

**网络安全社团网站项目**

**详**

**细**

**设**

**计**

**说**

**明**

**书**

G14

目录

[1. 引言 1](#_Toc182763518)

[1.1. 编写目的 1](#_Toc182763519)

[1.2. 项目背景 1](#_Toc182763520)

[1.3. 参考资料 1](#_Toc182763521)

[2. 总体设计 2](#_Toc182763522)

[2.1. 需求概述 2](#_Toc182763523)

[2.2. 软件结构 2](#_Toc182763524)

[3. 程序描述 4](#_Toc182763525)

[3.1. 登陆模块 4](#_Toc182763526)

[3.1.1. 功能流程图 4](#_Toc182763527)

[3.1.2. 功能描述 5](#_Toc182763528)

[3.1.3. 界面设计 6](#_Toc182763529)

[3.1.4. 伪代码 6](#_Toc182763530)

[3.1.5. 关键算法 10](#_Toc182763531)

[3.2. 报名管理模块 12](#_Toc182763532)

[3.2.1. 功能流程图 12](#_Toc182763533)

[3.2.2. 功能描述 12](#_Toc182763534)

[3.2.3. 界面设计 13](#_Toc182763535)

[3.2.4. 伪代码 13](#_Toc182763536)

[3.2.5. 关键算法 14](#_Toc182763537)

[3.3. 活动管理模块 15](#_Toc182763538)

[3.3.1. 功能流程图 15](#_Toc182763539)

[3.3.2. 功能描述 16](#_Toc182763540)

[3.3.3. 界面设计 16](#_Toc182763541)

[3.3.4. 伪代码 16](#_Toc182763542)

[3.3.5. 关键算法 19](#_Toc182763543)

[3.4. 用户管理模块 20](#_Toc182763544)

[3.4.1. 功能流程图 20](#_Toc182763545)

[3.4.2. 功能描述 20](#_Toc182763546)

[3.4.3. 界面设计 20](#_Toc182763547)

[3.4.4. 伪代码 20](#_Toc182763548)

[3.4.5. 关键算法 23](#_Toc182763549)

[3.5. 个人中心 24](#_Toc182763550)

[3.5.1. 功能流程图 24](#_Toc182763551)

[3.5.2. 功能描述 24](#_Toc182763552)

[3.5.3. 界面设计 24](#_Toc182763553)

[3.5.4. 伪代码 24](#_Toc182763554)

[3.5.5. 关键算法 27](#_Toc182763555)

[3.6. 浏览主页模块 27](#_Toc182763556)

[3.6.1. 功能流程图 27](#_Toc182763557)

[3.6.2. 功能描述 28](#_Toc182763558)

[3.6.3. 界面设计 28](#_Toc182763559)

[3.6.4. 伪代码 28](#_Toc182763560)

[3.6.5. 关键算法 29](#_Toc182763561)

[3.7. 学生还书 29](#_Toc182763562)

[3.7.1. 功能流程图 30](#_Toc182763563)

[3.7.2. 功能描述 30](#_Toc182763564)

[3.7.3. 界面设计 30](#_Toc182763565)

[3.7.4. 伪代码 30](#_Toc182763566)

[3.7.5. 关键算法 30](#_Toc182763567)

[3.8. 图书注销 30](#_Toc182763568)

[3.8.1. 功能流程图 30](#_Toc182763569)

[3.8.2. 功能描述 30](#_Toc182763570)

[3.8.3. 界面设计 30](#_Toc182763571)

[3.8.4. 伪代码 30](#_Toc182763572)

[3.8.5. 关键算法 30](#_Toc182763573)

­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­

# 引言

## 编写目的

本软件详细设计说明书的编写目的是对浙大城市学院网络安全社团管理系统的详细设计进行全面说明，为开发团队提供清晰的指导，确保系统的设计、开发和实现能够严格按照需求规范执行。本说明书将细化系统的功能模块设计、数据结构、算法流程和接口定义，帮助开发人员准确理解每个模块的实现方法。此外，本说明书还将作为后期系统维护和升级的重要参考文档，支持系统的可扩展性和可维护性。

