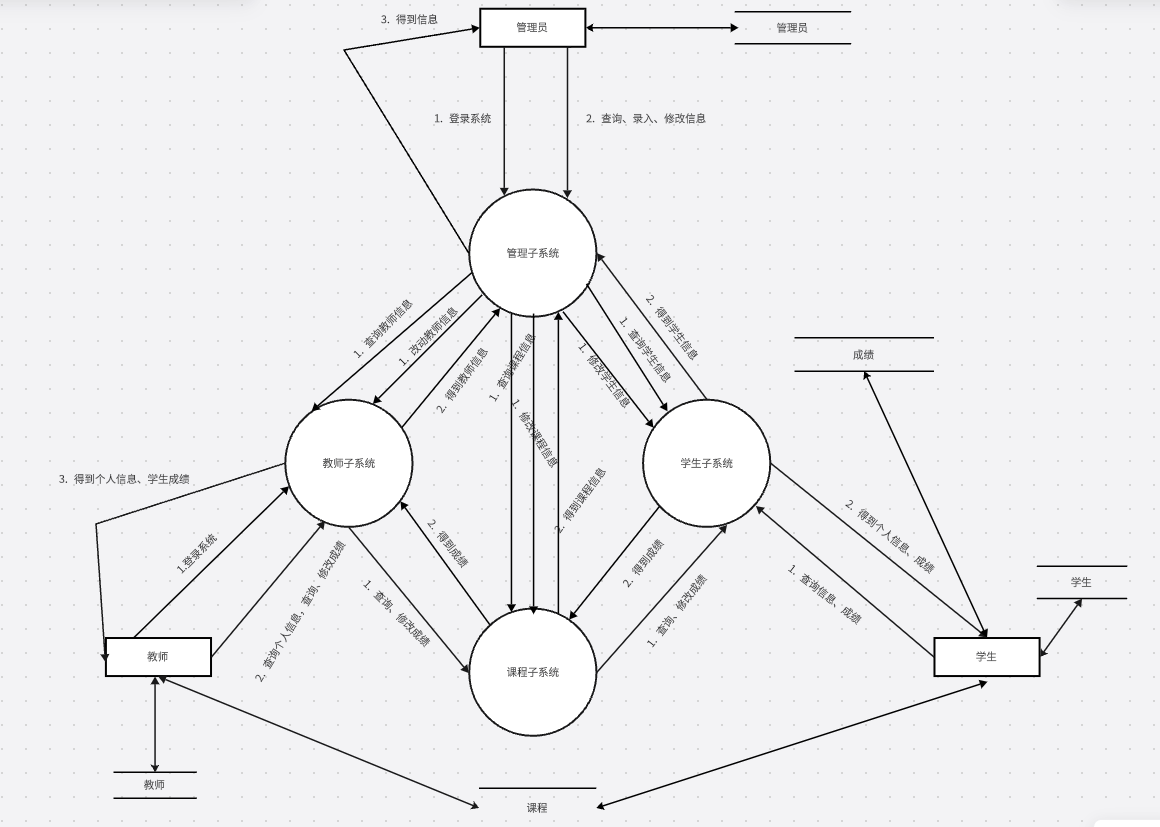
**课程设计实验报告**

1. **系统需求分析报告**

1.1 **数据流图**



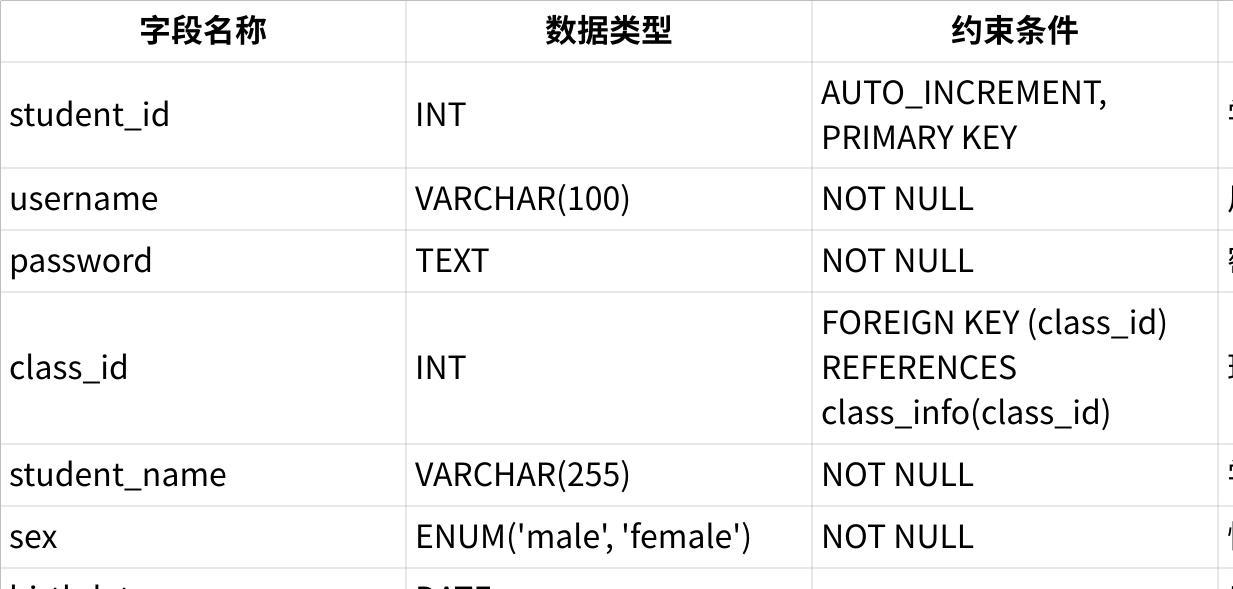
1.2 **数据字典**

**专业班级表 (class\_info)**



**点击图片可查看完整电子表格**

**学生表 (student\_info)**



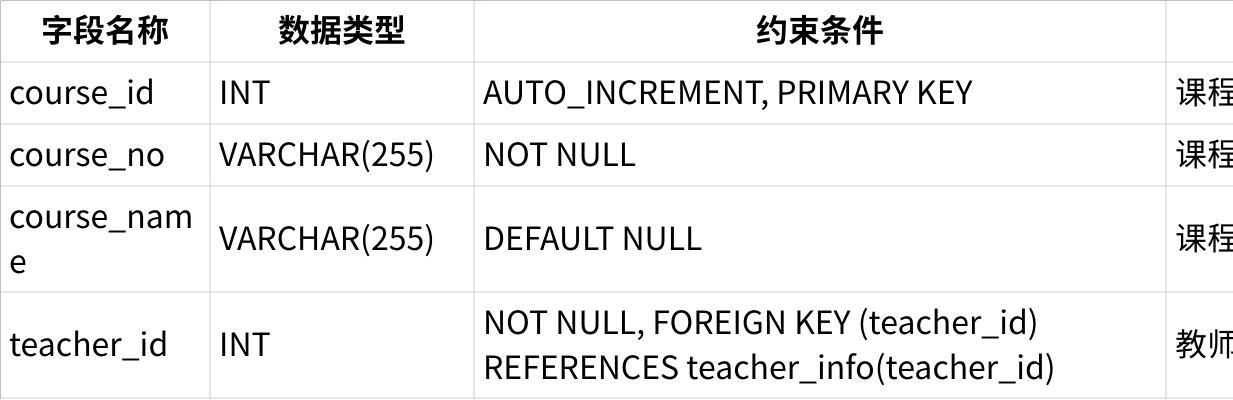
**点击图片可查看完整电子表格**

**教师表 (teacher\_info)**



**点击图片可查看完整电子表格**

**课程表 (course\_info)**



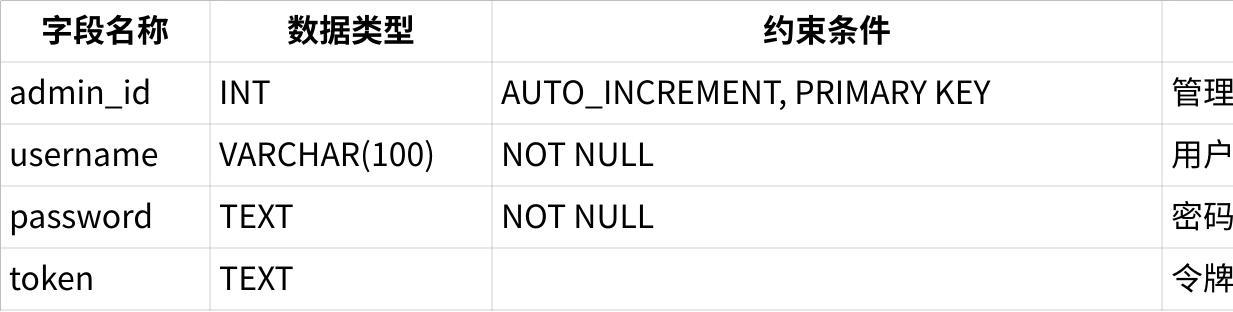
**点击图片可查看完整电子表格**

**学生选课成绩表 (student\_course)**



**点击图片可查看完整电子表格**

**管理员表 (admin\_info)**



**点击图片可查看完整电子表格**

1.3 **功能分析**

**角色分析**

1. **学生**

* 登录系统。
* 查询个人基本信息和成绩。

1. **教师**

* 登录系统。
* 查询个人基本信息。
* 查询所授课程的学生成绩。
* 录入、修改、删除学生成绩。

1. **管理员**

* 登录系统。
* 查询、录入、修改、删除学生、教师和课程的基本信息。

**功能模块分析**

1. **查询模块**

* **学生**
* 查询个人基本信息。
* 查询个人成绩。
* **教师**
* 查询个人基本信息。
* 查询所授课程的学生成绩。
* **管理员**
* 查询学生、教师和课程的基本信息。

