|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 未标题5  **实验报告**  学院（系） 名称： 华信软件学院 | | | | | | | Windows平台下，[IntelliJ IDEA](https://www.so.com/link?m=bIKlEUOwsW2BeuneCIIDZFVYhfQLy5OPfUUJ6fL3T83sENHVJMOD2FfECDSOcxcWuhCjq7CjXPC0zlfTLFcIoTxK8E5qfSOGG2zuSpJZLqN%2BCmG7jll93Kg%3D%3D) 2022、Web服务器用Tomcat10.0、Java开发环境用JDK17或更高。  **五、实验报告要求**   1. 说明项目结构中各文件的功能 2. 描述数据库设计 3. 着重说明一个功能的前后端数据交互设计过程（如，添加（删除、修改）等功能） 4. 记录实验错误，写出改正过程 5. 提交代码 6. 结论与体会   AdminUsers.vue是管理员管理用户时所访问的页面，在登录界面进行管理员身份验证后可对用户进行简单操作（如删除用户等）。  当用户成功登录后先跳转至Home.vue这是一个简单的欢迎页面，5秒后自动跳转至Books.vue，此界面为图书系统的主界面，用户可在此进行图书搜索，借阅等操作。  图二为后端项目结构。后端基于 JavaWeb 技术栈，采用 Servlet 作为核心组件，负责处理客户端发来的 HTTP 请求并返回响应。通过 JDBC 实现与 MySQL 数据库的连接与操作，完成用户信息、图书数据的增删改查功能。后端接口提供了清  晰规范的 API，供前端进行数据交互。使用 JSON 格式作为前后端通信的数据格式，保证数据传输的高效和标准化。项目部署于 Tomcat 容器中，支持跨域访问配置，确保前端 Vue 应用能够顺畅调用后端接口。整体架构注重模块化和分层设计，便于维护和后续扩展。  整体采用 **MVC 架构模式** 实现系统功能的分层与解耦。后端以 **Servlet** 作为控制器（Controller），负责接收前端请求、调用业务逻辑并返回处理结果。在业务逻辑层（Service），封装了对用户、图书等核心功能的操作流程，保持代码的复用性与逻辑清晰。数据访问层（DAO）通过 **JDBC** 连接 **MySQL** 数据库，负责对数据库进行增删改查操作，确保数据读写的规范性与安全性。  UserDao 是一个操作数据库中 **users 表** 的工具类，位于后端的 **DAO 层**，用于封装所有用户相关的数据库操作逻辑，避免在业务逻辑层或控制层中直接书写 SQL，它对数据库进行 **增删改查** 操作，屏蔽了 SQL 的复杂性，便于上层调用。提高系统的**可维护性与可复用性**。  User是实体类，这里不进行过多说明。  RegisterServlet是项目后端中**实现的“注册功能接口”**，负责接收前端发来的注册请求并把用户信息存入数据库，是系统后端控制层（Controller）的核心部分。  LoginServlet是系统后端用于处理**登录请求**的 Servlet 控制器，路径为 /api/login，属于 MVC 架构中控制层的一部分，前端 Vue 登录页通过 axios.post('/api/login') 请求它来完成用户验证。他负责负责接收前端传来的用户名和密码，查询数据库验证登录信息，并返回登录是否成功以及用户的角色（如：admin、user）。  UserDeleteServlet是系统后端中负责**删除用户**的 Servlet 控制器，路径为 /api/users/delete，在 MVC 架构中处于 **控制层 Controller**，负责接收前端发送的  （三）前后端数据交互（以删除用户为例）  前端部分：  <template>  <div style="max-width: 900px; margin: 50px auto; padding: 20px; background: #f9fafb; border-radius: 8px; box-shadow: 0 2px 8px rgba(0,0,0,0.1);">  <h2 style="text-align: center; margin-bottom: 20px; color: #409EFF; font-weight: 600;">用户列表</h2>  <el-table  :data="users"  border  stripe  style="width: 100%; background: #fff; border-radius: 6px; overflow: hidden;"  :header-cell-style="{ backgroundColor: '#e6f2ff', color: '#409EFF', fontWeight: '600' }"  >  <el-table-column prop="id" label="ID" width="100" align="center" />  <el-table-column prop="username" label="用户名" />  <el-table-column prop="role" label="角色" width="140" align="center" />  <el-table-column label="删除" width="140" align="center">  <template #default="scope">  <el-button  size="small"  type="danger"  plain  circle  icon="el-icon-delete"  @click="deleteUser(scope.row)"  :title="`删除用户 ${scope.row.username}`"  />  </template>  </el-table-column>  </el-table>  </div> </template>  <script> import *axios* from 