(LocalDateTime.now());

major.setCreateTime(existingMajor.getCreateTime());

int rows = majorMapper.updateById(major);

if (rows > 0) {

log.info("更新专业成功，ID：{}", major.getId());

return Result.success("更新专业成功");

} else {

log.warn("更新专业失败，ID：{}", major.getId());

return Result.error("更新专业失败");

}

} catch (Exception e) {

log.error("更新专业失败", e);

return Result.error("更新专业失败：" + e.getMessage());

}

}

@Operation(summary = "获取所有专业", description = "获取所有专业，不分页")

@GetMapping("/all")

@PreAuthorize("hasAuthority('education:major:list')")

public Result<List<MajorVO>> getAllMajors() {

log.info("开始查询所有专业信息");

try {

List<MajorVO> majorList = majorMapper.selectMajorVOList();

log.info("查询到{}个专业信息", majorList.size());

return Result.success(majorList);

} catch (Exception e) {

log.error("查询专业列表失败", e);

return Result.error("获取专业列表失败：" + e.getMessage());

}

}

@Operation(summary = "获取院系下的专业", description = "获取特定院系下的所有专业")

@GetMapping("/department/{departmentId}")

@PreAuthorize("hasAuthority('education:major:list')")

public Result<List<MajorVO>> getMajorsByDepartment(

@PathVariable @Parameter(description = "院系ID") Long departmentId) {

log.info("查询院系下的专业，院系ID：{}", departmentId);

try {

Major queryMajor = new Major();

queryMajor.setDepartmentId(departmentId);

Page<Major> page = new Page<>(1, 1000);

Page<MajorVO> majorVOPage = majorMapper.selectMajorVOPage(page, queryMajor);

log.info("查询到院系{}下的{}个专业", departmentId, majorVOPage.getRecords().size());

return Result.success(majorVOPage.getRecords());

} catch (Exception e) {

log.error("查询院系下的专业失败", e);

return Result.error("获取院系下的专业失败：" + e.getMessage());

}

}

@Operation(summary = "获取所有专业（用于下拉选择）", description = "获取所有专业信息，用于前端下拉选择")

@GetMapping(path = "/listAll", produces = "application/json")

@PreAuthorize("hasAuthority('education:major:list')")

public Result<List<MajorVO>> listAllMajors() {

log.info("开始查询所有专业信息（用于下拉选择）");

try {

List<MajorVO> majorList = majorMapper.selectMajorVOList();

log.info("查询到{}个专业信息（用于下拉选择）", majorList.size());

return Result.success(majorList);

} catch (Exception e) {

log.error("查询专业列表失败", e);

return Result.error("获取专业列表失败：" + e.getMessage());

}

}

@Operation(summary = "获取专业详情", description = "根据ID获取专业详情")

@GetMapping("/{id}")

@PreAuthorize("hasAuthority('education:major:list')")

public Result<MajorVO> getMajorDetail(@PathVariable @Parameter(description = "专业ID") Long id) {

log.info("查询专业详情，ID：{}", id);

try {

MajorVO major = majorMapper.selectMajorVOById(id);

if (major == null) {

log.warn("专业不存在，ID：{}", id);

return Result.error("专业不存在");

}

log.info("查询专业详情成功，ID：{}", id);

return Result.success(major);

} catch (Exception e) {

log.error("查询专业详情失败", e);

return Result.error("获取专业详情失败：" + e.getMessage());

}

}

@Operation(summary = "删除专业", description = "根据ID删除专业")

@DeleteMapping("/{id}")

@PreAuthorize("hasAuthority('education:major:remove')")

public Result<Void> deleteMajor(@PathVariable @Parameter(description = "专业ID") Long id) {

log.info("删除专业，ID：{}", id);

try {

Major major = majorMapper.selectById(id);

if (major == null) {

log.warn("删除专业失败：专业不存在，ID：{}", id);

return Result.error("专业不存在");

}

int rows = majorMapper.deleteById(id);

if (rows > 0) {

log.info("删除专业成功，ID：{}", id);

return Result.success("删除专业成功");

} else {

log.warn("删除专业失败，ID：{}", id);

return Result.error("删除专业失败");

}

} catch (Exception e) {

log.error("删除专业失败", e);

return Result.error("删除专业失败：" + e.getMessage());

}

}

@Operation(summary = "获取专业选项列表", description = "获取专业选项列表，适用于下拉框，不需要权限验证")

@GetMapping("/options")

public Result<List<java.util.Map<String, Object>>> getMajorOptions() {

log.info("开始查询专业选项列表");

try {

List<java.util.Map<String, Object>> options = majorMapper.selectMajorOptions();

log.info("查询到{}个专业选项", options.size());

return Result.success(options);

} catch (Exception e) {

log.error("查询专业选项列表失败", e);

return Result.error("获取专业选项列表失败：" + e.getMessage());

}

}

}

package com.example.student.controller;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.example.student.common.Result;

import com.example.student.entity.SysMenu;

import com.example.student.mapper.SysMenuMapper;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import lombok.extern.slf4j.Slf4j;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

@Tag(name = "菜单管理", description = "菜单管理相关接口")

@RestController

@RequestMapping("/system/menu")

@Slf4j

public class MenuController {

@Resource

private SysMenuMapper menuMapper;

@Operation(summary = "获取菜单列表", description = "获取菜单列表")

@GetMapping("/list")

@PreAuthorize("hasAuthority('system:menu:list')")

public Result<List<SysMenu>> getMenuList(

@RequestParam(required = false) String menuName,

@RequestParam(required = false) Integer status) {

log.info("查询菜单列表，参数：menuName={}, status={}", menuName, status);

try {

SysMenu menu = new SysMenu();

if (menuName != null && !menuName.isEmpty()) {

menu.setMenuName(menuName);

}

if (status != null) {

menu.setStatus(status);

}

List<SysMenu> menuList = menuMapper.selectMenuList(menu);

if (menuList == null || menuList.isEmpty()) {

LambdaQueryWrapper<SysMenu> queryWrapper = new LambdaQueryWrapper<>();

if (menuName != null && !menuName.isEmpty()) {

queryWrapper.like(SysMenu::getMenuName, menuName);

}

if (status != null) {

queryWrapper.eq(SysMenu::getStatus, status);

}

queryWrapper.orderByAsc(SysMenu::getParentId, SysMenu::getOrderNum);

menuList = menuMapper.selectList(queryWrapper);

}

log.info("查询到菜单数量：{}", menuList.size());

return Result.success(menuList);

} catch (Exception e) {

log.error("查询菜单列表失败", e);

return Result.error("获取菜单列表失败：" + e.getMessage());

}

}

@Operation(summary = "获取菜单详情", description = "根据ID获取菜单详情")

@GetMapping("/{menuId}")

@PreAuthorize("hasAuthority('system:menu:list')")

public Result<SysMenu> getMenuDetail(@PathVariable @Parameter(description = "菜单ID") Long menuId) {

log.info("查询菜单详情，ID：{}", menuId);

try {

SysMenu menu = menuMapper.selectMenuById(menuId);

if (menu == null) {

menu = menuMapper.selectById(menuId);

if (menu == null) {

log.warn("菜单不存在，ID：{}", menuId);

return Result.error("菜单不存在");

}

}

log.info("查询菜单详情成功，ID：{}", menuId);

return Result.success(menu);

} catch (Exception e) {

log.error("查询菜单详情失败", e);

return Result.error("获取菜单详情失败：" + e.getMessage());

}

}

@Operation(summary = "添加菜单", description = "添加菜单信息")

@PostMapping

@PreAuthorize("hasAuthority('system:menu:add')")

public Result<Void> addMenu(@RequestBody SysMenu menu) {

log.info("添加菜单：{}", menu);

try {

validateMenu(menu, true);

LocalDateTime now = LocalDateTime.now();

menu.setCreateTime(now);

menu.setUpdateTime(now);

int rows = menuMapper.insert(menu);

if (rows > 0) {

log.info("添加菜单成功，ID：{}", menu.getId());

return Result.success("添加菜单成功");

} else {

log.warn("添加菜单失败");

return Result.error("添加菜单失败");

}

} catch (Exception e) {

log.error("添加菜单失败", e);

return Result.error("添加菜单失败：" + e.getMessage());

}

}

@Operation(summary = "更新菜单", description = "更新菜单信息")

@PutMapping

@PreAuthorize("hasAuthority('system:menu:edit')")

public Result<Void> updateMenu(@RequestBody SysMenu menu) {

log.info("更新菜单：{}", menu);

try {

if (menu.getId() == null) {

log.warn("更新菜单失败：ID不能为空");

return Result.error("菜单ID不能为空");