1. **录入模块**

* **教师**
* 录入学生成绩。
* **管理员**
* 录入学生、教师和课程信息。

1. **修改模块**

* **教师**
* 修改学生成绩。
* **管理员**
* 修改学生、教师和课程信息。

1. **删除模块**

* **教师**
* 删除学生成绩。
* **管理员**
* 删除学生、教师和课程信息。

1. **系统配置模块**

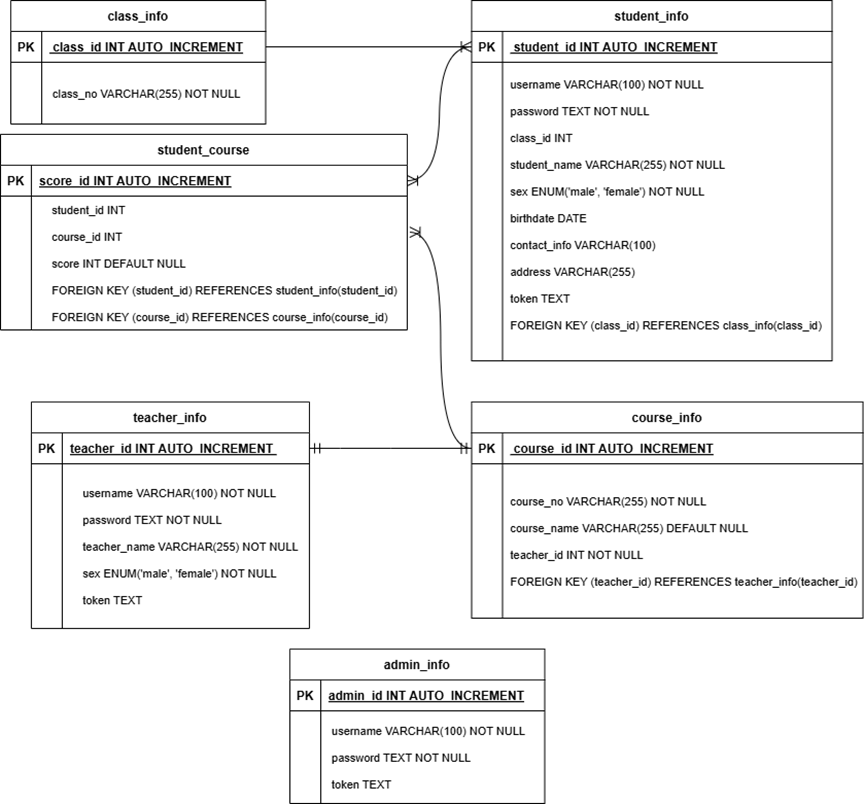
* 密码修改和重置。

1. **统计模块**

* 统计各分数段学生的成绩分布，生成直方图和饼图。

2. **数据库逻辑结构设计**

2.1 **E-R图**



2.2 **关系模式**

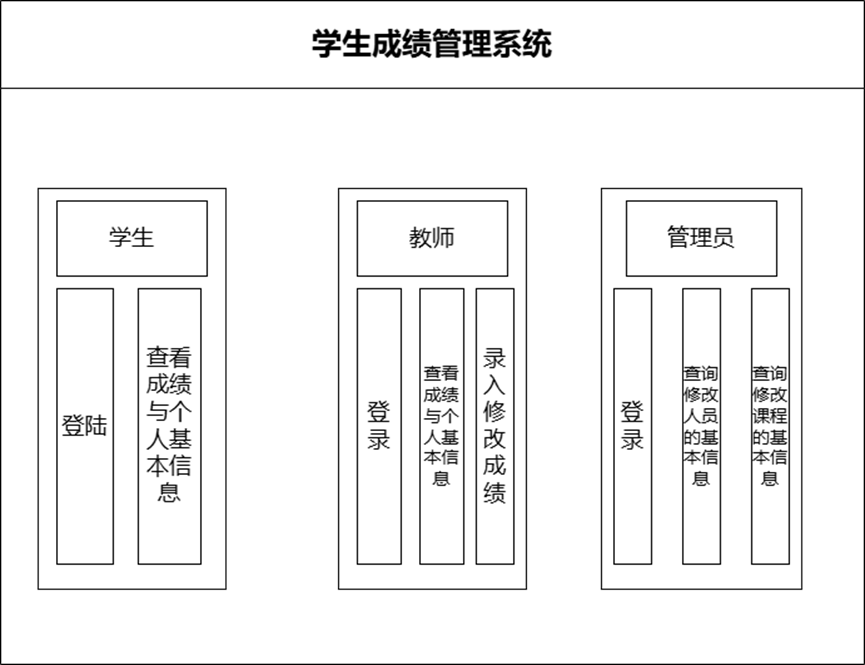
* student\_info (student\_id，username，password，class\_id，student\_name，sex，birthdate，address，token，class\_id)
* class\_info(class\_id，class\_no)
* teacher\_info(teacher\_id，username.password，teacher\_name，sex，token)
* course\_info(course\_id，course\_no，course\_name，teacher\_id)
* student\_course(score\_id，score，score，student\_id，course\_id)
* admin\_info(admin\_id，username，password，token)

2.3 **数据库结构**

* 数据库建立所需要的表
* 学生表（student\_info）：存储学生的基本信息，包括学号、用户名、密码、姓名、性别、出生日期、联系方式、地址和令牌。其中学号是主键，用于唯一标识每个学生。
* 班级表（class\_info）：存储班级的基本信息，包括班级编号和班级名称。其中班级编号是主键，用于唯一标识每个班级。
* 教师表（teacher\_info）：存储教师的基本信息，包括教师编号、用户名、密码、姓名、性别和令牌。其中教师编号是主键，用于唯一标识每个教师。
* 课程表（course\_info）：存储课程的基本信息，包括课程编号、课程编号、课程名称和教师编号。其中课程编号是主键，用于唯一标识每个课程。教师编号是外键，与教师表中的教师编号关联。
* 选课成绩表（student\_course）：存储学生选课的成绩信息，包括成绩编号、学生编号、课程编号和成绩。其中成绩编号是主键，用于唯一标识每个成绩记录。学生编号和课程编号分别是外键，分别与学生表和课程表中的主键关联。
* 管理员表（admin\_info）：存储管理员的基本信息，包括管理员编号、用户名、密码和令牌。其中管理员编号是主键，用于唯一标识每个管理员。
* 表之间的关系
* 学生表与班级表之间存在多对一的关系，即一个学生属于一个班级，一个班级可以有多个学生。通过在学生表中添加一个外键字段 class\_id 来表示这种关系。
* 教师表与课程表之间存在一对多的关系，即一个教师可以教授多门课程，一门课程只能由一位教师教授。通过在课程表中添加一个外键字段 teacher\_id 来表示这种关系。
* 学生表与选课成绩表之间存在一对多的关系，即一个学生可以选择多门课程，每门课程对应一个成绩。通过在选课成绩表中添加两个外键字段 student\_id 和 course\_id 来表示这种关系。
* 综上所述，这个数据库结构包含了学生、班级、教师、课程和选课成绩等实体，并通过外键关联了它们之间的关系。

3. **软件结构设计**

3.1 **系统软件（功能）结构图**



3.2 **程序运行环境及开发环境**

**操作系统**

Windows11

**数据库管理系统**

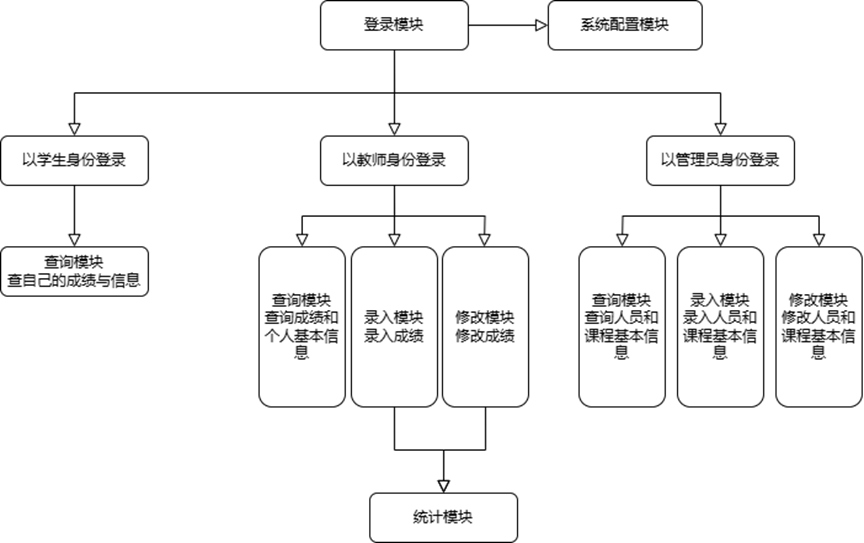
MySQL8.2

**开发工具**

Apifox+MySQL CE+JDK 22

3.3 **程序的详细设计**

**模块之间的关系**



**模块的功能**

* 登录模块：负责选择用户登录的类型，并完成登录动作
* 查询模块：负责学生查询自己的基本信息和成绩，教师查询成绩和个人基本信息，管理员查询基本信息情况
* 录入模块：以教师的角色登录，允许录入成绩；以管理员身份登录允许录入人员和课程信息；
* 修改模块：以教师的角色登录，允许修改成绩；以管理员身份登录允许修改人员和课程信息；
* 系统配置模块：密码修改和重置等
* 统计模块：统计各分数段学生的成绩分布，画出直方图和饼图。

**主要功能实现的程序段**

* 管理员控制器

|  |
| --- |
| Java // 管理员登录 @PostMapping("/login") public ResponseEntity<Response<String>> loginAdmin(@RequestBody LoginRequest loginRequest) {  try {  String token = adminService.login(loginRequest.getUsername(), loginRequest.getPassword());  return ResponseEntity.ok(new Response<>(200, "Login successful", token));  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java  // 管理员注销 @PostMapping("/logout") public ResponseEntity<Response<String>> logoutAdmin(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.logout(token);  return ResponseEntity.ok(new Response<>(200, "Logout successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员查询对应ID学生的基本信息 @GetMapping("/studentInfo/{studentId}") public ResponseEntity<Response<StudentInfo>> getStudentInfo(@RequestHeader("Authorization") String authHeader, @PathVariable("studentId") Integer studentId) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  StudentInfo studentInfo = adminService.getStudentInfo(token, studentId);  return ResponseEntity.ok(new Response<>(200, "Get student info successful", studentInfo));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员查询所有学生的基本信息 @GetMapping("/studentInfo") public ResponseEntity<Response<List<StudentInfo>>> getAllStudentInfo(@RequestHeader("Authorization") String authHeader) {  try {  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  List<StudentInfo> studentInfoList = adminService.getAllStudentInfo(token);  return ResponseEntity.ok(new Response<>(200, "Get all student info successful", studentInfoList));  } else {  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员查询教师的基本信息 @GetMapping("/teacherInfo/{teacherId}") public ResponseEntity<Response<TeacherInfo>> getTeacherInfo(@RequestHeader("Authorization") String authHeader, @PathVariable("teacherId") Integer teacherId) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  TeacherInfo teacherInfo = adminService.getTeacherInfo(token, teacherId);  return ResponseEntity.ok(new Response<>(200, "Get teacher info successful", teacherInfo));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员查询所有教师的基本信息 @GetMapping("/teacherInfo") public ResponseEntity<Response<List<TeacherInfo>>> getAllTeacherInfo(@RequestHeader("Authorization") String authHeader) {  try {  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  List<TeacherInfo> teacherInfoList = adminService.getAllTeacherInfo(token);  return ResponseEntity.ok(new Response<>(200, "Get all teacher info successful", teacherInfoList));  } else {  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员修改学生的基本信息 @PutMapping("/studentInfo/{studentId}") public ResponseEntity<Response<String>> updateStudentInfo(@RequestHeader("Authorization") String authHeader, @PathVariable("studentId") Integer studentId, @RequestBody StudentInfo studentInfo) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.updateStudentInfo(token, studentId, studentInfo);  return ResponseEntity.ok(new Response<>(200, "Update student info successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员修改教师的基本信息 @PutMapping("/teacherInfo/{teacherId}") public ResponseEntity<Response<String>> updateTeacherInfo(@RequestHeader("Authorization") String authHeader, @PathVariable("teacherId") Integer teacherId, @RequestBody TeacherInfo teacherInfo) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.updateTeacherInfo(token, teacherId, teacherInfo);  return ResponseEntity.ok(new Response<>(200, "Update teacher info successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员删除学生 @DeleteMapping("/studentInfo/{studentId}") public ResponseEntity<Response<String>> deleteStudent(@RequestHeader("Authorization") String authHeader, @PathVariable("studentId") Integer studentId) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.deleteStudent(token, studentId);  return ResponseEntity.ok(new Response<>(200, "Delete student successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员删除教师 @DeleteMapping("/teacherInfo/{teacherId}") public ResponseEntity<Response<String>> deleteTeacher(@RequestHeader("Authorization") String authHeader, @PathVariable("teacherId") Integer teacherId) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.deleteTeacher(token, teacherId);  return ResponseEntity.ok(new Response<>(200, "Delete teacher successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员查询课程的基本信息 @GetMapping("/courseInfo/{courseId}") public ResponseEntity<Response<CourseInfo>> getCourseInfo(@RequestHeader("Authorization") String authHeader, @PathVariable("courseId") Integer courseId) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  CourseInfo courseInfo = adminService.getCourseInfo(token, courseId);  return ResponseEntity.ok(new Response<>(200, "Get course info successful", courseInfo));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员查询所有课程的基本信息 @GetMapping("/courseInfo") public ResponseEntity<Response<List<CourseInfo>>> getAllCourseInfo(@RequestHeader("Authorization") String authHeader) {  try {  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  List<CourseInfo> courseInfoList = adminService.getAllCourseInfo(token);  return ResponseEntity.ok(new Response<>(200, "Get all course info successful", courseInfoList));  } else {  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员修改课程的基本信息 @PutMapping("/courseInfo/{courseId}") public ResponseEntity<Response<String>> updateCourseInfo(@RequestHeader("Authorization") String authHeader, @PathVariable("courseId") Integer courseId, @RequestBody CourseInfo courseInfo) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.updateCourseInfo(token, courseId, courseInfo);  return ResponseEntity.ok(new Response<>(200, "Update course info successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员删除课程 @DeleteMapping("/courseInfo/{courseId}") public ResponseEntity<Response<String>> deleteCourse(@RequestHeader("Authorization") String authHeader, @PathVariable("courseId") Integer courseId) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.deleteCourse(token, courseId);  return ResponseEntity.ok(new Response<>(200, "Delete course successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员重置学生密码 @PutMapping("/studentPassword/{studentId}") public ResponseEntity<Response<String>> resetStudentPassword(@RequestHeader("Authorization") String authHeader, @PathVariable("studentId") Integer studentId, @RequestBody Password password) {  try {  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.resetStudentPassword(token, studentId, password.getPassword());  return ResponseEntity.ok(new Response<>(200, "Reset student password successful", null));  } else {  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 管理员重置教师密码 @PutMapping("/teacherPassword/{teacherId}") public ResponseEntity<Response<String>> resetTeacherPassword(@RequestHeader("Authorization") String authHeader, @PathVariable("teacherId") Integer teacherId, @RequestBody Password password) {  try {  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  adminService.resetTeacherPassword(token, teacherId, password.getPassword());  return ResponseEntity.ok(new Response<>(200, "Reset teacher password successful", null));  } else {  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