## 项目背景

浙大城市学院网络安全社团管理系统是为浙大城市学院网络安全社团设计的一款线上管理平台，旨在提升社团活动的管理效率和学生用户的参与体验。该系统主要用于发布社团风采展示和活动信息，为学生提供在线报名和状态查询功能，同时支持管理员对活动信息、用户数据和报名记录的管理。

传统的手工管理方式无法满足现代化高校社团的需求，存在信息分散、管理效率低下等问题。网络安全社团管理系统通过信息化手段，实现活动信息的集中管理和在线操作，不仅提升了社团的运营效率，还为学生用户提供了更便捷的参赛渠道和更高效的反馈机制。

系统功能包括用户注册与登录、活动信息发布与展示、在线报名与审核、通知发送、数据统计与导出等模块。系统以Spring Boot为开发框架，采用分层架构设计，确保性能稳定、安全可靠，并具备良好的扩展性和可维护性。

## 参考资料

# 总体设计

## 需求概述

网络安全社团管理系统旨在为高校网络安全社团提供一个集活动信息发布、在线报名管理、用户通知和数据统计于一体的综合管理平台。系统需满足普通用户（学生）和管理员两种角色的功能需求，同时兼顾高效性能和数据安全性。

管理员：

活动发布与管理：管理员需能够发布、编辑和删除活动信息，设置报名截止日期和活动规则等。

报名审核：管理员需能够查看用户的报名信息，进行审核操作，并设置报名状态（通过或拒绝）。

用户管理：管理员需能够查看和管理注册用户的信息，包括禁用账户、修改信息等操作。

统计与导出：管理员需能够导出活动的报名数据，用于后续分析和管理。

操作日志记录：系统需记录管理员操作日志，便于审计和问题追踪。

用户需求：

注册与登录：普通用户需能够通过系统注册账户，并使用邮箱和密码登录。

活动信息查看：用户需能查看所有社团发布的活动，包括活动的详细信息（如时间、地点、规则等）。

在线报名：用户需能够在线报名参加活动，填写相关报名信息，并提交给系统。

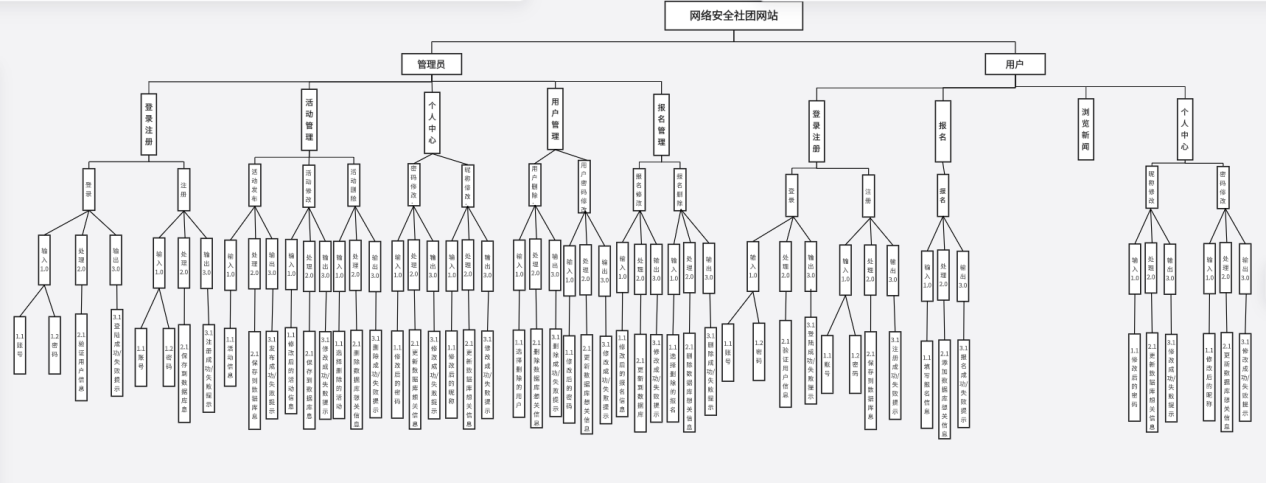
状态查询：用户需在个人中心查询报名审核状态，以及接收系统的通知信息。

个人信息管理：用户需能够更新个人资料，如修改邮箱、联系方式等。

管理员需求

## 软件结构

软件层次结构图：



**系统由2大模块，6小模块组成：**

**序号 编号 名称**

1. 用户管理模块
2. 用户注册
3. 用户登录
4. 用户信息管理
5. 活动发布模块
6. 活动信息修改模块
7. 报名管理模块
8. 后台管理模块
9. 活动管理