}

SysMenu existingMenu = menuMapper.selectById(menu.getId());

if (existingMenu == null) {

log.warn("更新菜单失败：菜单不存在，ID：{}", menu.getId());

return Result.error("菜单不存在");

}

if (menu.getId().equals(menu.getParentId())) {

log.warn("更新菜单失败：上级菜单不能选择自己，ID：{}", menu.getId());

return Result.error("上级菜单不能选择自己");

}

if (isChildOfMenu(menu.getId(), menu.getParentId())) {

log.warn("更新菜单失败：上级菜单不能选择自己的子菜单，ID：{}", menu.getId());

return Result.error("上级菜单不能选择自己的子菜单");

}

validateMenu(menu, false);

menu.setUpdateTime(LocalDateTime.now());

menu.setCreateTime(existingMenu.getCreateTime());

int rows = menuMapper.updateById(menu);

if (rows > 0) {

log.info("更新菜单成功，ID：{}", menu.getId());

return Result.success("更新菜单成功");

} else {

log.warn("更新菜单失败，ID：{}", menu.getId());

return Result.error("更新菜单失败");

}

} catch (Exception e) {

log.error("更新菜单失败", e);

return Result.error("更新菜单失败：" + e.getMessage());

}

}

@Operation(summary = "删除菜单", description = "根据ID删除菜单")

@DeleteMapping("/{menuId}")

@PreAuthorize("hasAuthority('system:menu:remove')")

public Result<Void> deleteMenu(@PathVariable @Parameter(description = "菜单ID") Long menuId) {

log.info("删除菜单，ID：{}", menuId);

try {

SysMenu existingMenu = menuMapper.selectById(menuId);

if (existingMenu == null) {

log.warn("删除菜单失败：菜单不存在，ID：{}", menuId);

return Result.error("菜单不存在");

}

int childCount = menuMapper.selectChildrenMenuCount(menuId);

if (childCount > 0) {

log.warn("删除菜单失败：存在子菜单，不允许删除，ID：{}", menuId);

return Result.error("存在子菜单，不允许删除");

}

int rows = menuMapper.deleteById(menuId);

if (rows > 0) {

log.info("删除菜单成功，ID：{}", menuId);

return Result.success("删除菜单成功");

} else {

log.warn("删除菜单失败，ID：{}", menuId);

return Result.error("删除菜单失败");

}

} catch (Exception e) {

log.error("删除菜单失败", e);

return Result.error("删除菜单失败：" + e.getMessage());

}

}

@Operation(summary = "获取菜单下拉树结构", description = "获取所有菜单的树形结构数据")

@GetMapping("/tree")

@PreAuthorize("hasAuthority('system:menu:list')")

public Result<List<Map<String, Object>>> getMenuTree() {

log.info("获取菜单树结构");

try {

List<SysMenu> menuList = menuMapper.selectMenuTreeAll();

if (menuList == null || menuList.isEmpty()) {

LambdaQueryWrapper<SysMenu> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.orderByAsc(SysMenu::getParentId, SysMenu::getOrderNum);

menuList = menuMapper.selectList(queryWrapper);

}

List<Map<String, Object>> treeList = buildMenuTree(menuList);

log.info("查询菜单树成功，数量：{}", treeList.size());

return Result.success(treeList);

} catch (Exception e) {

log.error("获取菜单树失败", e);

return Result.error("获取菜单树失败：" + e.getMessage());

}

}

@Operation(summary = "获取角色菜单树", description = "获取特定角色的菜单树及选中状态")

@GetMapping("/roleMenuTree/{roleId}")

@PreAuthorize("hasAuthority('system:role:edit')")

public Result<Map<String, Object>> getRoleMenuTree(@PathVariable @Parameter(description = "角色ID") Long roleId) {

log.info("获取角色菜单树，角色ID：{}", roleId);

try {

List<SysMenu> allMenus = menuMapper.selectMenuTreeAll();

if (allMenus == null || allMenus.isEmpty()) {

LambdaQueryWrapper<SysMenu> queryWrapper = new LambdaQueryWrapper<>();

queryWrapper.orderByAsc(SysMenu::getParentId, SysMenu::getOrderNum);

allMenus = menuMapper.selectList(queryWrapper);

}

List<Long> checkedKeys = menuMapper.selectMenuListByRoleId(roleId);

if (checkedKeys == null) {

checkedKeys = new ArrayList<>();

}

List<Map<String, Object>> menuTree = buildMenuTree(allMenus);

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

result.put("menus", menuTree);

result.put("checkedKeys", checkedKeys);

log.info("获取角色菜单树成功，角色ID：{}", roleId);

return Result.success(result);

} catch (Exception e) {

log.error("获取角色菜单树失败", e);

return Result.error("获取角色菜单树失败：" + e.getMessage());

}

}

private List<Map<String, Object>> buildMenuTree(List<SysMenu> menuList) {

Map<Long, List<SysMenu>> parentMap = menuList.stream()

.collect(Collectors.groupingBy(SysMenu::getParentId));

List<SysMenu> rootMenus = parentMap.getOrDefault(0L, new ArrayList<>());

return rootMenus.stream()

.map(menu -> buildTreeNode(menu, parentMap))

.collect(Collectors.toList());

}

private Map<String, Object> buildTreeNode(SysMenu menu, Map<Long, List<SysMenu>> parentMap) {

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

node.put("id", menu.getId());

node.put("menuId", menu.getId());

node.put("label", menu.getMenuName());

node.put("menuName", menu.getMenuName());

node.put("icon", menu.getIcon() != null && !menu.getIcon().equals("#") ? menu.getIcon() : "");

node.put("menuType", menu.getMenuType());

node.put("parentId", menu.getParentId());

node.put("orderNum", menu.getOrderNum());

node.put("path", menu.getPath());

node.put("component", menu.getComponent());

node.put("perms", menu.getPerms());

node.put("visible", menu.getVisible());

node.put("status", menu.getStatus());

List<SysMenu> children = parentMap.getOrDefault(menu.getId(), new ArrayList<>());

if (!children.isEmpty()) {

node.put("children", children.stream()

.map(child -> buildTreeNode(child, parentMap))

.collect(Collectors.toList()));

}

return node;

}

private boolean isChildOfMenu(Long id, Long parentId) {

if (parentId == null || parentId == 0L) {

return false;

}

if (id.equals(parentId)) {

return true;

}

SysMenu parentMenu = menuMapper.selectById(parentId);

if (parentMenu == null) {

return false;

}

return isChildOfMenu(id, parentMenu.getParentId());

}

private void validateMenu(SysMenu menu, boolean isAdd) {

if (menu.getMenuName() == null || menu.getMenuName().isEmpty()) {

throw new IllegalArgumentException("菜单名称不能为空");

}

if (menu.getMenuType() == null || menu.getMenuType().isEmpty()) {

throw new IllegalArgumentException("菜单类型不能为空");

}

if ("F".equals(menu.getMenuType()) && (menu.getPerms() == null || menu.getPerms().isEmpty())) {

throw new IllegalArgumentException("按钮类型菜单必须设置权限标识");

}

if (("M".equals(menu.getMenuType()) || "C".equals(menu.getMenuType()))

&& menu.getPath() != null && !menu.getPath().isEmpty() && !menu.getPath().startsWith("/")) {

menu.setPath("/" + menu.getPath());

}

if (menu.getParentId() != null && menu.getParentId() != 0) {

SysMenu parentMenu = menuMapper.selectById(menu.getParentId());

if (parentMenu == null) {

throw new IllegalArgumentException("父菜单不存在");

}

if ("F".equals(parentMenu.getMenuType())) {

throw new IllegalArgumentException("不能将按钮类型菜单设置为父菜单");

}

}

}

}

package com.example.student.controller;

import com.example.student.common.PageResult;

import com.example.student.common.Result;

import com.example.student.util.SecurityUtils;

import com.example.student.dto.SysUserDTO;

import com.example.student.dto.UpdatePasswordDTO;

import com.example.student.dto.UserQueryDTO;

import com.example.student.service.SysUserService;

import com.example.student.vo.SysUserVO;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.validation.Valid;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.util.List;

import java.util.Map;

/\*\*

\* 系统用户控制器

\*

\* @author example

\*/

@Tag(name = "用户管理", description = "用户管理相关接口")

@RestController

@RequestMapping("/system/user")

public class SysUserController {

@Resource

private SysUserService sysUserService;

/\*\*

\* 分页查询用户列表

\*

\* @param queryDTO 查询条件

\* @return 用户分页数据

\*/

@Operation(summary = "分页查询用户列表", description = "分页查询用户列表接口")