* 学生控制器

|  |
| --- |
| Java // 学生登录 @PostMapping("/login") public ResponseEntity<Response<String>> loginStudent(@RequestBody LoginRequest loginRequest) {  try {  String token = studentService.login(loginRequest.getUsername(), loginRequest.getPassword());  return ResponseEntity.ok(new Response<>(200, "Login successful", token));  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 学生注销 @PostMapping("/logout") public ResponseEntity<Response<String>> logoutStudent(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  studentService.logout(token);  return ResponseEntity.ok(new Response<>(200, "Logout successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 学生获取自己的基本信息 @GetMapping("/info") public ResponseEntity<Response<StudentInfo>> getStudentInfo(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  StudentInfo studentInfo = studentService.getStudentInfo(token);  return ResponseEntity.ok(new Response<>(200, "Get student info successful", studentInfo));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 学生查询自己的成绩 @GetMapping("/score") public ResponseEntity<Response<List<CourseScore>>> getStudentScore(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设getStudentScore方法接受一个参数：令牌  List<CourseScore> score = studentService.getStudentScore(token);  return ResponseEntity.ok(new Response<>(200, "Get student score successful", score));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 学生修改密码 @PutMapping("/password") public ResponseEntity<Response<String>> changePassword(@RequestHeader("Authorization") String authHeader, @RequestBody Password password) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  studentService.changePassword(token, password.getPassword());  return ResponseEntity.ok(new Response<>(200, "Change password successful, please login again", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } } |

* 教师控制器

|  |
| --- |
| Java // 教师登录 @PostMapping("/login") public ResponseEntity<Response<String>> loginTeacher(@RequestBody LoginRequest loginRequest) {  try {  String token = teacherService.login(loginRequest.getUsername(), loginRequest.getPassword());  return ResponseEntity.ok(new Response<>(200, "Login successful", token));  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师注销 @PostMapping("/logout") public ResponseEntity<Response<String>> logoutTeacher(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  teacherService.logout(token);  return ResponseEntity.ok(new Response<>(200, "Logout successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师获取自己的基本信息 @GetMapping("/info") public ResponseEntity<Response<TeacherInfo>> getStudentInfo(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  TeacherInfo teacherInfo = teacherService.getTeacherInfo(token);  return ResponseEntity.ok(new Response<>(200, "Get teacher info successful", teacherInfo));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师查询所授课程的基本信息 @GetMapping("/course") public ResponseEntity<Response<List<Course>>> getTeacherCourse(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设getTeacherCourse方法接受一个参数：令牌  List<Course> course = teacherService.getTeacherCourse(token);  return ResponseEntity.ok(new Response<>(200, "Get teacher course successful", course));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师查询所授课程的所有学生的成绩 @GetMapping("/score") public ResponseEntity<Response<List<CourseScore>>> getTeacherScore(@RequestHeader("Authorization") String authHeader) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设getTeacherScore方法接受一个参数：令牌  List<CourseScore> score = teacherService.getTeacherScore(token);  return ResponseEntity.ok(new Response<>(200, "Get teacher score successful", score));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师更新所上课程某学生的成绩 @PutMapping("/score") public ResponseEntity<Response<String>> updateStudentScore(@RequestHeader("Authorization") String authHeader, @RequestBody TakeCourse takeCourse) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设updateStudentScore方法接受两个参数：令牌和学生成绩  teacherService.updateStudentScore(token, takeCourse);  return ResponseEntity.ok(new Response<>(200, "Update student score successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师新增所上课程的学生 @PostMapping("/addStudent") public ResponseEntity<Response<String>> addStudent(@RequestHeader("Authorization") String authHeader, @RequestBody TakeCourse takeCourse) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设addStudent方法接受两个参数：令牌和学生信息  teacherService.addStudent(token, takeCourse);  return ResponseEntity.ok(new Response<>(200, "Add student successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

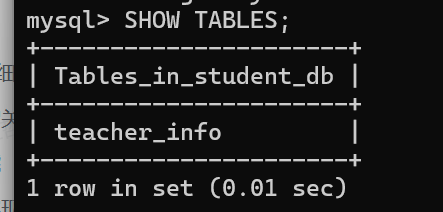
|  |
| --- |
| Java // 教师删除所上课程的学生 @DeleteMapping("/deleteStudent") public ResponseEntity<Response<String>> deleteStudent(@RequestHeader("Authorization") String authHeader, @RequestBody TakeCourse takeCourse) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设deleteStudent方法接受两个参数：令牌和学生信息  teacherService.deleteStudent(token, takeCourse);  return ResponseEntity.ok(new Response<>(200, "Delete student successful", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } |

|  |
| --- |
| Java // 教师修改密码 @PutMapping("/password") public ResponseEntity<Response<String>> updatePassword(@RequestHeader("Authorization") String authHeader, @RequestBody Password password) {  try {  // 通常Bearer Token是以"Bearer "为前缀，这里移除前缀获取真正的token  if (authHeader != null && authHeader.startsWith("Bearer ")) {  String token = authHeader.substring(7);  // 假设updatePassword方法接受两个参数：令牌和新密码  teacherService.updatePassword(token, password.getPassword());  return ResponseEntity.ok(new Response<>(200, "Update password successful, please login again", null));  } else {  // 如果请求头不包含Bearer Token，则返回错误响应  return ResponseEntity.badRequest().body(new Response<>(400, "No Bearer Token found", null));  }  } catch (RuntimeException e) {  return ResponseEntity.badRequest().body(new Response<>(400, e.getMessage(), null));  } } } |

**4 程序模块调试和运行**

**建表**

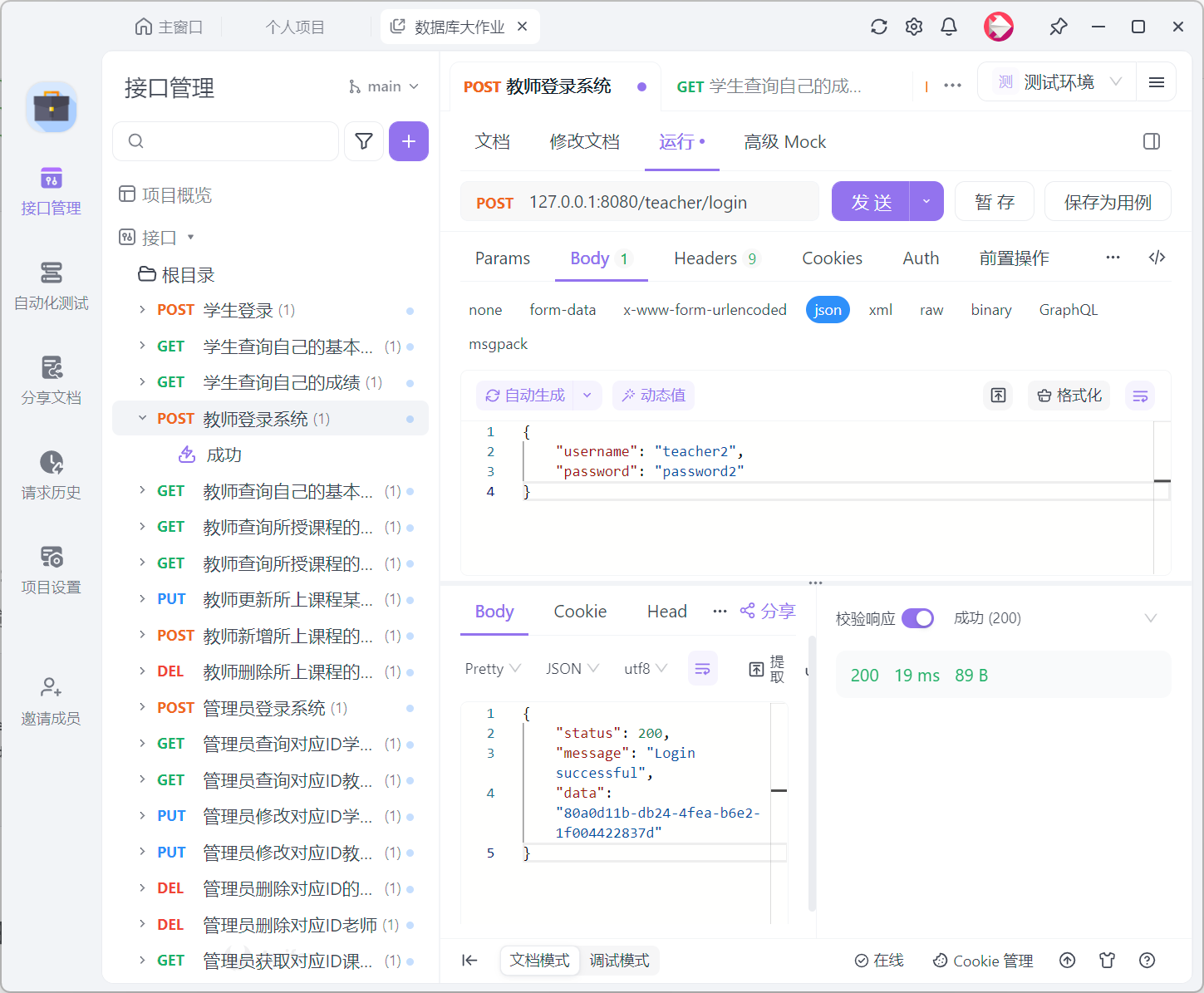
|  |
| --- |
| SQL CREATE DATABASE student\_db; USE student\_db;/ INSERT INTO teacher\_info (username, password, teacher\_name, sex) VALUES//建立teacher\_info表  ('teacher1', 'password1', '教师一', 'male'),  ('teacher2', 'password2', '教师二', 'female'),  ('teacher3', 'password3', '教师三', 'male'),  ('teacher4', 'password4', '教师四', 'female'),  ('teacher5', 'password5', '教师五', 'male'),  ('teacher6', 'password6', '教师六', 'female'),  ('teacher7', 'password7', '教师七', 'male'),  ('teacher8', 'password8', '教师八', 'female'),  ('teacher9', 'password9', '教师九', 'male'),  ('teacher10', 'password10', '教师十', 'female'),  ('teacher11', 'password11', '教师十一', 'male'),  ('teacher12', 'password12', '教师十二', 'female'),  ('teacher13', 'password13', '教师十三', 'male'),  ('teacher14', 'password14', '教师十四', 'female'),  ('teacher15', 'password15', '教师十五', 'male'),  ('teacher16', 'password16', '教师十六', 'female'),  ('teacher17', 'password17', '教师十七', 'male'),  ('teacher18', 'password18', '教师十八', 'female'),  ('teacher19', 'password19', '教师十九', 'male'),  ('teacher20', 'password20', '教师二十', 'female'); |