10， 用户管理

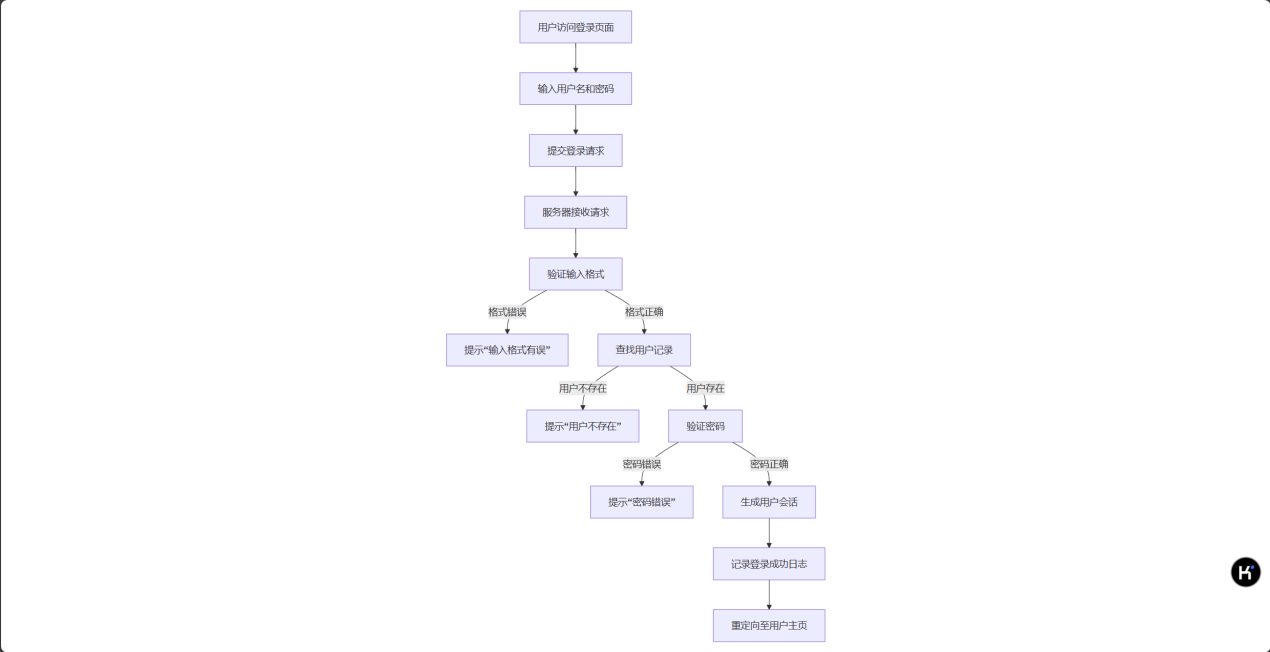
# 程序描述

## 登陆模块

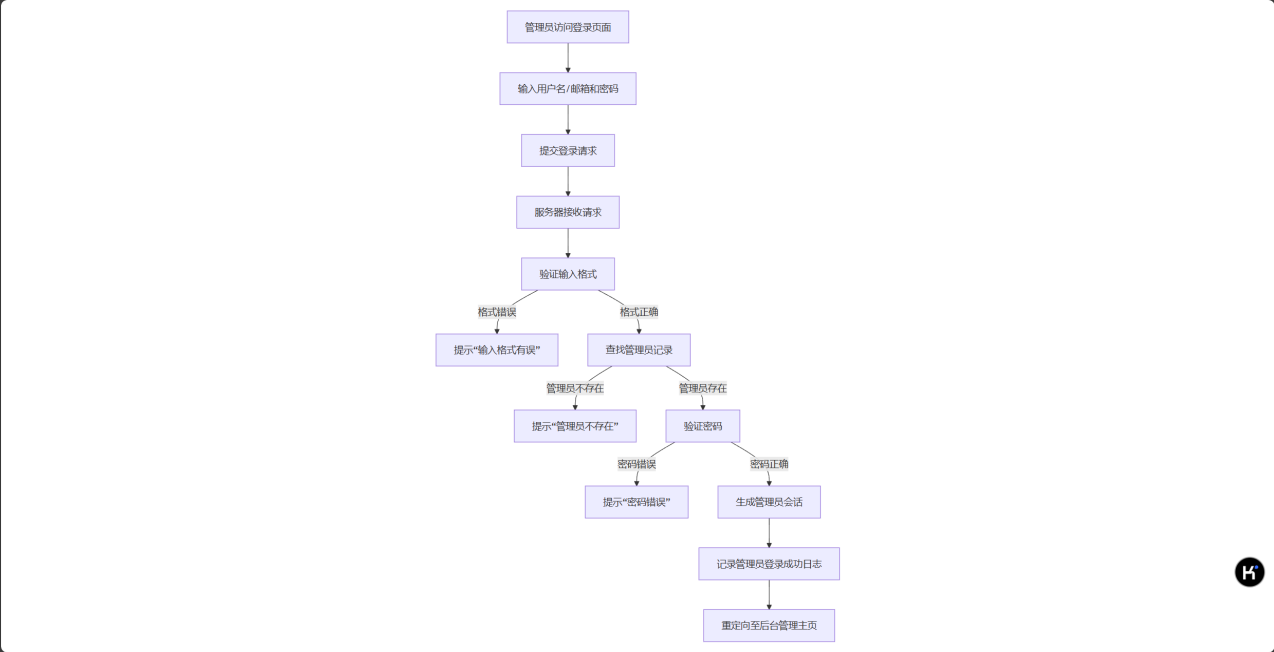
登录模块是网络安全社团管理系统的核心组件之一，负责验证用户和管理员的身份，确保只有授权人员能够访问系统的相关功能。该模块处理用户和管理员的登录请求，验证凭证，管理会话，并实施必要的安全措施以保护系统免受未经授权的访问和攻击。