@PreAuthorize("@ss.hasPermission('system:user:list')")

@GetMapping("/list")

public Result<PageResult<SysUserVO>> list(UserQueryDTO queryDTO) {

PageResult<SysUserVO> pageResult = sysUserService.pageUser(queryDTO);

return Result.success(pageResult);

}

/\*\*

\* 获取用户详情

\*

\* @param userId 用户ID

\* @return 用户信息

\*/

@Operation(summary = "获取用户详情", description = "获取用户详情接口")

@PreAuthorize("@ss.hasPermission('system:user:query')")

@GetMapping("/{userId}")

public Result<SysUserVO> getInfo(@Parameter(description = "用户ID") @PathVariable Long userId) {

SysUserVO user = sysUserService.getUserById(userId);

return Result.success(user);

}

/\*\*

\* 新增用户

\*

\* @param userDTO 用户信息

\* @return 结果

\*/

@Operation(summary = "新增用户", description = "新增用户接口")

@PreAuthorize("@ss.hasPermission('system:user:add')")

@PostMapping

public Result<Boolean> add(@Valid @RequestBody SysUserDTO userDTO) {

boolean result = sysUserService.addUser(userDTO);

return Result.success(result);

}

/\*\*

\* 修改用户

\*

\* @param userDTO 用户信息

\* @return 结果

\*/

@Operation(summary = "修改用户", description = "修改用户接口")

@PreAuthorize("@ss.hasPermission('system:user:edit')")

@PutMapping

public Result<Boolean> update(@Valid @RequestBody SysUserDTO userDTO) {

boolean result = sysUserService.updateUser(userDTO);

return Result.success(result);

}

/\*\*

\* 删除用户

\*

\* @param userId 用户ID

\* @return 结果

\*/

@Operation(summary = "删除用户", description = "删除用户接口")

@PreAuthorize("@ss.hasPermission('system:user:delete')")

@DeleteMapping("/{userId}")

public Result<Boolean> delete(@Parameter(description = "用户ID") @PathVariable Long userId) {

boolean result = sysUserService.deleteUser(userId);

return Result.success(result);

}

/\*\*

\* 修改用户状态

\*

\* @param userDTO 用户状态

\* @return 结果

\*/

@Operation(summary = "修改用户状态", description = "修改用户状态接口")

@PreAuthorize("@ss.hasPermission('system:user:edit')")

@PutMapping("/status")

public Result<Boolean> changeStatus(@RequestBody SysUserDTO userDTO) {

boolean result = sysUserService.changeUserStatus(userDTO.getId(), userDTO.getStatus());

return Result.success(result);

}

/\*\*

\* 重置用户密码

\*

\* @param userId 用户ID

\* @param userDTO 包含新密码的用户DTO

\* @return 结果

\*/

@Operation(summary = "重置用户密码", description = "重置用户密码接口")

@PreAuthorize("@ss.hasPermission('system:user:resetPwd')")

@PutMapping("/{userId}/password")

public Result<Boolean> resetPassword(

@Parameter(description = "用户ID") @PathVariable Long userId,

@RequestBody SysUserDTO userDTO) {

boolean result = sysUserService.resetUserPassword(userId, userDTO.getPassword());

return Result.success(result);

}

/\*\*

\* 修改用户密码

\*

\* @param passwordDTO 密码信息

\* @return 结果

\*/

@Operation(summary = "修改用户密码", description = "修改用户密码接口")

@PutMapping("/password")

public Result<Boolean> updatePassword(@Valid @RequestBody UpdatePasswordDTO passwordDTO) {

Long userId = SecurityUtils.getUserId();

boolean result = sysUserService.updateUserPassword(userId, passwordDTO);

return Result.success(result);

}

/\*\*

\* 更新用户个人信息

\*

\* @param userDTO 用户信息

\* @return 结果

\*/

@Operation(summary = "更新用户个人信息", description = "更新用户个人信息接口")

@PutMapping("/profile")

public Result<Boolean> updateProfile(@Valid @RequestBody SysUserDTO userDTO) {

Long userId = SecurityUtils.getUserId();

boolean result = sysUserService.updateUserProfile(userId, userDTO);

return Result.success(result);

}

/\*\*

\* 上传用户头像

\*

\* @param file 头像文件

\* @return 头像URL

\*/

@Operation(summary = "上传用户头像", description = "上传用户头像接口")

@PostMapping("/profile/avatar")

public Result<String> uploadAvatar(@RequestParam("file") MultipartFile file) {

Long userId = SecurityUtils.getUserId();

String avatarUrl = sysUserService.uploadAvatar(userId, file);

return Result.success(avatarUrl);

}

/\*\*

\* 获取用户角色ID列表

\*

\* @param userId 用户ID

\* @return 角色ID列表

\*/

@Operation(summary = "获取用户角色ID列表", description = "获取用户角色ID列表接口")

@PreAuthorize("@ss.hasPermission('system:user:query')")

@GetMapping("/{userId}/roles")

public Result<List<Long>> getUserRoles(@Parameter(description = "用户ID") @PathVariable Long userId) {

List<Long> roleIds = sysUserService.getUserRoleIds(userId);

return Result.success(roleIds);

}

/\*\*

\* 设置用户角色

\*

\* @param userId 用户ID

\* @param roleIds 角色ID列表

\* @return 结果

\*/

@Operation(summary = "设置用户角色", description = "设置用户角色接口")

@PreAuthorize("@ss.hasPermission('system:user:edit')")

@PutMapping("/roles")

public Result<Boolean> setUserRoles(@RequestBody Map<String, Object> params) {

Long userId = Long.valueOf(params.get("userId").toString());

@SuppressWarnings("unchecked")

List<Long> roleIds = (List<Long>) params.get("roleIds");

boolean result = sysUserService.setUserRoles(userId, roleIds);

return Result.success(result);

}

/\*\*

\* 导出用户数据

\*

\* @param queryDTO 查询条件

\* @param response HTTP响应

\*/

@Operation(summary = "导出用户数据", description = "导出用户数据接口")

@PreAuthorize("@ss.hasPermission('system:user:export')")

@GetMapping("/export")

public void export(UserQueryDTO queryDTO, HttpServletResponse response) throws IOException {

sysUserService.exportUser(queryDTO, response);

}

}

package com.example.student.controller;

import com.example.student.common.PageResult;

import com.example.student.common.Result;

import com.example.student.dto.TeacherDTO;

import com.example.student.dto.TeacherQueryDTO;

import com.example.student.service.TeacherService;

import com.example.student.vo.TeacherVO;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

import jakarta.annotation.Resource;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.validation.Valid;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.util.Map;

/\*\*

\* 教师控制器

\*

\* @author example

\*/

@Tag(name = "教师管理", description = "教师管理相关接口")

@RestController

@RequestMapping("/teacher")

public class TeacherController {

@Resource

private TeacherService teacherService;

/\*\*

\* 分页查询教师

\*

\* @param queryDTO 查询条件

\* @return 教师分页数据

\*/

@Operation(summary = "分页查询教师", description = "分页查询教师接口")

@PreAuthorize("@ss.hasPermission('education:teacher:list')")

@PostMapping("/list")

public Result<PageResult<TeacherVO>> list(@Valid @RequestBody TeacherQueryDTO queryDTO) {

PageResult<TeacherVO> pageResult = teacherService.pageTeacher(queryDTO);

return Result.success(pageResult);

}

/\*\*

\* 获取教师详情

\*

\* @param id 教师ID

\* @return 教师信息

\*/

@Operation(summary = "获取教师详情", description = "获取教师详情接口")

@PreAuthorize("@ss.hasPermission('education:teacher:query')")

@GetMapping("/{id}")

public Result<TeacherVO> getInfo(@Parameter(description = "教师ID") @PathVariable Long id) {

TeacherVO teacher = teacherService.getTeacherById(id);

return Result.success(teacher);

}

/\*\*

\* 新增教师

\*

\* @param teacherDTO 教师信息

\* @return 结果

\*/

@Operation(summary = "新增教师", description = "新增教师接口")

@PreAuthorize("@ss.hasPermission('education:teacher:add')")

@PostMapping

public Result<Boolean> add(@Valid @RequestBody TeacherDTO teacherDTO) {

boolean result = teacherService.addTeacher(teacherDTO);

return Result.success(result);

}

/\*\*

\* 修改教师

\*

\* @param teacherDTO 教师信息

\* @return 结果

\*/

@Operation(summary = "修改教师", description = "修改教师接口")

@PreAuthorize("@ss.hasPermission('education:teacher:edit')")