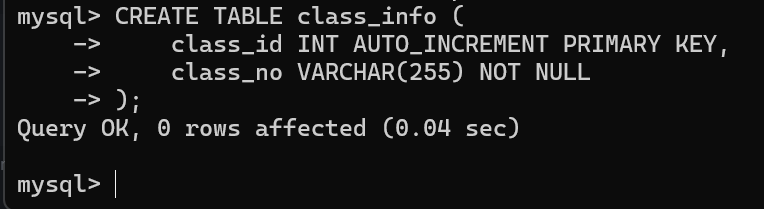
在apifox导入文件SGMS.apifox.json，测试环境设为127.0.0.1:8080

在教室登录系统接口输入：

|  |
| --- |
| SQL {  "username": "teacher2",  "password": "password2" } |

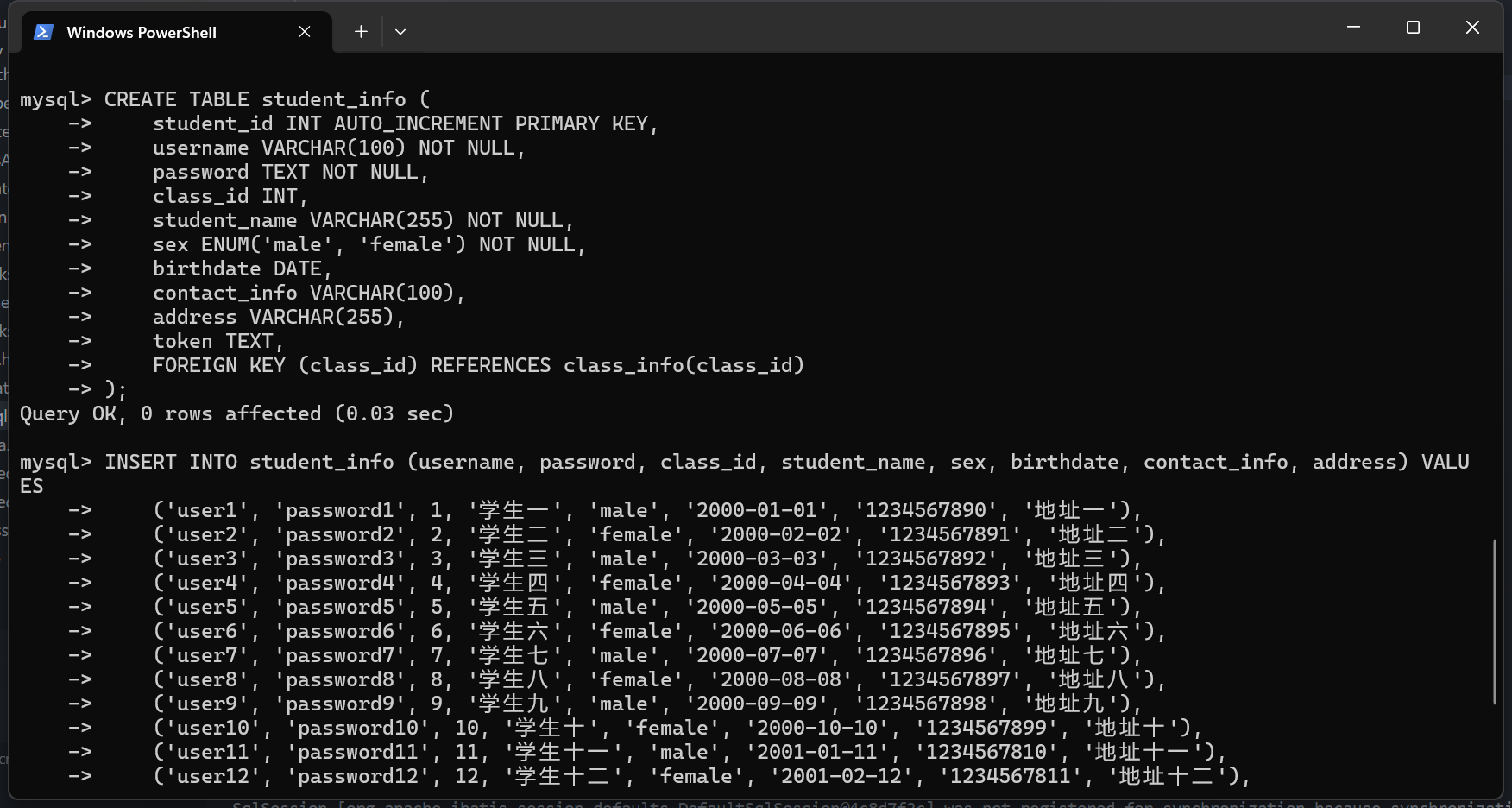


显示登录成功；

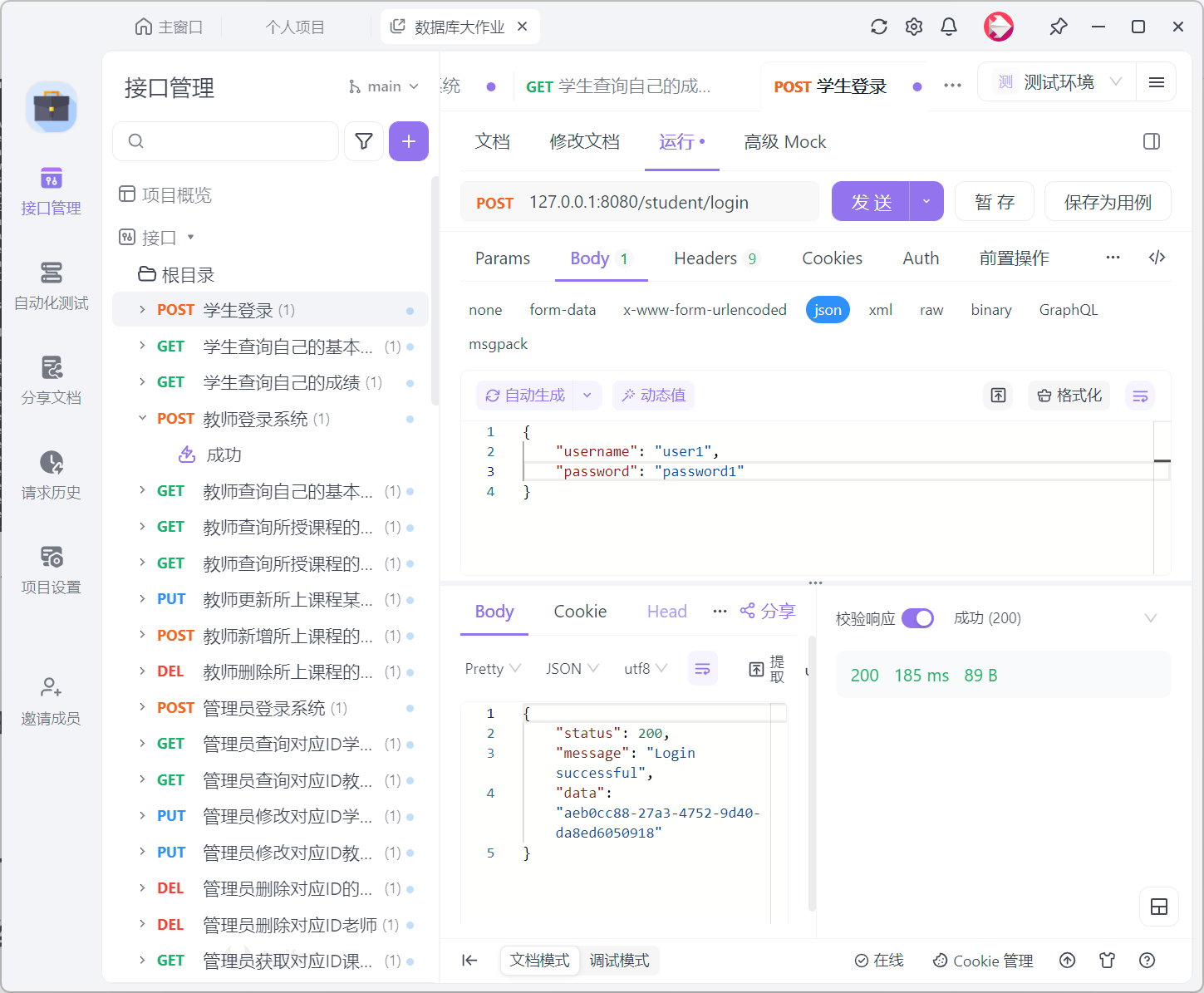


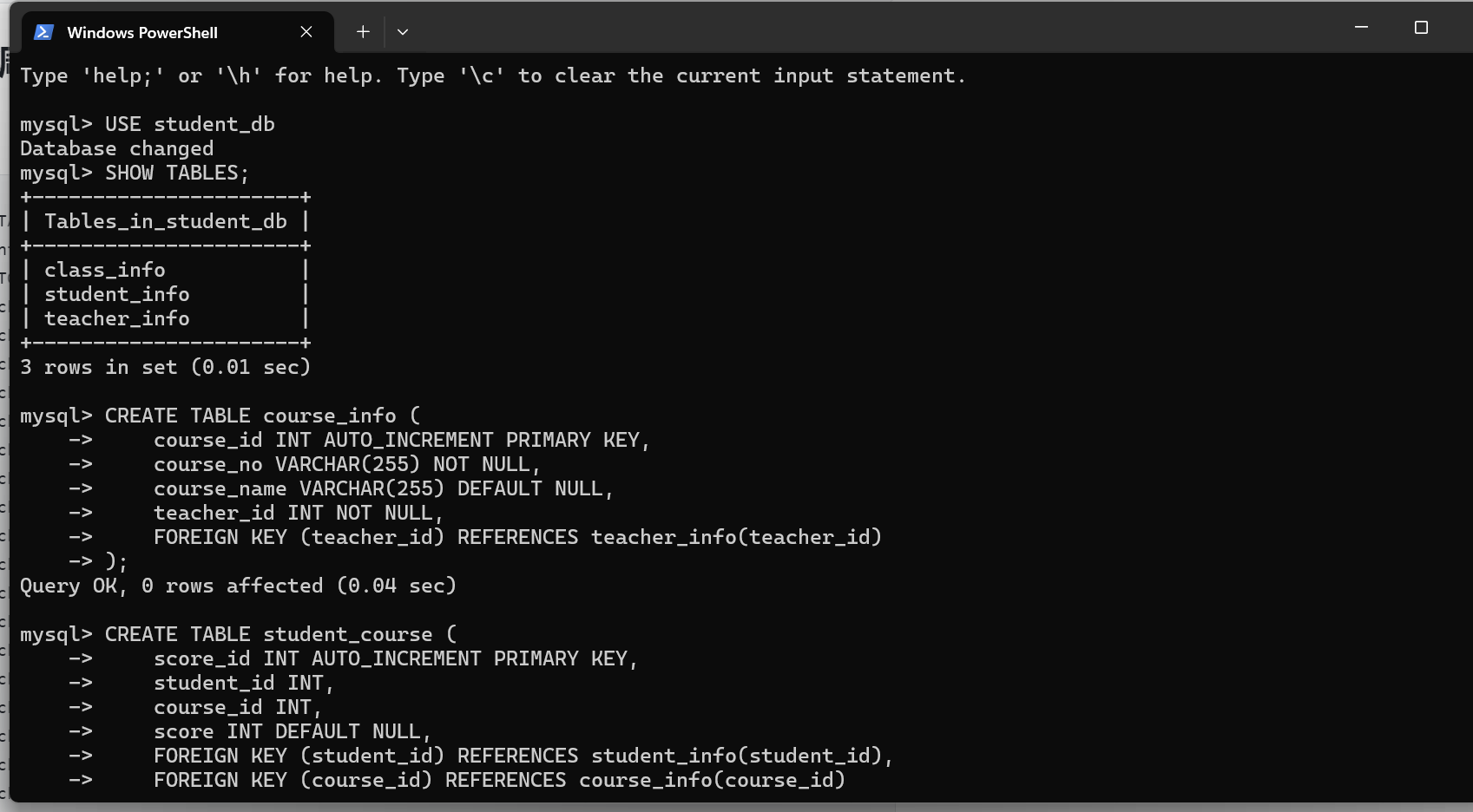


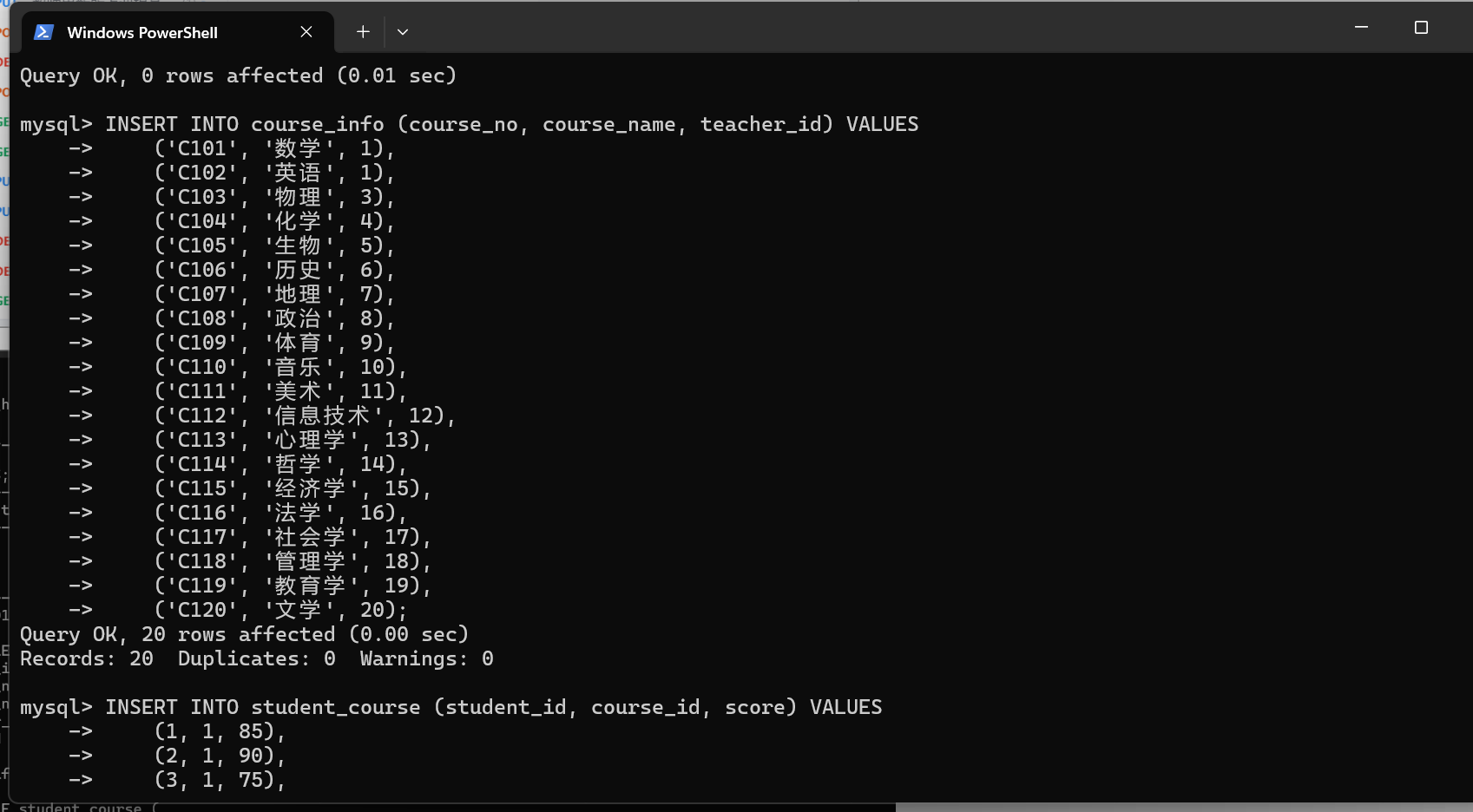
建立学生表并且添加信息；

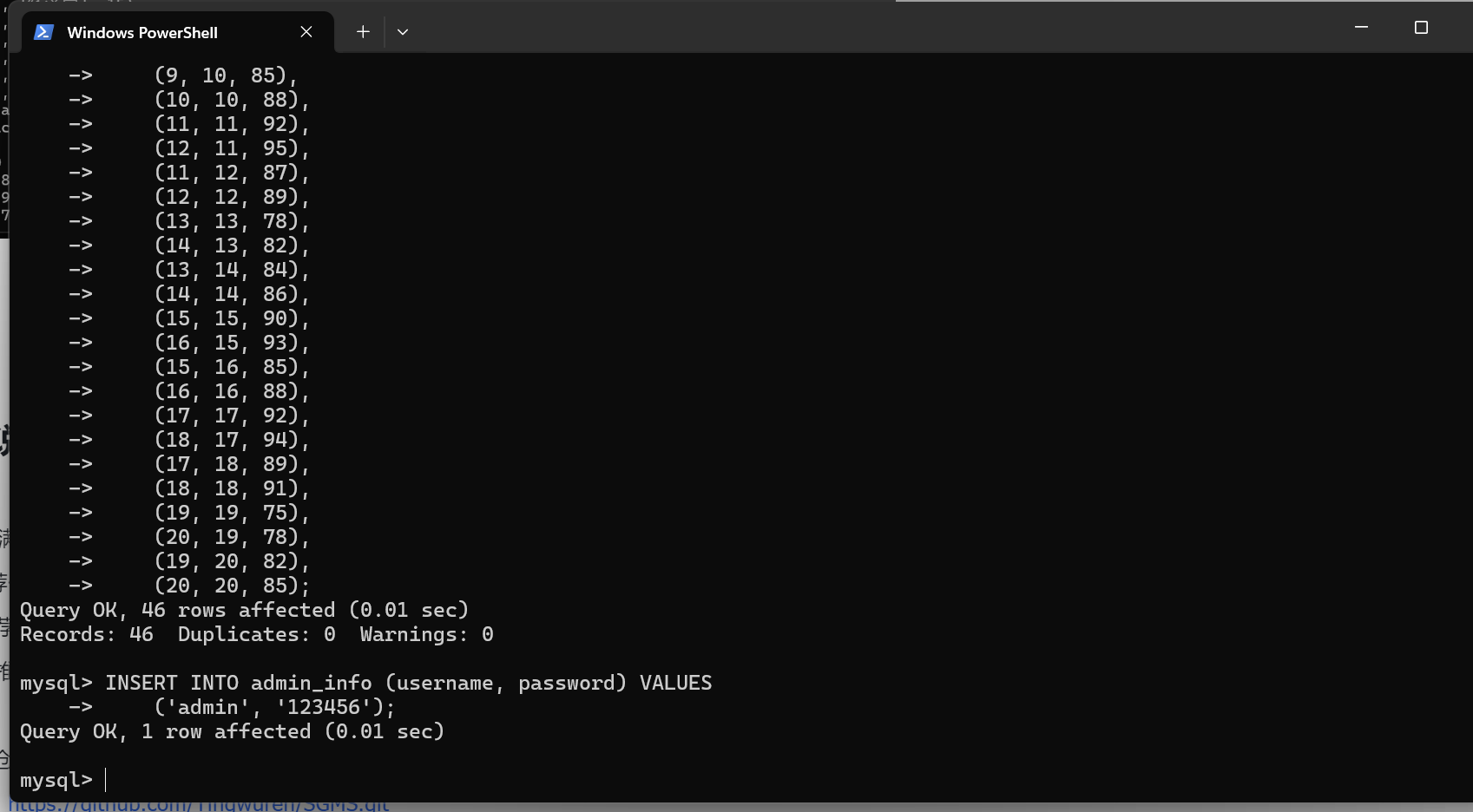


在apifox测试学生登录系统

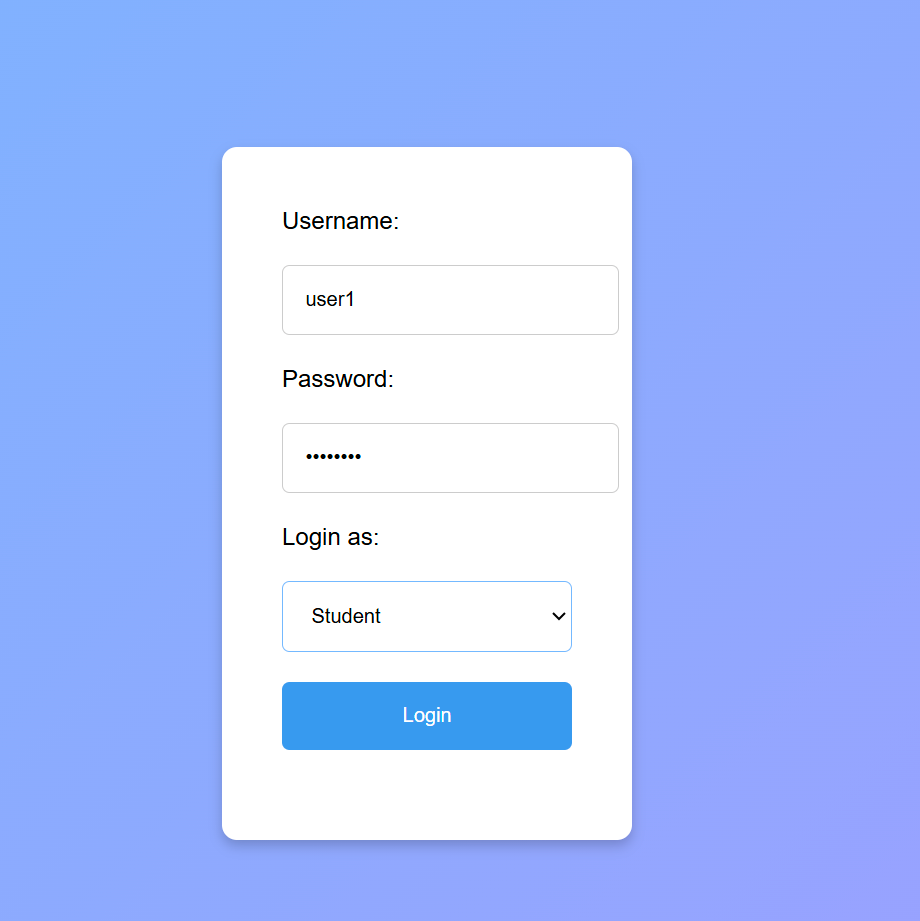








**学生系统**



登陆成功



查询成绩：



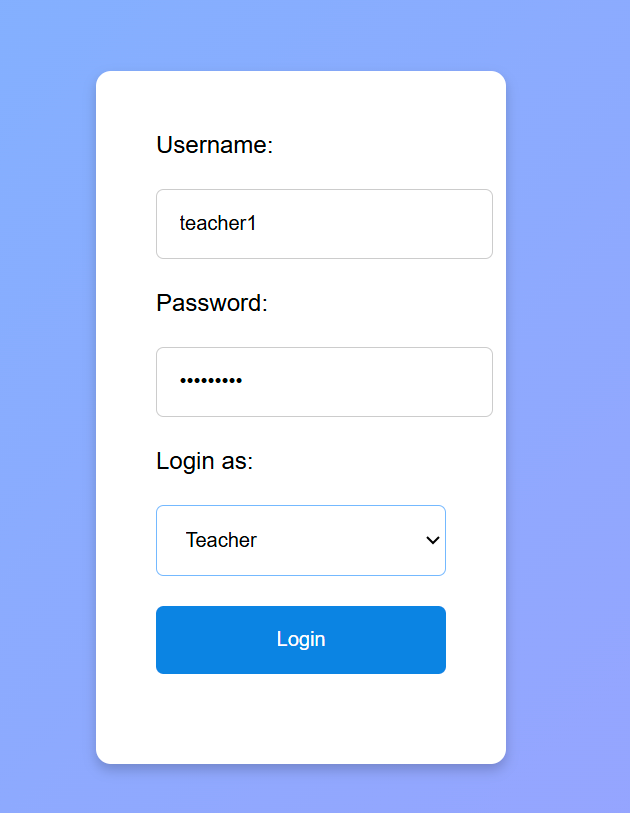
修改密码：这里我们将user1的密码修改为password1.1



点击修改密码后，页面会跳转到登陆界面，这时我们再输入新设的密码就可以成功登录。

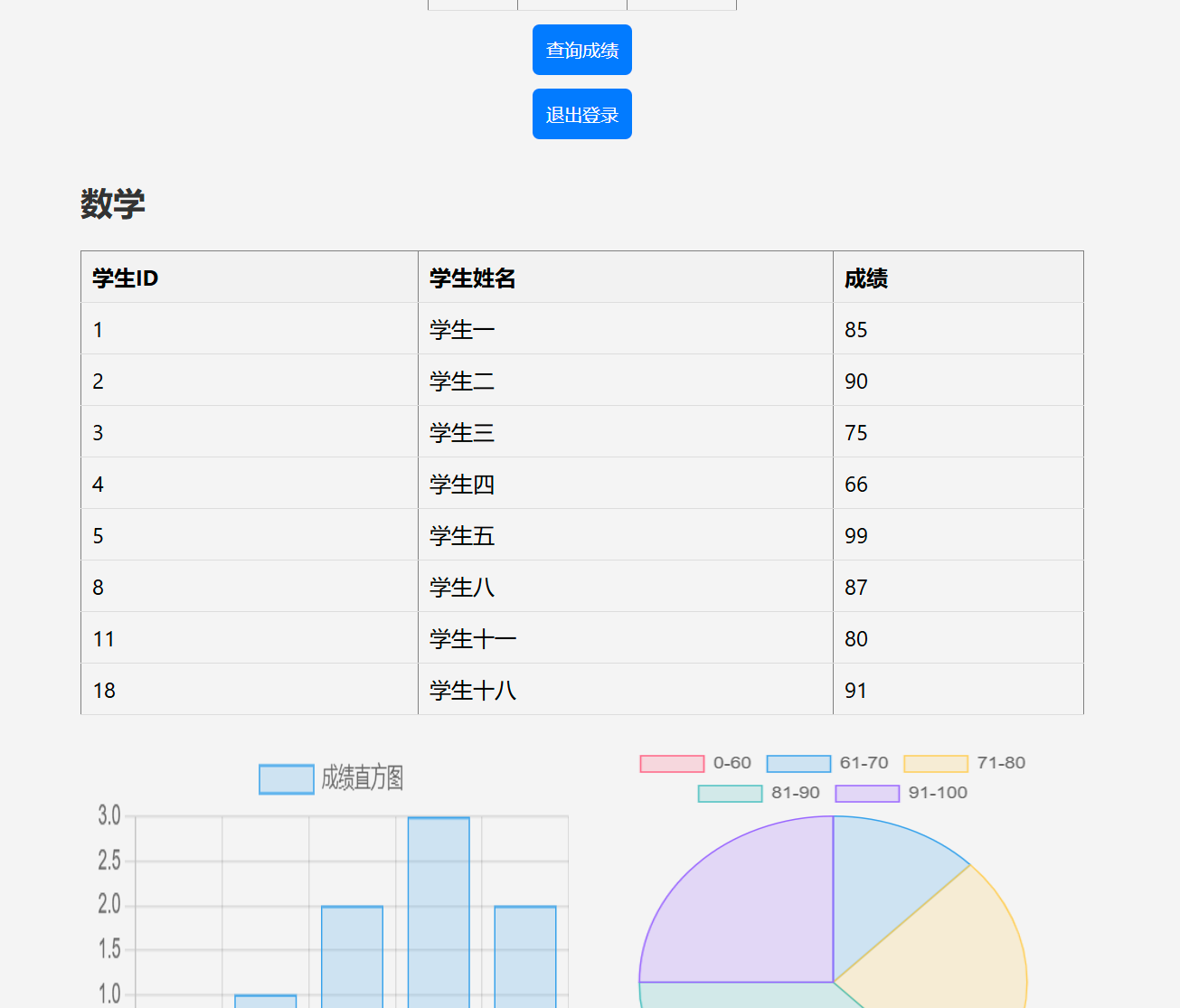
**教师系统**

登录：用户名：teacher1;密码：password1

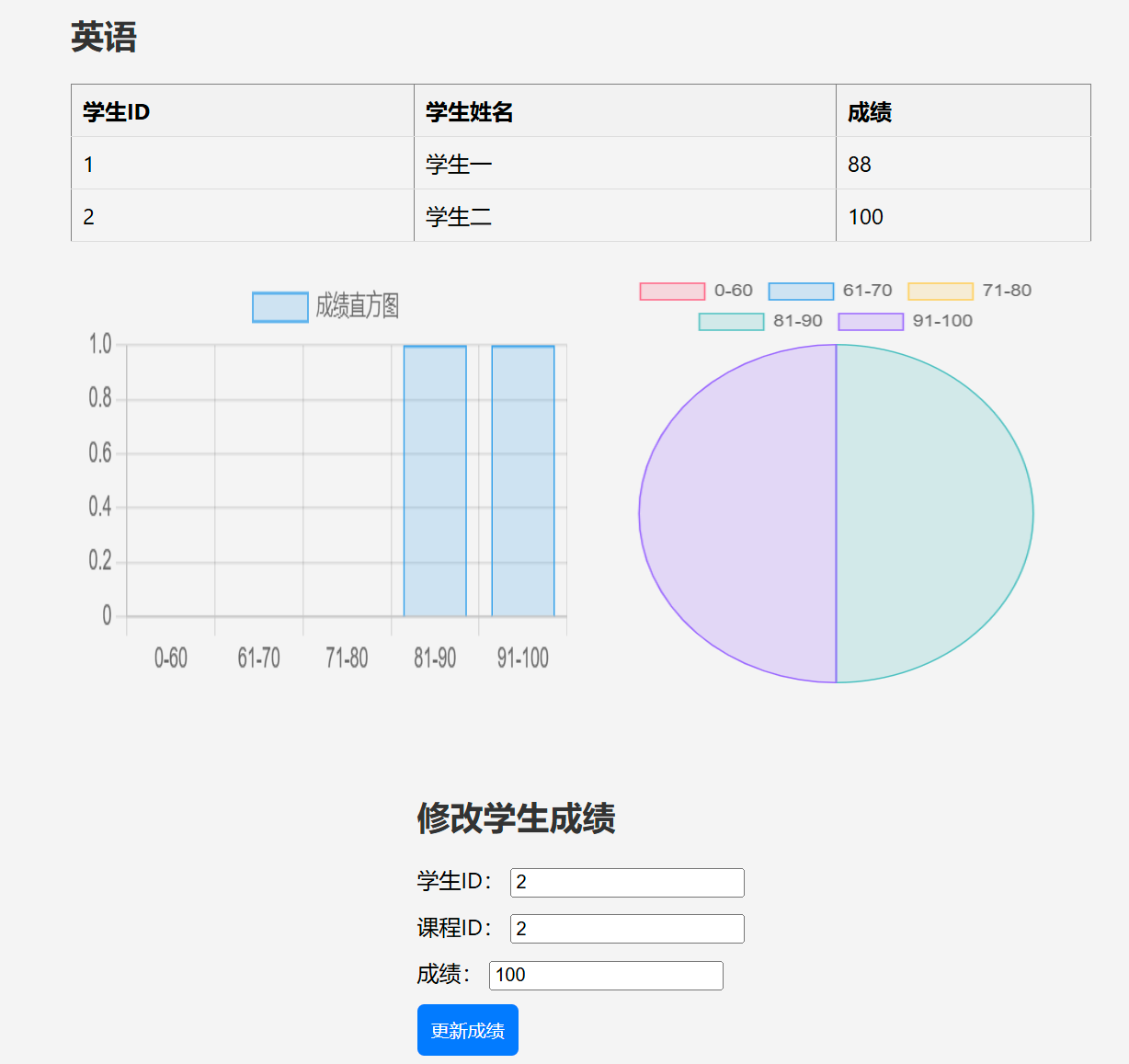




点击查询成绩：这里显示了每个学生的成绩



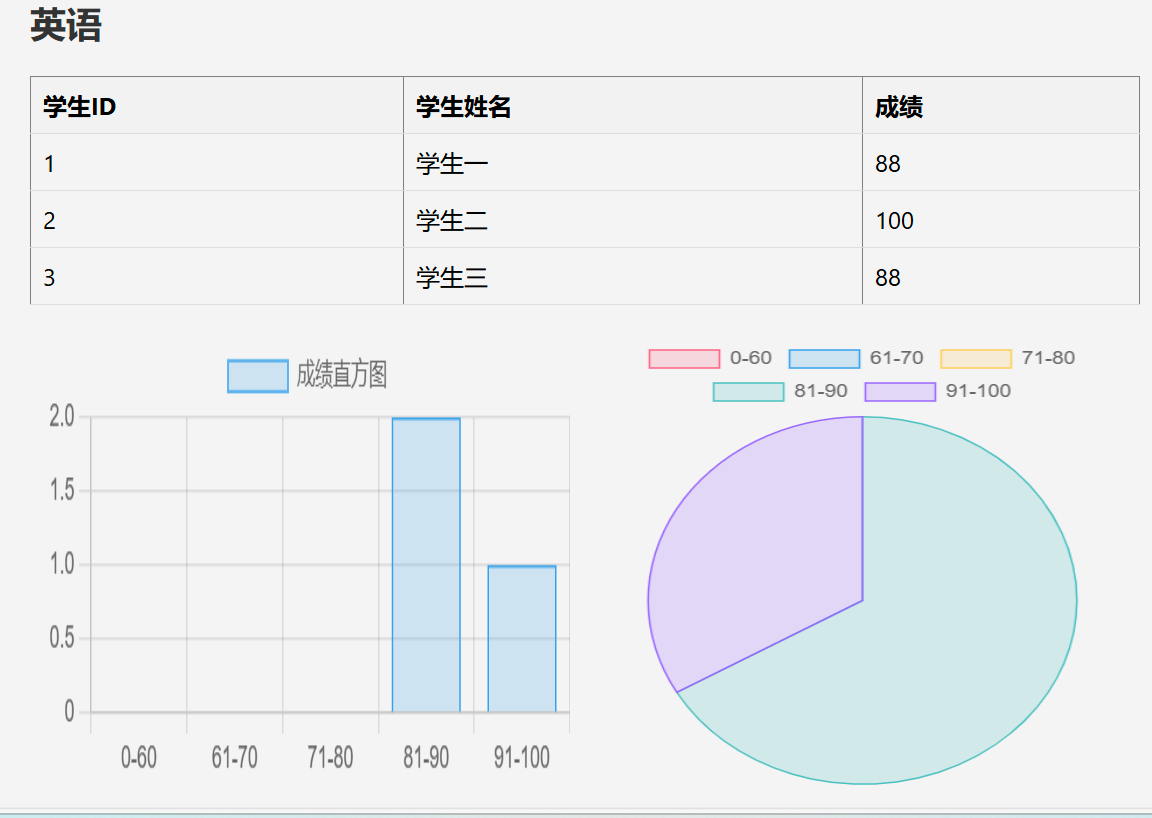
更改成绩，我们将学生二的英语成绩改为100；



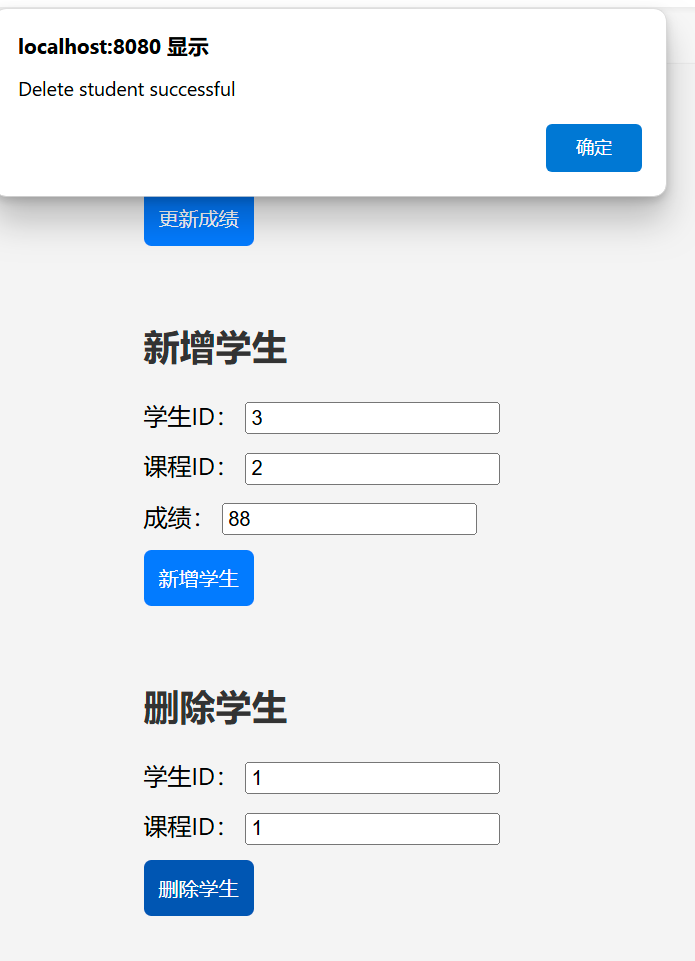
更改成功！

新增学生成绩：新增学生3的英语成绩为88





删除学生成绩：删除学生1的数学成绩

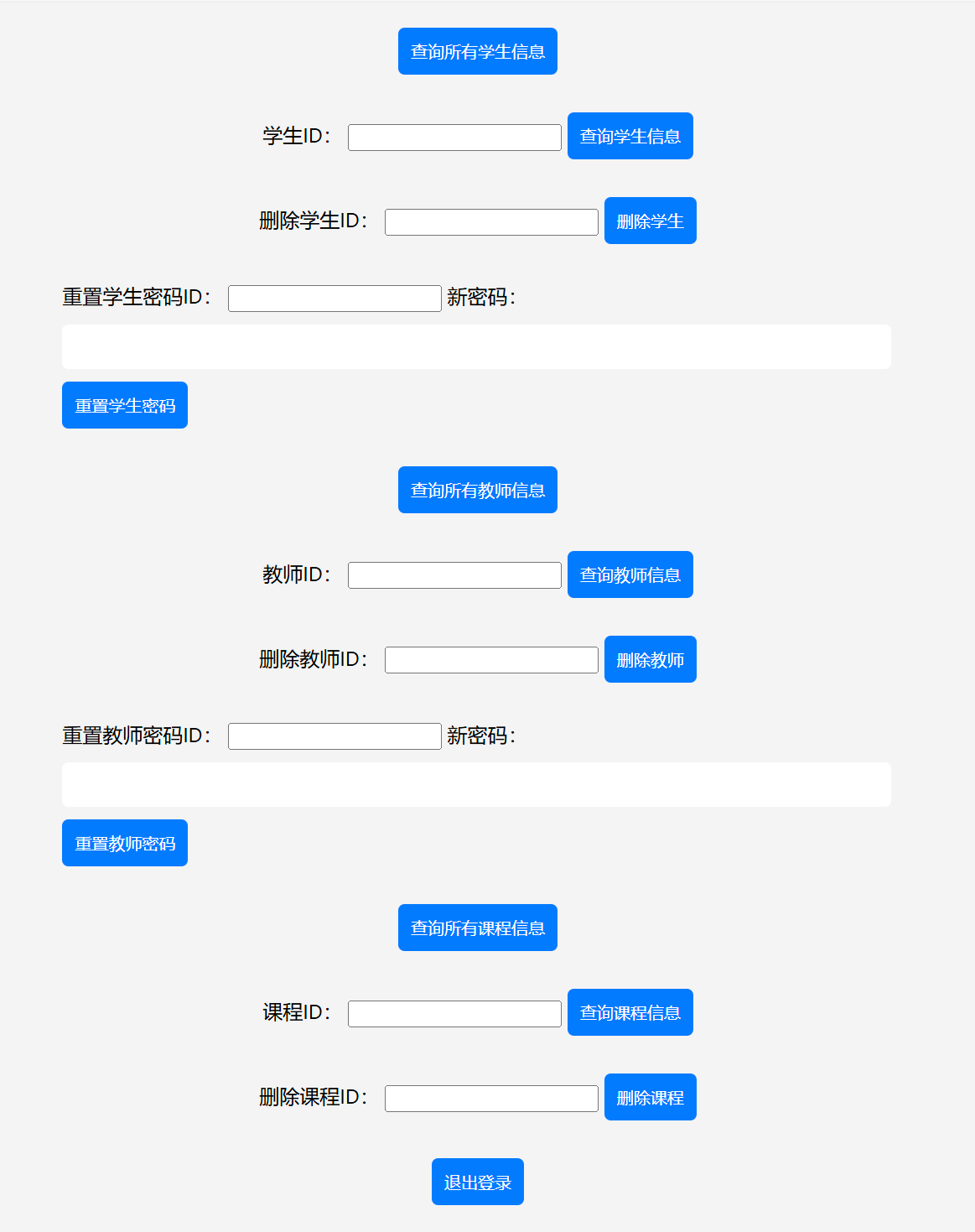




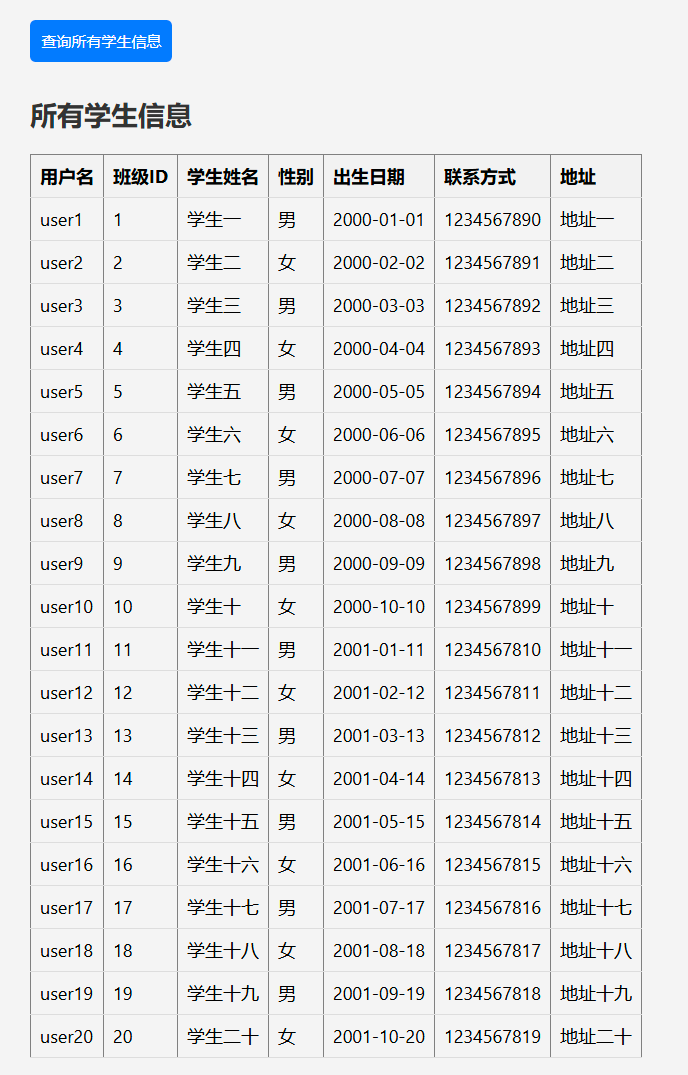
修改密码：同学生系统，点击修改密码会跳到登录界面，输入新设的密码成功登录；

**管理员系统：**

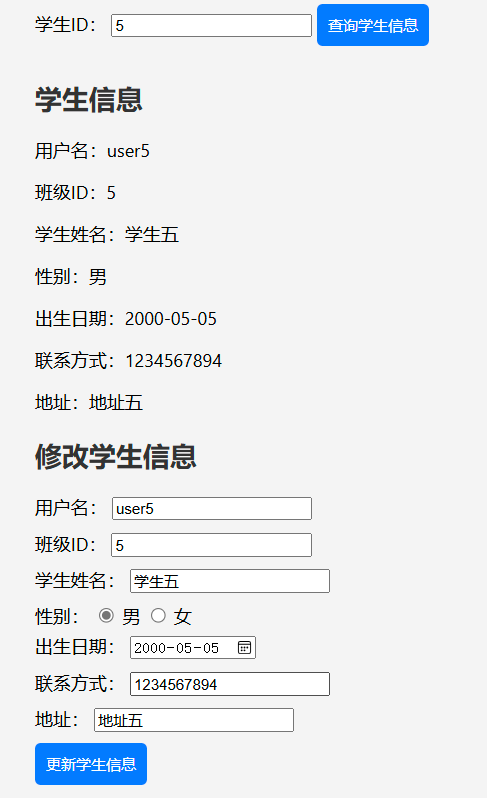
登录：用户名：admin;密码：123456



查询所有学生信息



查询修改学生信息:输入学生ID为5



将出生日期改为2002-05-05