'axios'  export default {  } </script>  后端部分：  package com.example.usersystemserver.servlet;  import com.example.usersystemserver.util.JDBCUtil; import jakarta.servlet.ServletException; import jakarta.servlet.annotation.WebServlet; import jakarta.servlet.http.HttpServlet; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse;  import java.io.BufferedReader; import java.io.IOException; import java.sql.Connection; import java.sql.PreparedStatement;  @WebServlet("/api/users/delete") public class UserDeleteServlet extends HttpServlet {   private void setCorsHeaders(HttpServletResponse response) {  response.setHeader("Access-Control-Allow-Origin", "\*");  response.setHeader("Access-Control-Allow-Methods", "POST, OPTIONS");  response.setHeader("Access-Control-Allow-Headers", "Content-Type");  }   @Override  protected void doOptions(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  setCorsHeaders(resp);  resp.setStatus(HttpServletResponse.*SC\_OK*);  }   @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  setCorsHeaders(response);   response.setContentType("application/json;charset=UTF-8");   ps.setInt(1, id);  int affected = ps.executeUpdate();   if (affected > 0) {  response.getWriter().write("{\"success\": true, \"message\": \"删除成功\"}");  } else {  response.getWriter().write("{\"success\": false, \"message\": \"用户不存在\"}");  }  } catch (Exception e) {  e.printStackTrace();  response.getWriter().write("{\"success\": false, \"message\": \"服务器异常: " + e.getMessage() + "\"}");  }  } }  **删除用户功能的前后端数据交互设计过程**  **1. 前端设计**  **界面展示：** 使用 Element Plus 的 el-table 组件，展示用户列表，每一行末尾有一个删除按钮（红色圆形垃圾桶图标）。  **事件触发：**用户点击某个用户行的删除按钮时，触发 deleteUser(user) 方法。  **2. 前端请求设计**  **请求方式：** 采用 POST 请求方式，向后端 /api/users/delete 发送 JSON 数据，内容是 { id: user.id }，明确告诉服务器要删除哪个用户。  **请求头:**请求带有 Content-Type: application/json，请求体是 JSON 格式。  **响应处理:** 根据服务器返回的 JSON { success: true/false, message: "提示信息" }，前端弹出相应提示，并更新页面数据。  **3. 后端设计**  **接口路径:**监听 /api/users/delete 路径，处理 POST 请求。  **跨域支持:**设置响应头允许跨域，支持前端访问。  **请求体读取:**读取请求体中的 JSON 字符串，手动解析出 id 参数（用户ID）。  **数据库操作:** 通过 JDBC 连接数据库，执行 DELETE FROM users WHERE id = ?，删除对应用户记录。      （五）关于防止恶意登录  在本项目的后端数据库操作中，我重点防范了**SQL注入攻击**，保障系统安全性。  **防护措施：使用PreparedStatement** 项目中所有涉及数据库增删改查的SQL语句，均采用PreparedStatement预编译语句，并通过参数绑定传递用户输入，避免了将输入直接拼接到SQL语句中。  示例代码片段：  String sql = "SELECT role FROM users WHERE username = ? AND password = ?"; |
| 姓名 | 魏小焱 | 学号 | 20231842 | 专业 | 软件工程 |  |
| 班级 | 5班 | 实验项目 | 实验三 基于JAVAWEB的在线图书管理系统 | | |
| 课程名称 | | JAVAWeb开发技术 | | 课程代码 | 1960723 |
| 实验时间 | | 2025.6.17 | | 实验地点 | C508、C510 |
| 批改意见 | | | | 成绩 |  |
| 教师签字： | | | | | |