### 功能流程图

用户登陆



管理员登陆



### 功能描述

**用户登录**

输入：用户名/邮箱、密码

输出：登录成功或失败的消息，生成会话

功能：

验证用户输入的凭证

区分普通用户和管理员

处理登录失败的情况（如错误密码、未注册用户）

记录登录尝试日志

**管理员登录**

输入：用户名/邮箱、密码

输出：登录成功或失败的消息，生成管理员会话

功能：

验证管理员凭证

确保只有具有管理员角色的用户能够登录

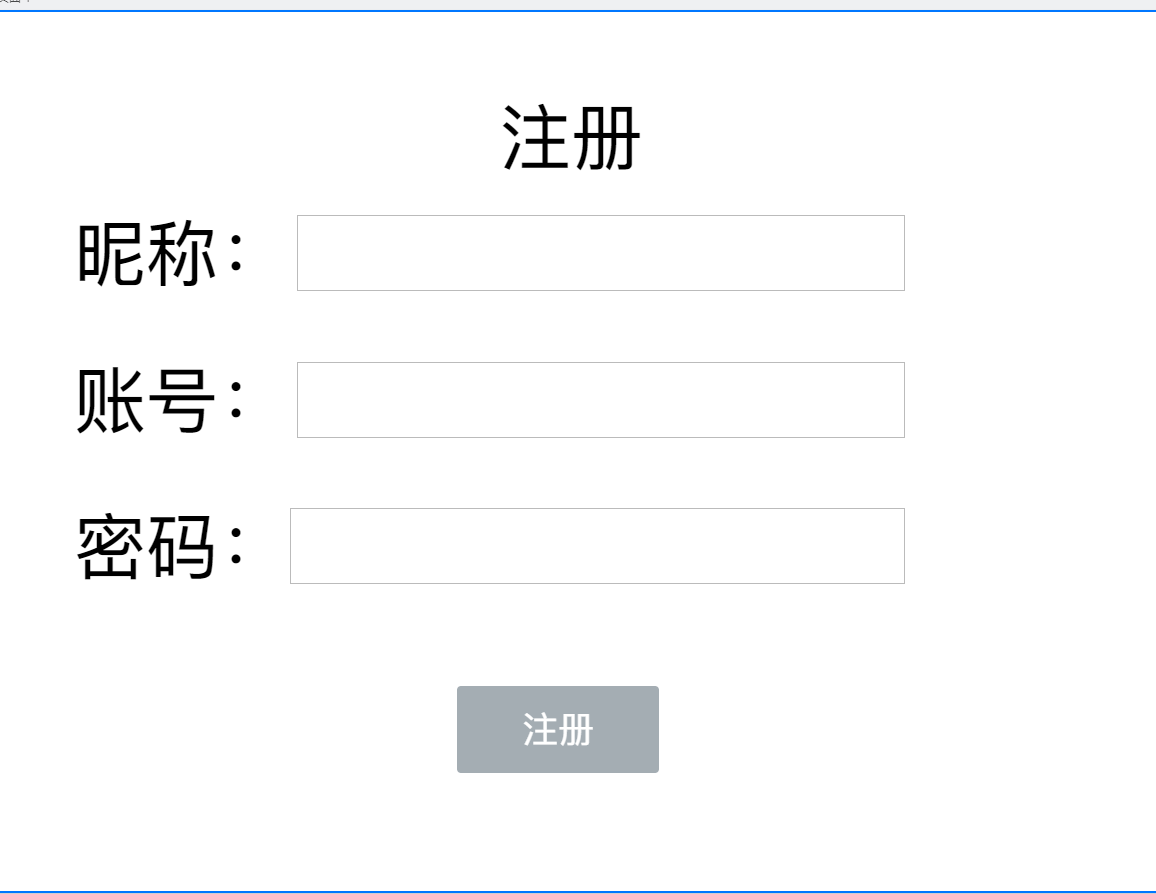
处理登录失败的情况

记录管理员登录日志

### 界面设计

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **字段名称** |  | **长度** | **录入方式** | **是否非空项** | **数据检验** | **默认显示** |
| **用户账号** |  | 10 | 文本框 | Y | Y |  |
| **用户密码**  **用户名** |  | 20  20 | 文本框  文本框 | Y  N | Y  Y |  |





### 伪代码

用户登录：

# 主过程：用户登录

过程 用户登录模块:

输入: 用户名或邮箱（inputUsernameOrEmail）, 密码（inputPassword）

输出: 会话ID（sessionID）或错误消息

# 步骤 1: 验证输入格式

if (not ValidateInput(inputUsernameOrEmail, "usernameOrEmail")) then

return "输入格式有误"

end if

if (not ValidateInput(inputPassword, "password")) then