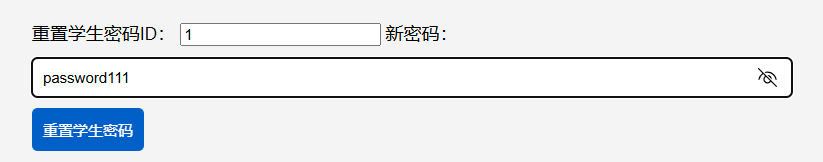
删除学生ID:删除ID为5的学生



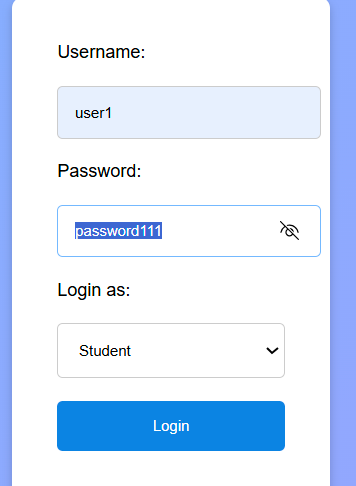
再次查询学生5：显示已被删除



重置学生密码：将学生1的密码改为password111



验证：输入password111,登陆成功





查询所有教师

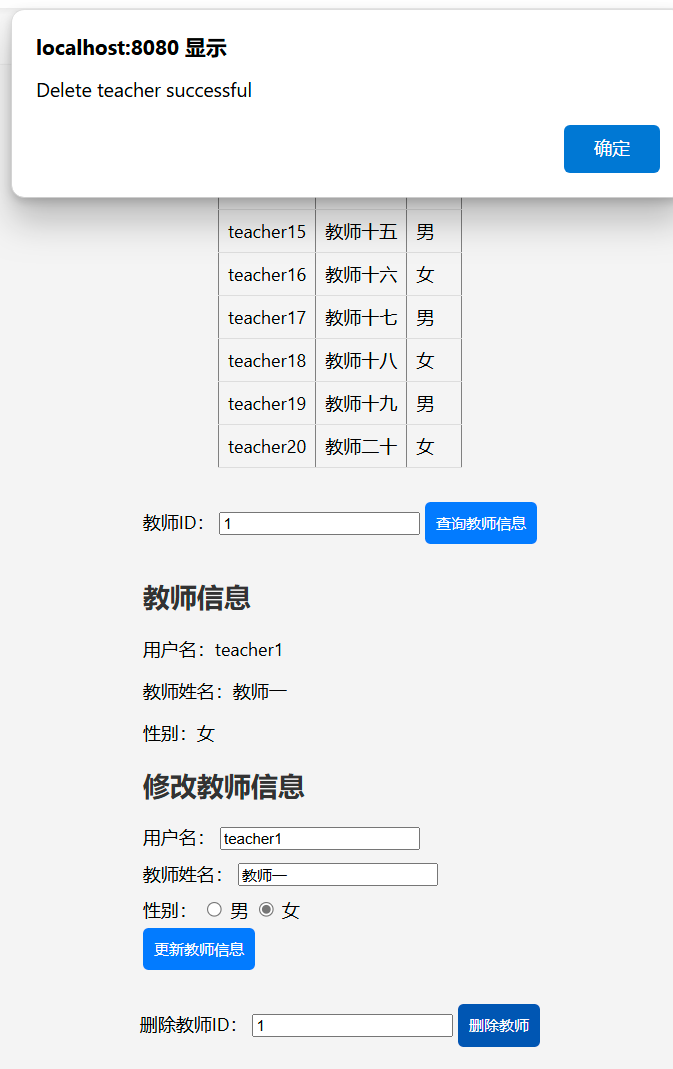


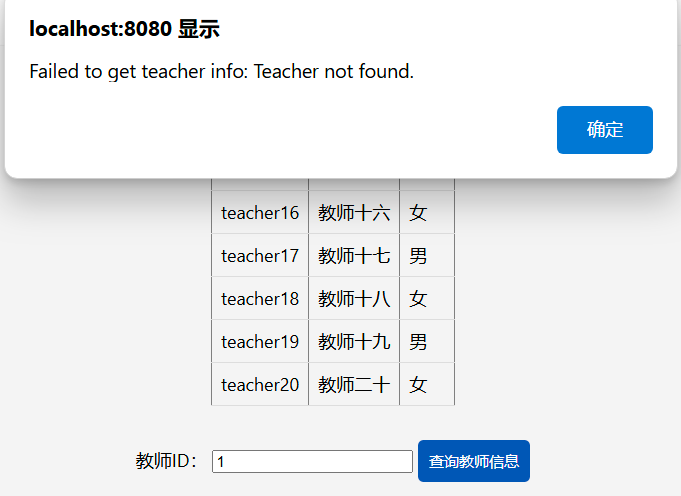
查询并修改教师，我们将教师1的性别改成女



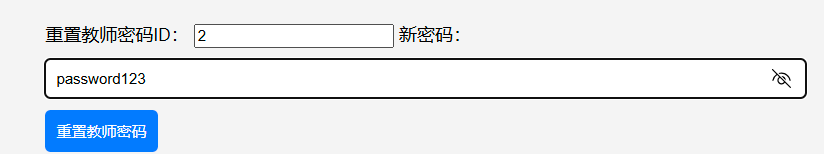


删除教师：

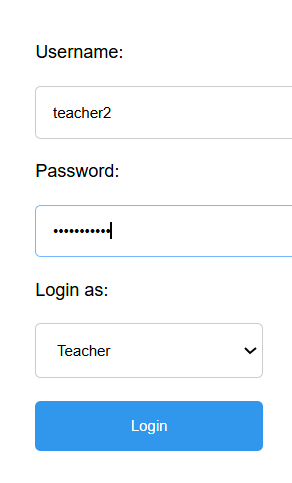




重置教师密码：



验证：输入password123,成功登录





查询并修改课程信息



将物理改为量子力学；

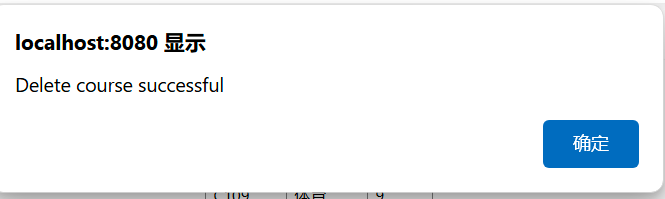


再次查询：

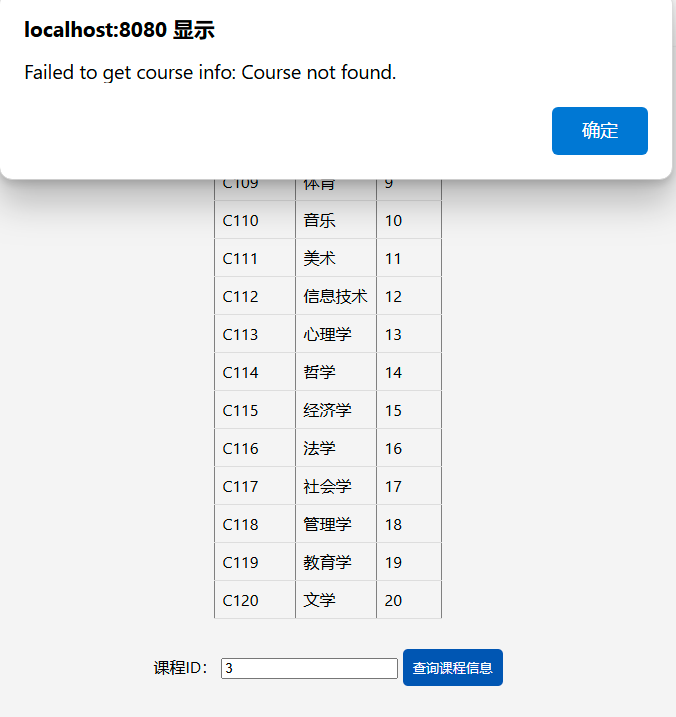


删除课程：





再次查询课程id为3的课程



**5 软件使用说明书**

1. 环境准备

确保您的开发环境满足以下要求：

* 操作系统：推荐使用 Windows 10 或 macOS 10.15+
* Java 版本：推荐 Java 11+
* MySQL 版本：推荐 MySQL 5.7+

1. 下载和安装

您可以从 GitHub 仓库下载项目代码：

* 项目仓库链接：https://github.com/Tingwuren/SGMS.git

下载并解压项目源代码到您的本地开发环境。

1. 配置数据库
2. 在 MySQL 中创建一个新数据库，student\_db。
3. 执行 schema.sql 脚本创建表结构，执行 data.sql 脚本初始化数据
4. 配置 Spring Boot 应用，导入 pom.xml 依赖

|  |
| --- |
| XML <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>cn.edu.bupt</groupId>  <artifactId>SGMS</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>SGMS</name>  <description>SGMS</description>  <properties>  <java.version>1.8</java.version>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <spring-boot.version>2.6.13</spring-boot.