@PutMapping

public Result<Boolean> update(@Valid @RequestBody TeacherDTO teacherDTO) {

boolean result = teacherService.updateTeacher(teacherDTO);

return Result.success(result);

}

/\*\*

\* 删除教师

\*

\* @param id 教师ID

\* @return 结果

\*/

@Operation(summary = "删除教师", description = "删除教师接口")

@PreAuthorize("@ss.hasPermission('education:teacher:delete')")

@DeleteMapping("/{id}")

public Result<Boolean> delete(@Parameter(description = "教师ID") @PathVariable Long id) {

boolean result = teacherService.deleteTeacher(id);

return Result.success(result);

}

/\*\*

\* 重置密码

\*

\* @param id 教师ID

\* @return 结果

\*/

@Operation(summary = "重置密码", description = "重置密码接口")

@PreAuthorize("@ss.hasPermission('education:teacher:resetPwd')")

@PutMapping("/resetPassword/{id}")

public Result<Boolean> resetPassword(@Parameter(description = "教师ID") @PathVariable Long id) {

boolean result = teacherService.resetPassword(id);

return Result.success(result);

}

/\*\*

\* 更新状态

\*

\* @param id 教师ID

\* @param status 状态

\* @return 结果

\*/

@Operation(summary = "更新状态", description = "更新状态接口")

@PreAuthorize("@ss.hasPermission('education:teacher:edit')")

@PutMapping("/status/{id}/{status}")

public Result<Boolean> updateStatus(

@Parameter(description = "教师ID") @PathVariable Long id,

@Parameter(description = "状态") @PathVariable Integer status) {

boolean result = teacherService.updateStatus(id, status);

return Result.success(result);

}

/\*\*

\* 获取教师课程列表

\*

\* @param teacherId 教师ID

\* @param queryDTO 查询条件

\* @return 课程分页数据

\*/

@Operation(summary = "获取教师课程列表", description = "获取教师课程列表接口")

@PreAuthorize("@ss.hasAnyRole('admin', 'teacher')")

@PostMapping("/courses/{teacherId}")

public Result<PageResult<?>> getCourses(

@Parameter(description = "教师ID") @PathVariable Long teacherId,

@Valid @RequestBody TeacherQueryDTO queryDTO) {

PageResult<?> pageResult = teacherService.getTeacherCourses(teacherId, queryDTO);

return Result.success(pageResult);

}

/\*\*

\* 获取教师课程学生列表

\*

\* @param teacherId 教师ID

\* @param courseId 课程ID

\* @param queryDTO 查询条件

\* @return 学生分页数据

\*/

@Operation(summary = "获取教师课程学生列表", description = "获取教师课程学生列表接口")

@PreAuthorize("@ss.hasAnyRole('admin', 'teacher')")

@PostMapping("/courses/{teacherId}/students/{courseId}")

public Result<PageResult<?>> getCourseStudents(

@Parameter(description = "教师ID") @PathVariable Long teacherId,

@Parameter(description = "课程ID") @PathVariable Long courseId,

@Valid @RequestBody TeacherQueryDTO queryDTO) {

PageResult<?> pageResult = teacherService.getTeacherCourseStudents(teacherId, courseId, queryDTO);

return Result.success(pageResult);

}

/\*\*

\* 获取课程学生列表（支持管理员访问）

\*

\* @param courseId 课程ID

\* @param queryDTO 查询条件，包含可选的teacherId

\* @return 学生分页数据

\*/

@Operation(summary = "获取课程学生列表", description = "获取课程学生列表接口，支持管理员直接访问")

@PreAuthorize("@ss.hasAnyRole('admin', 'teacher')")

@PostMapping("/courses/students/{courseId}")

public Result<PageResult<?>> getCourseStudentsNew(

@Parameter(description = "课程ID") @PathVariable Long courseId,

@Valid @RequestBody TeacherQueryDTO queryDTO) {

// 从查询DTO中获取teacherId

Long teacherId = queryDTO.getTeacherId();

// 调用相同的服务方法，处理逻辑不变

PageResult<?> pageResult = teacherService.getTeacherCourseStudents(teacherId, courseId, queryDTO);

return Result.success(pageResult);

}

/\*\*

\* 提交学生成绩

\*

\* @param teacherId 教师ID

\* @param courseId 课程ID

\* @param data 成绩数据

\* @return 结果

\*/

@Operation(summary = "提交学生成绩", description = "提交学生成绩接口")

@PreAuthorize("@ss.hasAnyRole('admin', 'teacher')")

@PutMapping("/courses/{teacherId}/grade/{courseId}")

public Result<Boolean> submitGrade(

@Parameter(description = "教师ID") @PathVariable Long teacherId,

@Parameter(description = "课程ID") @PathVariable Long courseId,

@RequestBody Map<String, Object> data) {

boolean result = teacherService.submitStudentGrade(teacherId, courseId, data);

return Result.success(result);

}

/\*\*

\* 提交学生考勤

\*

\* @param teacherId 教师ID

\* @param courseId 课程ID

\* @param data 考勤数据

\* @return 结果

\*/

@Operation(summary = "提交学生考勤", description = "提交学生考勤接口")

@PreAuthorize("@ss.hasAnyRole('admin', 'teacher')")

@PutMapping("/courses/{teacherId}/attendance/{courseId}")

public Result<Boolean> submitAttendance(

@Parameter(description = "教师ID") @PathVariable Long teacherId,

@Parameter(description = "课程ID") @PathVariable Long courseId,

@RequestBody Map<String, Object> data) {

boolean result = teacherService.submitStudentAttendance(teacherId, courseId, data);

return Result.success(result);

}

/\*\*

\* 管理员提交学生考勤（无教师ID）

\*

\* @param courseId 课程ID

\* @param data 考勤数据

\* @return 操作结果

\*/

@Operation(summary = "管理员提交学生考勤", description = "管理员提交学生考勤接口")

@PreAuthorize("@ss.hasAnyRole('admin')")

@PutMapping("/courses/attendance/{courseId}")

public Result<Boolean> submitAttendanceByAdmin(

@Parameter(description = "课程ID") @PathVariable Long courseId,

@RequestBody Map<String, Object> data) {

boolean result = teacherService.submitStudentAttendance(null, courseId, data);

return Result.success(result);

}

/\*\*

\* 获取教师统计数据

\*

\* @param teacherId 教师ID

\* @return 统计数据

\*/

@Operation(summary = "获取教师统计数据", description = "获取教师统计数据接口")

@PreAuthorize("@ss.hasAnyRole('admin', 'teacher')")

@GetMapping("/statistics/{teacherId}")

public Result<Map<String, Object>> getStatistics(

@Parameter(description = "教师ID") @PathVariable Long teacherId) {

Map<String, Object> statistics = teacherService.getTeacherStatistics(teacherId);

return Result.success(statistics);

}

/\*\*

\* 导入教师数据

\*

\* @param file 教师数据文件

\* @return 导入结果

\* @throws IOException IO异常

\*/

@Operation(summary = "导入教师数据", description = "导入教师数据接口")

@PreAuthorize("@ss.hasPermission('education:teacher:import')")

@PostMapping("/import")

public Result<Map<String, Object>> importData(@RequestParam("file") MultipartFile file) throws IOException {

Map<String, Object> result = teacherService.importTeacher(file);

return Result.success(result);

}

/\*\*

\* 导出教师数据

\*

\* @param queryDTO 查询条件

\* @param response HTTP响应

\* @throws IOException IO异常

\*/

@Operation(summary = "导出教师数据", description = "导出教师数据接口")

@PreAuthorize("@ss.hasPermission('education:teacher:export')")

@PostMapping("/export")

public void exportData(@Valid @RequestBody TeacherQueryDTO queryDTO, HttpServletResponse response) throws IOException {

teacherService.exportTeacher(queryDTO, response);

}

/\*\*

\* 下载教师导入模板

\*

\* @param response HTTP响应

\* @throws IOException IO异常

\*/

@Operation(summary = "下载教师导入模板", description = "下载教师导入模板接口")

@PreAuthorize("@ss.hasPermission('education:teacher:import')")

@GetMapping("/template")

public void downloadTemplate(HttpServletResponse response) throws IOException {

teacherService.downloadTeacherTemplate(response);

}

}

package com.example.student.controller;

import com.example.student.common.Result;

import com.example.student.entity.TodoItem;

import com.example.student.entity.SysUser;

import com.example.student.service.TodoItemService;

import com.example.student.service.SysUserService;

import lombok.RequiredArgsConstructor;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.security.core.Authentication;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

/\*\*

\* 待办事项Controller

\*/

@RestController

@RequestMapping("/todo")

