**《课程管理系统》概要设计**

**一、系统架构设计**

课程管理系统采用多层架构设计，包括表现层、业务逻辑层、数据访问层和数据库层，各层之间相互协作，实现系统功能。

**（一）表现层**

1. **设计目标**  
   为用户（管理员、教师、学生）提供简洁、直观、易用的交互界面。通过图形化界面元素，使用户能够方便地执行各种操作，如信息查询、数据输入、选课、成绩管理等。
2. **技术选型**  
   采用 Web 界面技术，如 HTML、CSS、JavaScript 等构建前端页面。结合前端框架（如 Vue.js、React.js 等）实现动态交互效果和组件化开发，提高开发效率和页面性能。

**（二）业务逻辑层**

1. **设计目标**  
   处理系统的核心业务逻辑，包括课程信息管理逻辑、课程安排逻辑、选课逻辑、成绩管理逻辑、用户权限管理逻辑等。它是连接表现层和数据访问层的桥梁，负责协调各层之间的数据交互和业务处理流程。
2. **功能模块划分**
   * **课程管理模块**：处理课程的添加、修改、删除、查询等操作相关的业务逻辑。例如，在添加课程时，要验证课程编号的唯一性、检查课程信息的完整性等。
   * **课程安排模块**：根据教学资源和课程要求，实现课程安排的算法和逻辑。包括检查教师和教室的可用性、避免时间冲突等功能。
   * **选课管理模块**：处理学生选课和退选的业务逻辑，如检查选课冲突、更新选课信息、处理选课人数限制等。
   * **成绩管理模块**：负责教师录入成绩、成绩计算、成绩查询和修改的业务逻辑。根据设定的成绩计算规则（如平时成绩和考试成绩的权重）计算总成绩，并对成绩修改进行权限验证。
   * **用户权限管理模块**：根据用户类型（管理员、教师、学生）分配不同的操作权限，确保系统的安全性和数据的保密性。例如，管理员具有最高权限，可以对所有功能进行操作；教师只能操作与自己授课相关的功能；学生只能进行选课、查询成绩等操作。

**（三）数据访问层**

1. **设计目标**  
   负责与数据库进行交互，实现对数据库中数据的增、删、改、查操作。将业务逻辑层的请求转换为对数据库的具体操作，并将结果返回给业务逻辑层。通过抽象数据访问接口，提高系统的可维护性和可扩展性，方便在不同的数据库环境下进行切换。
2. **技术选型**  
   使用数据库访问框架，如 MyBatis（Java 环境下）或其他适合的框架。这些框架可以方便地编写 SQL 语句与 Java 对象（或其他编程语言的对象）之间的映射关系，减少代码的复杂性和重复性。

**（四）数据库层**

1. **设计目标**  
   存储系统的所有数据，包括课程信息、教师信息、学生信息、课程安排信息、选课信息、成绩信息等。设计合理的数据库结构，保证数据的完整性、一致性和高效存储。
2. **数据库选型**  
   选择关系型数据库，如 MySQL、Oracle 或 SQL Server 等。这些数据库具有成熟的技术支持、强大的数据存储和管理能力，适合处理课程管理系统这种结构化数据较多的应用场景。

**二、数据库设计**

**（一）数据库概念模型设计（E - R 模型）**

图示

描述已自动生成

**三、系统安全设计**

**（一）用户认证与授权**

1. **用户认证**  
   采用用户名和密码的方式进行用户认证。用户在登录界面输入账号和密码后，系统将对其进行验证。为了提高安全性，可以采用加密算法（如 MD5、SHA - 256 等）对密码进行加密存储和传输。同时，可以设置密码强度要求，如长度、包含字母和数字等。
2. **授权管理**  
   根据用户类型（管理员、教师、学生）在系统中设置不同的角色和权限。通过角色 - 权限映射关系，为每个用户分配相应的操作权限。例如，管理员角色具有所有功能的操作权限，教师角色只能操作与教学相关的功能，学生角色只能进行选课和成绩查询等操作。在系统实现过程中，可以使用权限管理框架（如 Spring Security 等）来实现细粒度的权限控制。

**（二）数据安全**

1. **数据加密**  
   对系统中的敏感数据，如学生成绩、教师联系方式、用户密码等进行加密存储。在数据传输过程中，采用安全的传输协议（如 HTTPS），确保数据在网络中的保密性和完整性。
2. **数据备份与恢复**  
   定期对数据库进行备份，可以采用全量备份和增量备份相结合的方式。同时，建立数据恢复机制，以便在出现数据丢失、损坏等情况时能够快速恢复数据。可以使用数据库自带的备份和恢复工具，也可以编写自定义的备份脚本。

**（三）系统防护**

1. **防止 SQL 注入攻击**  
   在数据访问层，对用户输入的数据进行严格的过滤和验证，防止恶意用户通过 SQL 注入攻击获取或篡改系统数据。可以使用参数化查询等技术来避免 SQL 语句的拼接，从而降低 SQL 注入的风险。
2. **防止 XSS 攻击**  
   在表现层，对用户输入的内容进行 HTML 编码处理，防止跨站脚本攻击（XSS）。同时，对从数据库中读取的数据在显示到页面之前也进行相应的编码处理，确保用户界面的安全性。