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  <optional>true</optional>  </dependency>  <dependency>  <groupId>com.mysql</groupId>  <artifactId>mysql-connector-j</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <optional>true</optional>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>3.5.1</version>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-dependencies</artifactId>  <version>${spring-boot.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>   <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.1</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <version>${spring-boot.version}</version>  <configuration>  <mainClass>cn.edu.bupt.sgms.SgmsApplication</mainClass>  <skip>true</skip>  </configuration>  <executions>  <execution>  <id>repackage</id>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build>  </project> |

1. 数据库连接配置：

在 application.properties 文件中添加以下配置：

|  |
| --- |
| Properties spring.datasource.url=jdbc:mysql://localhost:3306/student\_db spring.datasource.username=root spring.datasource.password=password spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver |

1. MyBatis Plus 配置：

在 Spring Boot 主类上添加 @MapperScan 注解，指定 MyBatis Mapper 扫描路径。

1. 前端开发集成

JavaScript 集成：

在前端页面中使用 AJAX 调用后端提供的 API 接口，以实现数据交互和展示。

1. 启动和测试
2. 在 IDE 中打开项目，并配置 JDK 和 Maven。
3. 运行 Spring Boot 主类，启动应用程序。
4. 打开浏览器访问 http://localhost:8080，确保能够访问项目首页并正常显示。

**6 课程设计小结**

1. 系统总体功能要求

本系统旨在为学生、教师和管理员提供一个便捷的成绩管理平台。主要功能包括学生查询成绩，教师查询、录入、修改成绩，管理员查询、录入、修改基本信息。通过明确的角色权限设置，确保数据的安全性和系统的稳定性。

1. 人员角色权限

* **学生：**
* 仅允许查询自己的成绩和基本信息。
* **教师：**
* 查询成绩和个人基本信息，录入和修改成绩。
* **管理员：**
* 查询、录入、修改人员和课程的基本信息。

1. 主要功能模块

* **查询模块：**
* 学生查询自己的基本信息和成绩。
* 教师查询成绩和个人基本信息。
* 管理员查询人员和课程的基本信息。
* **录入模块：**
* 教师录入成绩。
* 管理员录入人员和课程信息。
* **修改模块：**
* 教师修改成绩。
* 管理员修改人员和课程信息。
* **系统配置模块：**
* 密码修改和重置。
* **统计模块：**
* 统计各分数段学生的成绩分布，生成直方图和饼图。

1. 数据库表设计

* **学生表 (StudentInfo)：** 存储学生的基本信息。
* **教师表 (TeacherInfo)：** 存储教师的基本信息。
* **专业班级表 (ClassInfo)：** 存储专业和班级信息。
* **课程表 (CourseInfo)：** 存储课程信息。
* **选课成绩表 (StudentCourse)：** 存储学生选课及其成绩信息。

1. 系统功能流图

* **用户角色：** 系统用户分为管理员、教师和学生三类。所有用户均需通过用户名和口令进行登录，登录成功后根据角色权限访问相应功能。
* **学生功能：**
* 登录后查询自己的基本信息和成绩。
* 不能查看其他学生的信息。
* **教师功能：**
* 登录后查询个人基本信息和所授课程的基本信息。
* 查询所授课程所有学生的成绩。
* 录入、修改和删除所授课程学生的成绩。
* **管理员功能：**
* 登录后查询、修改、删除人员和课程的基本信息。