| **一、实验目的**  通过实验掌握下列知识:   1. 掌握vue3、elementPlus和MVC的基本使用 2. 掌握通过vue3 的axios实现异步加载 3. 掌握servlet如何提供合格的json数据给jsp页面的异步请求   **二、实验任务：**  该系统应该具有如下内容:   1. 该系统具有两类用户:前端注册用户和后台管理用户: 2. 根据实际需求设计区分前端注册用户和后台管理用户的不同   **三、实验要求：**   1. 能够在理解 MVC模式的基础上，将该系统在代码和页面的组织上分成视图、控制和模型三层。 2. 要有防止恶意登录功能 3. 界面的美化使用前端框架完成 4. 要有数据库连接池的使用    1. **实验环境**   （一）项目结构  本次实验采用了前后端分离的设计方法，提高了程序的灵活性和可维护性，便于进行一些功能的扩展    图一为前端项目结构。前端我采用的是 **Vue3**作为核心框架，负责构建用户界面和交互，**Element UI**组件库，提供丰富的 UI 组件（按钮、输入框、表格等）， **Axios** 作为 HTTP 客户端，负责跟后端 API 做异步请求（登录、注册、图书数据请求等），**Vue Router** 用于前端页面路由管理，实现单页应用多页面导航，**ES6+ JavaScript** 现代 JS 语法编写代码，提高开发效率和代码可读性。  Login.vue 是该系统的主应用界面，用户在此处进行登录操作（如未注册可以点击注册跳转至Register.vue界面进行注册操作），管理员可以输入管理员账号进入管理系统（AdminUsers.vue）  删除请求，并操作数据库删除指定用户。它接收前端发来的用户 ID（JSON 格式），然后调用数据库删除该用户记录，并返回删除是否成功的 JSON 响应。U  UserServlet是用于处理**获取用户列表**请求的 Servlet，路径为 /api/users，返回用户信息列表,处于 **控制器 Controller 层**，主要用于 **前台或管理端展示用户数据**。它接收前端 GET 请求，查询 users 表中的所有用户（只查 id, username, role），并将数据封装成 JSON 格式响应给前端。  **JDBCUtil数据库连接工具类**，用于简化 JDBC 编程中对数据库连接的管理。属于 **模型层（Model）中的数据库连接辅助工具类**。它提供数据库连接（getConnection()），释放数据库资源（close(...)），常常被 UserDao 等数据访问对象调用，以连接和操作 MySQL 数据库。  **CorsFilter全局跨域处理过滤器** ，用来解决前端与 Java Servlet 后端之间的 **跨域请求（CORS）问题**。当浏览器跨域发起请求时会先发送一个 **预检请求。** 它的职责是：   1. 拦截所有请求（包括 OPTIONS） 2. 设置响应头允许跨域 3. 如果是 OPTIONS 请求，**提前响应 200**，避免请求继续向下执行 4. 其他请求正常放行（chain.doFilter()）   （二）数据库设计  CREATE DATABASE systemuser DEFAULT CHARSET utf8mb4;  USE systemuser;  CREATE TABLE users (  id INT PRIMARY KEY AUTO\_INCREMENT,  username VARCHAR(50) NOT NULL UNIQUE,  password VARCHAR(100) NOT NULL,  role VARCHAR(10) DEFAULT 'user' COMMENT 'user/admin',  create\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* );  INSERT INTO users (username, password, role) VALUES  ('user1', '123456', 'user'),  ('user2', 'abcdef', 'user'),  ('admin1', 'admin123', 'admin'),  ('admin2', 'root456', 'admin');  data() {  return {  users: []  }  },  created() {  this.fetchUsers()  },  methods: {  async fetchUsers() {  try {  const res = await axios.get('http://localhost:8080/user\_system\_server\_war\_exploded/api/users')  if (res.data.success) {  this.users = res.data.users  } else {  this.*$message*.error('获取用户列表失败')  }  } catch (err) {  this.*$message*.error('网络错误，获取用户列表失败')  }  },  async deleteUser(user) {  try {  const res = await axios.post('http://localhost:8080/user\_system\_server\_war\_exploded/api/users/delete', { id: user.id })  if (res.data.success) {  this.users = this.users.filter(u => u.id !== user.id)  this.*$message*.success(`已删除用户：${user.username}`)  } else {  this.*$message*.error('删除失败：' + (res.data.message || '未知错误'))  }  } catch (err) {  this.