@RequiredArgsConstructor

public class TodoItemController {

private final TodoItemService todoItemService;

private final SysUserService userService;

/\*\*

\* 查询待办事项列表

\*/

@GetMapping("/list")

public Result<List<TodoItem>> list(TodoItem todoItem, Authentication authentication) {

// 普通用户只能查看自己的待办事项

SysUser user = userService.findByUsername(authentication.getName());

if (!userService.isAdmin(user)) {

todoItem.setUserId(user.getId());

}

List<TodoItem> list = todoItemService.selectTodoItemList(todoItem);

return Result.success(list);

}

/\*\*

\* 获取当前用户的待办事项列表

\*/

@GetMapping("/my")

public Result<List<TodoItem>> myTodoList(Authentication authentication) {

SysUser user = userService.findByUsername(authentication.getName());

List<TodoItem> list = todoItemService.selectTodoItemsByUserId(user.getId());

return Result.success(list);

}

/\*\*

\* 获取待办事项详细信息

\*/

@GetMapping("/{id}")

public Result<TodoItem> getInfo(@PathVariable("id") Long id, Authentication authentication) {

TodoItem todoItem = todoItemService.selectTodoItemById(id);

// 权限检查：普通用户只能查看自己的待办事项

SysUser user = userService.findByUsername(authentication.getName());

if (!userService.isAdmin(user) && !todoItem.getUserId().equals(user.getId())) {

return Result.error("无权访问此待办事项");

}

return Result.success(todoItem);

}

/\*\*

\* 新增待办事项

\*/

@PostMapping

public Result<Integer> add(@RequestBody TodoItem todoItem, Authentication authentication) {

// 如果未指定用户ID，则默认为当前用户

if (todoItem.getUserId() == null) {

SysUser user = userService.findByUsername(authentication.getName());

todoItem.setUserId(user.getId());

} else {

// 权限检查：普通用户只能为自己创建待办事项

SysUser user = userService.findByUsername(authentication.getName());

if (!userService.isAdmin(user) && !todoItem.getUserId().equals(user.getId())) {

return Result.error("无权为其他用户创建待办事项");

}

}

return Result.success(todoItemService.insertTodoItem(todoItem));

}

/\*\*

\* 修改待办事项

\*/

@PutMapping

public Result<Integer> edit(@RequestBody TodoItem todoItem, Authentication authentication) {

// 权限检查：普通用户只能修改自己的待办事项

SysUser user = userService.findByUsername(authentication.getName());

TodoItem original = todoItemService.selectTodoItemById(todoItem.getId());

if (original == null) {

return Result.error("待办事项不存在");

}

if (!userService.isAdmin(user) && !original.getUserId().equals(user.getId())) {

return Result.error("无权修改此待办事项");

}

return Result.success(todoItemService.updateTodoItem(todoItem));

}

/\*\*

\* 删除待办事项

\*/

@DeleteMapping("/{id}")

public Result<Integer> remove(@PathVariable Long id, Authentication authentication) {

// 权限检查：普通用户只能删除自己的待办事项

SysUser user = userService.findByUsername(authentication.getName());

TodoItem todoItem = todoItemService.selectTodoItemById(id);

if (todoItem == null) {

return Result.error("待办事项不存在");

}

if (!userService.isAdmin(user) && !todoItem.getUserId().equals(user.getId())) {

return Result.error("无权删除此待办事项");

}

return Result.success(todoItemService.deleteTodoItemById(id));

}

/\*\*

\* 批量删除待办事项

\*/

@DeleteMapping("/batch/{ids}")

@PreAuthorize("hasAuthority('system:todo:remove')")

public Result<Integer> removeBatch(@PathVariable Long[] ids) {

return Result.success(todoItemService.deleteTodoItemByIds(ids));

}

/\*\*

\* 完成待办事项

\*/

@PutMapping("/complete/{id}")

public Result<Integer> complete(@PathVariable Long id, Authentication authentication) {

// 权限检查：普通用户只能完成自己的待办事项

SysUser user = userService.findByUsername(authentication.getName());

TodoItem todoItem = todoItemService.selectTodoItemById(id);

if (todoItem == null) {

return Result.error("待办事项不存在");

}

if (!userService.isAdmin(user) && !todoItem.getUserId().equals(user.getId())) {

return Result.error("无权操作此待办事项");

}

return Result.success(todoItemService.completeTodoItem(id));

}

/\*\*

\* 获取未完成的待办事项数量

\*/

@GetMapping("/count/uncompleted")

public Result<Integer> countUncompleted(Authentication authentication) {

SysUser user = userService.findByUsername(authentication.getName());

int count = todoItemService.countUncompletedTodoItems(user.getId());

return Result.success(count);

}

/\*\*

\* 获取即将到期的待办事项

\*/

@GetMapping("/upcoming/{days}")

public Result<List<TodoItem>> upcomingTodos(@PathVariable("days") int days, Authentication authentication) {

SysUser user = userService.findByUsername(authentication.getName());

List<TodoItem> list = todoItemService.selectUpcomingTodoItems(user.getId(), days);

return Result.success(list);

}

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import lombok.Data;

/\*\*

\* 管理员数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "管理员数据传输对象")

public class AdminDTO {

@Schema(description = "管理员ID")

private Long id;

@Schema(description = "用户名")

@NotBlank(message = "用户名不能为空")

@Pattern(regexp = "^[a-zA-Z0-9]{4,16}$", message = "用户名必须为4-16位字母或数字")

private String username;

@Schema(description = "密码")

@NotBlank(message = "密码不能为空")

private String password;

@Schema(description = "姓名")

@NotBlank(message = "姓名不能为空")

private String name;

@Schema(description = "头像")

private String avatar;

@Schema(description = "邮箱")

@Email(message = "邮箱格式不正确")

private String email;

@Schema(description = "手机号")

@Pattern(regexp = "^1[3-9]\\d{9}$", message = "手机号格式不正确")

private String phone;

@Schema(description = "状态 0:正常 1:禁用")

private Integer status;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import lombok.Data;

/\*\*

\* 管理员数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "管理员数据传输对象")

public class AdminDTO {

@Schema(description = "管理员ID")

private Long id;

@Schema(description = "用户名")

@NotBlank(message = "用户名不能为空")

@Pattern(regexp = "^[a-zA-Z0-9]{4,16}$", message = "用户名必须为4-16位字母或数字")

private String username;

@Schema(description = "密码")

@NotBlank(message = "密码不能为空")

private String password;

@Schema(description = "姓名")

@NotBlank(message = "姓名不能为空")

private String name;

@Schema(description = "头像")

private String avatar;

@Schema(description = "邮箱")

@Email(message = "邮箱格式不正确")

private String email;

@Schema(description = "手机号")

@Pattern(regexp = "^1[3-9]\\d{9}$", message = "手机号格式不正确")

private String phone;

@Schema(description = "状态 0:正常 1:禁用")

private Integer status;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

import java.math.BigDecimal;

import java.util.List;

/\*\*

\* 课程数据传输对象

\*/

@Data

@Schema(description = "课程数据传输对象")

public class CourseDTO {

@Schema(description = "课程ID")

private Long id;

@NotBlank(message = "课程编号不能为空")

@Pattern(regexp = "^[A-Za-z0-9-]+$", message = "课程编号只能包含字母、数字和连字符")

@Size(max = 30, message = "课程编号长度不能超过30个字符")

@Schema(description = "课程编号", requiredMode = Schema.RequiredMode.REQUIRED)

private String code;

@NotBlank(message = "课程名称不能为空")

@Size(max = 100, message = "课程名称长度不能超过100个字符")

@Schema(description = "课程名称", requiredMode = Schema.RequiredMode.REQUIRED)

private String name;

@NotNull(message = "学分不能为空")

@Schema(description = "学分", requiredMode = Schema.RequiredMode.REQUIRED)

private BigDecimal credit;

@NotNull(message = "学时不能为空")

@Schema(description = "学时", requiredMode = Schema.RequiredMode.REQUIRED)

private Integer hours;

@NotNull(message = "课程类型不能为空")

@Schema(description = "课程类型 0:必修 1:选修", requiredMode = Schema.RequiredMode.REQUIRED)

private Integer type;

@NotNull(message = "开课院系不能为空")

@Schema(description = "开课院系ID", requiredMode = Schema.RequiredMode.REQUIRED)

private Long departmentId;

@Size(max = 500, message = "课程描述长度不能超过500个字符")

@Schema(description = "课程描述")

private String description;

@Schema(description = "适用专业ID列表")

private List<Long> majorIds;

@Schema(description = "推荐学期 (例如: 1-1, 2-2)")

private String recommendSemester;