在本次课程设计中，我们从需求分析入手，设计了一个以角色为中心的学生成绩管理系统。通过明确的角色权限划分，确保系统的安全性和数据的准确性。在数据库设计过程中，我们设计了合理的表结构，以便高效地存储和查询数据。

在实现系统功能时，我们采用了Spring Boot框架，并结合了MyBatis和Thymeleaf，确保系统的高效性和用户界面的友好性。通过对数据库连接和查询的优化，提高了系统的响应速度。

**7参考文献**

1. Spring Boot 官方文档：<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
2. MyBatis 官方文档：https://mybatis.org/mybatis-3/
3. Thymeleaf 官方文档：https://www.thymeleaf.org/documentation.html
4. Java Persistence with Hibernate, Christian Bauer, Gavin King, Gary Gregory, Manning Publications
5. https://blog.csdn.net/qq\_45173404/category\_10225261.html
6. https://blog.csdn.net/qq\_45173404/category\_10270690.html
7. https://blog.csdn.net/qq\_45173404/article/details/115712758
8. https://www.runoob.com/mysql/mysql-tutorial.html

**8其他说明**

**可能遇到的问题及解决方案**

**数据库连接失败**：

* 确保application.properties文件中数据库连接配置正确。
* 确认数据库服务已启动，并且数据库用户和密码正确。

**端口占用**：

* 如果8080端口被占用，可以通过修改application.properties文件中的server.port属性更改端口。

**用户认证和授权**：

* 确保Spring Security配置正确，实现不同角色的访问控制。
* 检查请求头中的Authorization字段是否存在并且格式正确。

**数据查询失败**：

* 检查SQL语句是否正确，数据库表和字段是否存在。
* 使用Spring Data JPA或MyBatis Plus的日志功能，查看具体的SQL执行情况。

**未来改进方向**

* **增强安全性：** 实现更复杂的权限管理和数据加密。
* **用户体验优化：** 改进用户界面设计，使其更加直观和易用。
* **功能扩展：** 增加更多统计和分析功能，为教学管理提供更全面的数据支持