*$message*.error('网络错误，删除失败')  }  }  }  StringBuilder sb = new StringBuilder();  try (BufferedReader reader = request.getReader()) {  String line;  while ((line = reader.readLine()) != null) {  sb.append(line);  }  }   String jsonStr = sb.toString().trim();  int id = -1;   // 简单手动解析id，格式为{"id":数字}  try {  int idIndex = jsonStr.indexOf("\"id\"");  if (idIndex != -1) {  int colonIndex = jsonStr.indexOf(":", idIndex);  int commaIndex = jsonStr.indexOf(",", colonIndex);  int endIndex = commaIndex == -1 ? jsonStr.indexOf("}", colonIndex) : commaIndex;  String idStr = jsonStr.substring(colonIndex + 1, endIndex).trim();  id = Integer.*parseInt*(idStr);  }  } catch (Exception e) {  response.getWriter().write("{\"success\": false, \"message\": \"请求参数错误\"}");  return;  }   if (id == -1) {  response.getWriter().write("{\"success\": false, \"message\": \"缺少用户ID\"}");  return;  }   try (Connection conn = JDBCUtil.*getConnection*();  PreparedStatement ps = conn.prepareStatement("DELETE FROM users WHERE id = ?")) {//防止SQL注入 **响应结果** 根据数据库执行结果，返回 JSON：  删除成功：{"success": true, "message": "删除成功"}  用户不存在或删除失败：{"success": false, "message": "用户不存在"}  服务器异常：{"success": false, "message": "服务器异常: 错误信息"}  **4. 数据流动总结**   | **步骤序号** | **说明** | **内容** | | --- | --- | --- | | 1 | 用户点击删除按钮 | 触发 Vue 方法发送删除请求 | | 2 | 前端发送 POST 请求 | URL /api/users/delete，请求体 {id: 用户id} | | 3 | 后端Servlet接收请求 | 解析请求体，获取用户ID | | 4 | 后端数据库执行删除操作 | 执行 SQL 删除语句 | | 5 | 后端返回操作结果 | 返回 JSON 表示成功或失败 | | 6 | 前端接收响应并更新界面 | 删除对应用户，刷新用户列表 |   **优点:**  采用JSON格式，前后端通信格式统一，易于扩展和维护  使用POST请求传递参数，避免参数暴露在URL中，安全性更高  删除后前端页面立即更新，无需刷新，提高用户体验  后端异常处理清晰，便于定位问题  （四）部分界面展示    PreparedStatement ps = conn.prepareStatement(sql);  ps.setString(1, username);  ps.setString(2, password);  ResultSet rs = ps.executeQuery();  通过这种方式，数据库驱动会自动对输入进行转义，防止攻击者注入恶意代码。  **优势**：有效防止SQL注入，提升系统安全，保持代码简洁，易于维护，适用于各种用户输入，安全性强  （六）实验错误及解决方案  **1.跨域请求被阻止**  **错误描述**：前端调用后端接口时，出现跨域请求失败，导致无法正常交互。 **解决方案**：后端统一添加CORS响应头，使用过滤器（Filter）或在Servlet中设置Access-Control-Allow-Origin等响应头，允许跨域请求。  **2. 登录验证不严谨导致安全隐患**  **错误描述**：初期密码明文存储，存在安全风险；登录时简单字符串匹配，未做防注入处理。 **解决方案**：采用PreparedStatement防止SQL注入，后续可引入密码加密（如MD5、bcrypt）存储，提升系统安全。  **3. 前端路由跳转不生效**  **错误描述**：登录后路由跳转异常，用户体验不佳。 **解决方案**：调试前端路由配置，确保调用this.$router.push()正确执行，并对异步请求结果正确处理。  （七）实验总结  本次实验通过实现一个基于Vue3与Java Web的用户管理系统，深入理解了前后端分离架构设计。通过练习用户注册、登录、用户列表展示及删除功能，加深了对HTTP请求、JSON数据交互及数据库操作的掌握。 项目采用MVC模式组织代码，前端负责界面及交互，后端负责业务逻辑及数据持久化，模块职责分明，提升了系统的可维护性。  **（**八）实验收获   1. 熟悉了Vue3框架及Element Plus组件的使用，掌握了单页应用路由管理与异步数据请求方法。 2. 理解并实践了Java Servlet处理HTTP请求与响应的流程，掌握了跨域资源共享（CORS）解决方案。 3. 深刻认识到数据库安全的重要性，学会使用PreparedStatement有效防止SQL注入攻击。 4. 体会到前后端分离开发模式的优势，提升了系统开发效率及代码解耦能力。 5. 通过实际编码与调试，积累了解决常见错误和异常的经验，增强了开发问题排查能力。 | | | | | |