@Schema(description = "先修课程ID列表")

private List<Long> prerequisites;

@Schema(description = "课程难度 (1-5)")

private Integer difficulty;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.NotNull;

import lombok.Data;

import java.io.Serializable;

@Data

@Schema(description = "课程开设DTO")

public class CourseOfferingDTO implements Serializable {

@Schema(description = "主键ID")

private Long id;

@NotNull(message = "课程ID不能为空")

@Schema(description = "课程ID")

private Long courseId;

@NotNull(message = "教师ID不能为空")

@Schema(description = "教师ID")

private Long teacherId;

@Schema(description = "班级ID")

private Long classId;

@NotNull(message = "学期不能为空")

@Schema(description = "学期")

private String semester;

@NotNull(message = "上课时间不能为空")

@Schema(description = "上课时间")

private String classTime;

@NotNull(message = "上课地点不能为空")

@Schema(description = "上课地点")

private String location;

@NotNull(message = "容量不能为空")

@Schema(description = "容量")

private Integer capacity;

@Schema(description = "已选人数")

private Integer selectedCount;

@Schema(description = "状态 0:正常 1:已结课")

private Integer status;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

@Data

@Schema(description = "课程开设分页查询DTO")

public class CourseOfferingQueryDTO {

@Schema(description = "课程名称关键字")

private String courseName;

@Schema(description = "教师名称关键字")

private String teacherName;

@Schema(description = "学期")

private String semester;

@Schema(description = "页码")

private Integer pageNum = 1;

@Schema(description = "每页大小")

private Integer pageSize = 10;

}

package com.example.student.dto;

import com.example.student.common.PageRequest;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import lombok.EqualsAndHashCode;

import java.math.BigDecimal;

/\*\*

\* 课程查询数据传输对象

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@Schema(description = "课程查询数据传输对象")

public class CourseQueryDTO extends PageRequest {

@Schema(description = "课程编号")

private String code;

@Schema(description = "课程名称")

private String name;

@Schema(description = "课程类型 0:必修 1:选修")

private Integer type;

@Schema(description = "开课院系ID")

private Long departmentId;

@Schema(description = "学分下限")

private BigDecimal creditMin;

@Schema(description = "学分上限")

private BigDecimal creditMax;

@Schema(description = "学时下限")

private Integer hoursMin;

@Schema(description = "学时上限")

private Integer hoursMax;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import jakarta.validation.constraints.NotNull;

import java.math.BigDecimal;

import java.time.LocalDateTime;

/\*\*

\* 选课数据传输对象

\*/

@Data

@Schema(description = "选课信息")

public class CourseSelectionDTO {

/\*\*

\* 选课ID

\*/

@Schema(description = "选课ID")

@NotNull(message = "选课ID不能为空")

private Long id;

/\*\*

\* 学生ID

\*/

@Schema(description = "学生ID")

private Long studentId;

/\*\*

\* 课程开设ID

\*/

@Schema(description = "课程开设ID")

private Long courseOfferingId;

/\*\*

\* 选课状态：0-已选，1-已退，2-已修完

\*/

@Schema(description = "选课状态：0-已选，1-已退，2-已修完")

private Integer status;

/\*\*

\* 选课时间

\*/

@Schema(description = "选课时间")

private LocalDateTime selectionTime;

@Schema(description = "分数")

private BigDecimal score;

/\*\*

\* 兼容方法 - 设置选课状态

\*/

public void setSelectionStatus(Integer selectionStatus) {

this.status = selectionStatus;

}

/\*\*

\* 兼容方法 - 获取选课状态

\*/

public Integer getSelectionStatus() {

return this.status;

}

}

package com.example.student.dto;

import lombok.Data;

/\*\*

\* 选课查询数据传输对象

\*/

@Data

public class CourseSelectionQueryDTO {

/\*\*

\* 当前页码

\*/

private Integer pageNum = 1;

/\*\*

\* 每页大小

\*/

private Integer pageSize = 10;

/\*\*

\* 学生学号

\*/

private String studentNo;

/\*\*

\* 学生姓名

\*/

private String studentName;

/\*\*

\* 课程名称

\*/

private String courseName;

/\*\*

\* 学期

\*/

private String semester;

/\*\*

\* 选课状态

\*/

private Integer status;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

/\*\*

\* 成绩查询DTO

\*

\* @author example

\*/

@Data

@Schema(description = "成绩查询参数")

public class GradeQueryDTO {

@Schema(description = "学期")

private String semester;

@Schema(description = "课程ID")

private Long courseId;

@Schema(description = "班级ID")

private Long classId;

@Schema(description = "课程开设ID")

private Long courseOfferingId;

@Schema(description = "页码", defaultValue = "1")

private Integer pageNum = 1;

@Schema(description = "每页条数", defaultValue = "10")

private Integer pageSize = 10;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.NotBlank;

import lombok.Data;

/\*\*

\* 用户登录请求

\*

\* @author example

\*/

@Data

@Schema(description = "用户登录请求")

public class LoginDTO {

@NotBlank(message = "用户名不能为空")

@Schema(description = "用户名", example = "admin")

private String username;

@NotBlank(message = "密码不能为空")

@Schema(description = "密码", example = "123456")

private String password;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.util.List;

/\*\*

\* 用户登录响应

\*

\* @author example

\*/

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor

@Schema(description = "用户登录响应")

public class LoginResponseDTO {

@Schema(description = "用户ID")

private Long userId;

@Schema(description = "用户名")

private String username;

@Schema(description = "姓名")

private String name;

@Schema(description = "头像")

private String avatar;

@Schema(description = "用户类型 0:管理员 1:教师 2:学生")

private Integer userType;

@Schema(description = "角色列表")

private List<String> roles;

@Schema(description = "权限列表")

private List<String> permissions;

@Schema(description = "token")

private String token;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Pattern;

import lombok.Data;

import java.time.LocalDate;

/\*\*

\* 学生数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "学生数据传输对象")

public class StudentDTO {

@Schema(description = "学生ID")

private Long id;

@NotBlank(message = "学号不能为空")

@Schema(description = "学号", example = "S202001001")

private String studentNo;

@NotBlank(message = "姓名不能为空")

@Schema(description = "姓名", example = "张三")

private String name;

@NotNull(message = "班级ID不能为空")

@Schema(description = "班级ID", example = "1")

private Long classId;

@Schema(description = "班级名称", example = "计算机科学2020-1班")

private String className;

@NotNull(message = "入学日期不能为空")

@Schema(description = "入学日期", example = "2020-09-01")

private LocalDate admissionDate;

@Schema(description = "毕业日期", example = "2024-06-30")

private LocalDate graduationDate;

@NotNull(message = "性别不能为空")

@Schema(description = "性别 0:男 1:女", example = "0")

private Integer gender;

@Pattern(regexp = "(^\\d{15}$)|(^\\d{18}$)|(^\\d{17}(\\d|X|x)$)", message = "身份证号格式不正确")

@Schema(description = "身份证号", example = "110101200001010033")

private String idCard;

@Schema(description = "出生日期", example = "2000-01-01")

private LocalDate birthday;

@Schema(description = "家庭住址", example = "北京市海淀区")

private String address;

@Schema(description = "状态 0:在读 1:毕业 2:休学 3:退学", example = "0")

private Integer status;

@Schema(description = "用户名", example = "student1")

private String username;

@Schema(description = "密码", example = "123456")

private String password;

@Schema(description = "邮箱", example = "student1@example.com")

private String email;

@Schema(description = "手机号", example = "13800000000")

private String phone;

}

package com.example.student.dto;

import com.alibaba.excel.annotation.ExcelProperty;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import java.io.Serializable;

/\*\*

\* 学生导入数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "学生导入数据传输对象")

public class StudentImportDTO implements Serializable {

private static final long serialVersionUID = 1L;

@ExcelProperty("学号")

@Schema(description = "学号", example = "S202001001")

private String studentNo;

@ExcelProperty("姓名")

@Schema(description = "姓名", example = "张三")

private String name;

@ExcelProperty("性别")

@Schema(description = "性别", example = "男")

private String gender;

@ExcelProperty("班级名称")

@Schema(description = "班级名称", example = "计算机科学2020-1班")

private String className;

@ExcelProperty("班级编码")

@Schema(description = "班级编码", example = "CS2001")

private String classCode;

@ExcelProperty("入学日期")

@Schema(description = "入学日期", example = "2020-09-01")

private String admissionDate;

@ExcelProperty("毕业日期")

@Schema(description = "毕业日期", example = "2024-06-30")

private String graduationDate;

@ExcelProperty("身份证号")

@Schema(description = "身份证号", example = "110101200001010033")

private String idCard;