return "输入格式有误"

end if

# 步骤 2: 查找用户记录

用户 = 数据库查询(Users, {"$or": [{"username": inputUsernameOrEmail}, {"email": inputUsernameOrEmail}]})

if (用户 == 空) then

记录登录尝试("用户不存在", inputUsernameOrEmail)

return "用户不存在"

end if

# 步骤 3: 检查账户是否被锁定

if (当前时间 < 用户.锁定时间) then

return "账户暂时被锁定，请稍后再试"

end if

# 步骤 4: 验证密码

if (密码验证失败(inputPassword, 用户.密码哈希)) then

用户.失败登录尝试次数 = 用户.失败登录尝试次数 + 1

if (用户.失败登录尝试次数 >= 最大失败次数) then

用户.锁定时间 = 当前时间 + 锁定持续时间

记录登录尝试("账户锁定", inputUsernameOrEmail)

return "账户因多次失败尝试而被暂时锁定"

else

记录登录尝试("密码错误", inputUsernameOrEmail)

return "密码错误"

end if

else

# 密码验证成功

用户.失败登录尝试次数 = 0

用户.锁定时间 = 空

# 步骤 5: 生成会话

会话ID = 生成会话ID(用户.用户ID, 用户.角色)

会话 = {

"会话ID": 会话ID,

"用户ID": 用户.用户ID,

"角色": 用户.角色,

"创建时间": 当前时间,

"过期时间": 当前时间 + 会话持续时间

}

数据库插入(Sessions, 会话)