@ExcelProperty("出生日期")

@Schema(description = "出生日期", example = "2000-01-01")

private String birthday;

@ExcelProperty("家庭住址")

@Schema(description = "家庭住址", example = "北京市海淀区")

private String address;

@ExcelProperty("邮箱")

@Schema(description = "邮箱", example = "student1@example.com")

private String email;

@ExcelProperty("手机号")

@Schema(description = "手机号", example = "13800000000")

private String phone;

@ExcelProperty("状态")

@Schema(description = "状态", example = "在读")

private String status;

@Schema(description = "导入错误消息")

private String errorMsg;

}

package com.example.student.dto;

import com.example.student.common.PageRequest;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import lombok.EqualsAndHashCode;

/\*\*

\* 学生查询参数

\*

\* @author example

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@Schema(description = "学生查询参数")

public class StudentQueryDTO extends PageRequest {

@Schema(description = "学号", example = "S2020")

private String studentNo;

@Schema(description = "姓名", example = "张")

private String name;

@Schema(description = "班级ID", example = "1")

private Long classId;

@Schema(description = "性别 0:男 1:女", example = "0")

private Integer gender;

@Schema(description = "专业ID", example = "1")

private Long majorId;

@Schema(description = "院系ID", example = "1")

private Long departmentId;

@Schema(description = "手机号码", example = "13800000000")

private String phone;

@Schema(description = "状态 0:在读 1:毕业 2:休学 3:退学", example = "0")

private Integer status;

@Schema(description = "入学开始日期", example = "2020-09-01")

private String startDate;

@Schema(description = "入学结束日期", example = "2020-09-30")

private String endDate;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

import lombok.Data;

import java.util.List;

/\*\*

\* 系统用户数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "系统用户数据传输对象")

public class SysUserDTO {

@Schema(description = "用户ID")

private Long id;

@NotBlank(message = "用户名不能为空")

@Size(min = 2, max = 20, message = "用户名长度必须在2到20个字符之间")

@Schema(description = "用户名")

private String username;

@Schema(description = "密码")

private String password;

@NotBlank(message = "姓名不能为空")

@Size(min = 2, max = 20, message = "姓名长度必须在2到20个字符之间")

@Schema(description = "姓名")

private String name;

@Schema(description = "头像")

private String avatar;

@Email(message = "邮箱格式不正确")

@Schema(description = "邮箱")

private String email;

@Pattern(regexp = "^1[3-9]\\d{9}$", message = "手机号格式不正确")

@Schema(description = "手机号")

private String phone;

@Schema(description = "状态 0:正常 1:禁用")

private Integer status;

@Schema(description = "用户类型 0:管理员 1:教师 2:学生")

private Integer userType;

@Schema(description = "角色ID列表")

private List<Long> roleIds;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

import lombok.Data;

import java.util.List;

/\*\*

\* 系统用户数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "系统用户数据传输对象")

public class SysUserDTO {

@Schema(description = "用户ID")

private Long id;

@NotBlank(message = "用户名不能为空")

@Size(min = 2, max = 20, message = "用户名长度必须在2到20个字符之间")

@Schema(description = "用户名")

private String username;

@Schema(description = "密码")

private String password;

@NotBlank(message = "姓名不能为空")

@Size(min = 2, max = 20, message = "姓名长度必须在2到20个字符之间")

@Schema(description = "姓名")

private String name;

@Schema(description = "头像")

private String avatar;

@Email(message = "邮箱格式不正确")

@Schema(description = "邮箱")

private String email;

@Pattern(regexp = "^1[3-9]\\d{9}$", message = "手机号格式不正确")

@Schema(description = "手机号")

private String phone;

@Schema(description = "状态 0:正常 1:禁用")

private Integer status;

@Schema(description = "用户类型 0:管理员 1:教师 2:学生")

private Integer userType;

@Schema(description = "角色ID列表")

private List<Long> roleIds;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

import lombok.Data;

import java.util.List;

/\*\*

\* 系统用户数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "系统用户数据传输对象")

public class SysUserDTO {

@Schema(description = "用户ID")

private Long id;

@NotBlank(message = "用户名不能为空")

@Size(min = 2, max = 20, message = "用户名长度必须在2到20个字符之间")

@Schema(description = "用户名")

private String username;

@Schema(description = "密码")

private String password;

@NotBlank(message = "姓名不能为空")

@Size(min = 2, max = 20, message = "姓名长度必须在2到20个字符之间")

@Schema(description = "姓名")

private String name;

@Schema(description = "头像")

private String avatar;

@Email(message = "邮箱格式不正确")

@Schema(description = "邮箱")

private String email;

@Pattern(regexp = "^1[3-9]\\d{9}$", message = "手机号格式不正确")

@Schema(description = "手机号")

private String phone;

@Schema(description = "状态 0:正常 1:禁用")

private Integer status;

@Schema(description = "用户类型 0:管理员 1:教师 2:学生")

private Integer userType;

@Schema(description = "角色ID列表")

private List<Long> roleIds;

}

package com.example.student.dto;

import com.example.student.common.PageRequest;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import lombok.EqualsAndHashCode;

/\*\*

\* 教师查询条件

\*

\* @author example

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@Schema(description = "教师查询条件")

public class TeacherQueryDTO extends PageRequest {

@Schema(description = "教师ID")

private Long teacherId;

@Schema(description = "教师编号")

private String teacherNo;

@Schema(description = "姓名")

private String name;

@Schema(description = "院系ID")

private Long departmentId;

@Schema(description = "职称")

private String title;

@Schema(description = "状态 0:在职 1:离职 2:退休")

private Integer status;

@Schema(description = "学期")

private String semester;

@Schema(description = "课程名称")

private String courseName;

@Schema(description = "学生姓名")

private String studentName;

@Schema(description = "学号")

private String studentNo;

@Schema(description = "学生ID，用于查询单个学生的考勤详情")

private Long studentId;

@Schema(description = "邮箱")

private String email;

@Schema(description = "手机号")

private String phone;

@Schema(description = "出生日期开始")

private String birthdayStart;

@Schema(description = "出生日期结束")

private String birthdayEnd;

@Schema(description = "入职日期开始")

private String entryDateStart;

@Schema(description = "入职日期结束")

private String entryDateEnd;

}

package com.example.student.dto;

import io.swagger.v3.oas.annotations.media.Schema;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

import lombok.Data;

/\*\*

\* 修改密码数据传输对象

\*

\* @author example

\*/

@Data

@Schema(description = "修改密码数据传输对象")

public class UpdatePasswordDTO {

@NotBlank(message = "旧密码不能为空")

@Schema(description = "旧密码")

private String oldPassword;

@NotBlank(message = "新密码不能为空")

@Size(min = 6, max = 20, message = "密码长度必须在6到20个字符之间")

@Schema(description = "新密码")

private String newPassword;

}

package com.example.student.dto;

import com.example.student.common.PageRequest;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import lombok.EqualsAndHashCode;

/\*\*

\* 用户查询条件

\*

\* @author example

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@Schema(description = "用户查询条件")

public class UserQueryDTO extends PageRequest {

@Schema(description = "用户名")

private String username;

@Schema(description = "姓名")

private String name;

@Schema(description = "手机号")

private String phone;

@Schema(description = "状态 0:正常 1:禁用")

private Integer status;

@Schema(description = "用户类型 0:管理员 1:教师 2:学生")

private Integer userType;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

import com.baomidou.mybatisplus.annotation.TableName;

import lombok.Data;

import java.io.Serializable;

import java.time.LocalDate;

import java.time.LocalDateTime;

/\*\*

\* 考勤实体类

\*

\* @author example

\*/

@Data

@TableName("attendance")

public class Attendance implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* ID

\*/

@TableId(value = "id", type = IdType.AUTO)

private Long id;

/\*\*

\* 课程开设ID

\*/

private Long courseOfferingId;

/\*\*

\* 学生ID

\*/

private Long studentId;

/\*\*

\* 考勤日期

\*/

private LocalDate attendanceDate;

/\*\*

\* 状态 0:出勤 1:缺勤 2:迟到 3:早退 4:请假

\*/

private Integer status;

/\*\*

\* 备注

\*/

private String remark;

/\*\*

\* 创建时间

\*/

private LocalDateTime createTime;

/\*\*

\* 更新时间

\*/

private LocalDateTime updateTime;

}

package com.example.student.entity;

import lombok.Data;

import lombok.AllArgsConstructor;

import lombok.NoArgsConstructor;

import lombok.Builder;

import java.util.List;

import java.util.Map;

/\*\*

\* 图表数据实体类

\*/

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class ChartData {

/\*\*

\* 标签数组

\*/

private List<String> labels;

/\*\*

\* 数据数组（单系列）

\*/

private List<Integer> data;

/\*\*

\* 多系列数据（键为系列名称，值为数据数组）

\*/

private Map<String, List<Integer>> seriesData;

/\*\*

\* 多系列浮点数据（键为系列名称，值为浮点数据数组）

\*/

private Map<String, List<Double>> seriesDataDouble;

/\*\*

\* 图表类型

\*/

private String chartType;

/\*\*

\* 标题

\*/

private String title;

/\*\*

\* 单位

\*/

private String unit;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

import com.baomidou.mybatisplus.annotation.TableName;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import java.io.Serializable;

import java.time.LocalDateTime;

/\*\*

\* 班级实体

\*

\* @author example

\*/

@Data

@TableName("class")

@Schema(description = "班级实体")

public class Class implements Serializable {

private static final long serialVersionUID = 1L;

@Schema(description = "班级ID")

@TableId(value = "id", type = IdType.AUTO)

private Long id;

@Schema(description = "班级名称")

private String name;

@Schema(description = "班级编码")

private String code;

@Schema(description = "所属专业ID")

private Long majorId;

@Schema(description = "年级")

private String grade;

@Schema(description = "班主任ID")

private Long advisorId;

@Schema(description = "创建时间")

private LocalDateTime createTime;

@Schema(description = "更新时间")

private LocalDateTime updateTime;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.\*;

import lombok.Data;

import java.math.BigDecimal;

import java.time.LocalDateTime;

/\*\*

\* 课程实体类

\*/

@Data

@TableName("course")

public class Course {

/\*\*

\* 课程ID

\*/

@TableId(value = "id", type = IdType.AUTO)

private Long id;

/\*\*

\* 课程代码

\*/

private String code;

/\*\*

\* 课程名称

\*/

private String name;

/\*\*

\* 课程类型: 0-必修，1-选修

\*/

private Integer type;

/\*\*

\* 学分

\*/

private BigDecimal credit;

/\*\*

\* 教学学时

\*/

private Integer hours;

/\*\*

\* 课程描述

\*/

private String description;

/\*\*

\* 院系ID

\*/

private Long departmentId;

/\*\*

\* 创建时间

\*/

@TableField(fill = FieldFill.INSERT)

private LocalDateTime createTime;

/\*\*

\* 更新时间

\*/

@TableField(fill = FieldFill.INSERT\_UPDATE)

private LocalDateTime updateTime;

/\*\*

\* 教师ID

\*/

@TableField(exist = false)

private Long teacherId;

/\*\*

\* 教师姓名

\*/

@TableField(exist = false)

private String teacherName;

/\*\*

\* 教师联系方式

\*/

@TableField(exist = false)

private String teacherContact;

/\*\*

\* 学期

\*/

@TableField(exist = false)

private String semester;

/\*\*

\* 上课时间

\*/

@TableField(exist = false)

private String classTime;

/\*\*

\* 上课地点

\*/

@TableField(exist = false)

private String location;

/\*\*

\* 课程状态

\*/

@TableField(exist = false)

private Integer status;

/\*\*

\* 状态描述

\*/

@TableField(exist = false)

private String statusDesc;

/\*\*

\* 课程容量

\*/

@TableField(exist = false)

private Integer capacity;

/\*\*

\* 已选人数

\*/

@TableField(exist = false)

private Integer selectedCount;

/\*\*

\* 所属院系名称

\*/

@TableField(exist = false)

private String departmentName;

/\*\*

\* 考核方式

\*/

@TableField(exist = false)

private String assessmentMethod;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.\*;

import lombok.Data;

import java.io.Serializable;

import java.time.LocalDateTime;

/\*\*

\* 课程开设实体类

\*

\* @author example

\*/

@Data

@TableName("course\_offering")

public class CourseOffering implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 开课ID

\*/

@TableId(value = "id", type = IdType.AUTO)

private Long id;

/\*\*

\* 课程ID

\*/

private Long courseId;

/\*\*

\* 教师ID

\*/

private Long teacherId;

/\*\*

\* 学期

\*/

private String semester;

/\*\*

\* 上课时间

\*/

private String classTime;

/\*\*

\* 上课地点

\*/

private String location;

/\*\*

\* 容量

\*/

private Integer capacity;

/\*\*

\* 已选人数

\*/

private Integer selectedCount;

/\*\*

\* 状态 0:正常 1:已结课

\*/

private Integer status;

/\*\*

\* 创建时间

\*/

@TableField(fill = FieldFill.INSERT)

private LocalDateTime createTime;

/\*\*

\* 更新时间

\*/

@TableField(fill = FieldFill.INSERT\_UPDATE)

private LocalDateTime updateTime;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

import com.baomidou.mybatisplus.annotation.TableName;

import lombok.Data;

import java.io.Serializable;

import java.math.BigDecimal;

import java.time.LocalDateTime;

/\*\*

\* 选课实体类

\*

\* @author example

\*/

@Data

@TableName("course\_selection")

public class CourseSelection implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 选课ID

\*/

@TableId(value = "id", type = IdType.AUTO)

private Long id;

/\*\*

\* 学生ID

\*/

private Long studentId;

/\*\*

\* 课程开设ID

\*/

private Long courseOfferingId;

/\*\*

\* 选课状态：0-已选，1-已退，2-已修完

\*/

private Integer status;

/\*\*

\* 成绩

\*/

private BigDecimal score;

/\*\*

\* 选课时间

\*/

private LocalDateTime selectionTime;

/\*\*

\* 创建时间

\*/

private LocalDateTime createTime;

/\*\*

\* 更新时间

\*/

private LocalDateTime updateTime;

/\*\*

\* 兼容方法 - 设置选课状态

\*/

public void setSelectionStatus(Integer selectionStatus) {

this.status = selectionStatus;

}

/\*\*

\* 兼容方法 - 获取选课状态

\*/

public Integer getSelectionStatus() {

return this.status;

}

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

import com.baomidou.mybatisplus.annotation.TableName;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import java.io.Serializable;

import java.time.LocalDateTime;

/\*\*

\* 院系实体

\*

\* @author example

\*/

@Data

@TableName("department")

@Schema(description = "院系实体")

public class Department implements Serializable {

private static final long serialVersionUID = 1L;

@Schema(description = "院系ID")

@TableId(value = "id", type = IdType.AUTO)

private Long id;

@Schema(description = "院系名称")

private String name;

@Schema(description = "院系编码")

private String code;

@Schema(description = "院系描述")

private String description;

@Schema(description = "创建时间")

private LocalDateTime createTime;

@Schema(description = "更新时间")

private LocalDateTime updateTime;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

import com.baomidou.mybatisplus.annotation.TableName;

import lombok.Data;

import java.time.LocalDateTime;

/\*\*

\* 成绩实体类

\*/

@Data

@TableName("grade")

public class Grade {

/\*\*

\* 成绩ID

\*/

@TableId(value = "id", type = IdType.AUTO)

private Long id;

/\*\*

\* 学生ID

\*/

private Long studentId;

/\*\*

\* 课程ID

\*/

private Long courseId;

/\*\*

\* 分数

\*/

private Double score;

/\*\*

\* 成绩构成，如：平时成绩:30%:85,期中考试:30%:78,期末考试:40%:92

\*/

private String gradeComponents;

/\*\*

\* 状态：0-未评分，1-已评分，2-已作废

\*/

private Integer status;

/\*\*

\* 评分时间

\*/

private LocalDateTime gradeTime;

/\*\*

\* 备注

\*/

private String remark;

/\*\*

\* 创建时间

\*/

private LocalDateTime createTime;

/\*\*

\* 更新时间

\*/

private LocalDateTime updateTime;

}

package com.example.student.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

import com.baomidou.mybatisplus.annotation.TableName;

import io.swagger.v3.oas.annotations.media.Schema;

import lombok.Data;

import java.io.Serializable;

import java.time.LocalDateTime;

/\*\*

\* 专业实体

\*

\* @author example

\*/

@Data

@TableName("major")

@Schema(description = "专业实体")

public class Major implements Serializable {

private static final long serialVersionUID = 1L;

@Schema(description = "专业ID")

@TableId(value = "id", type = IdType.AUTO)

private Long id;

@Schema(description = "专业名称")

private String name;

@Schema(description = "专业编码")

private String code;

@Schema(description = "所属院系ID")

private Long departmentId;

@Schema(description = "专业描述")

private String description;

@Schema(description = "创建时间")

private LocalDateTime createTime;

@Schema(description = "更新时间")

private LocalDateTime updateTime;

}