# 步骤 6: 记录成功登录

记录登录尝试("登录成功", inputUsernameOrEmail)

return 会话ID

end if

结束过程

# 辅助过程：验证输入格式

过程 ValidateInput(input, type):

if (type == "usernameOrEmail") then

return RegexMatch("^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$", input) or RegexMatch("^[a-zA-Z0-9\_]{3,20}$", input)

else if (type == "password") then

return RegexMatch("^(?=.\*[A-Za-z])(?=.\*\d)[A-Za-z\d]{8,}$", input)

else

return False

end if

end process

# 辅助过程：密码验证

过程 密码验证失败(inputPassword, storedHashedPassword):

return not BCrypt.verify(inputPassword, storedHashedPassword)

end process

# 辅助过程：记录登录尝试

过程 记录登录尝试(status, usernameOrEmail):

日志条目 = {

"用户名或邮箱": usernameOrEmail,

"状态": status,

"时间戳": 当前时间,

"IP地址": 获取用户IP(),

"用户代理": 获取用户代理()

}

数据库插入(LoginLogs, 日志条目)

end process

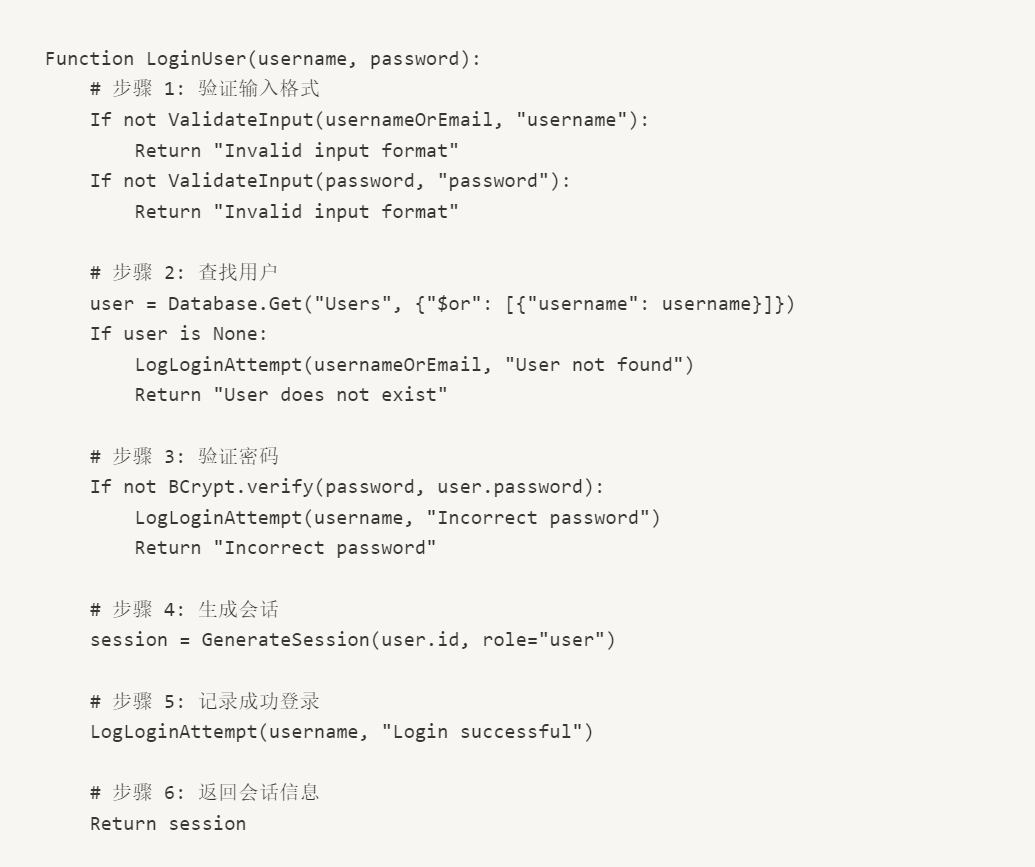
# 辅助过程：生成会话ID

过程 生成会话ID(userID, role):

return UUID生成()

end process

### 关键算法



圈复杂度分析

用户登录模块总圈复杂度：6

决策点：

输入格式验证（2个if）

用户存在性检查（1个if）

账户锁定检查（1个if）

密码验证（1个if）

失败尝试次数检查（1个if）



圈复杂度计算：

决策点：

If not ValidateInput(usernamel, "username") → +1

If not ValidateInput(password, "password") → +1

If admin is None → +1

If not BCrypt.verify(password, admin.password) → +1

基本圈复杂度公式：M = E - N + 2P

E（边数）= 10

N（节点数）= 6

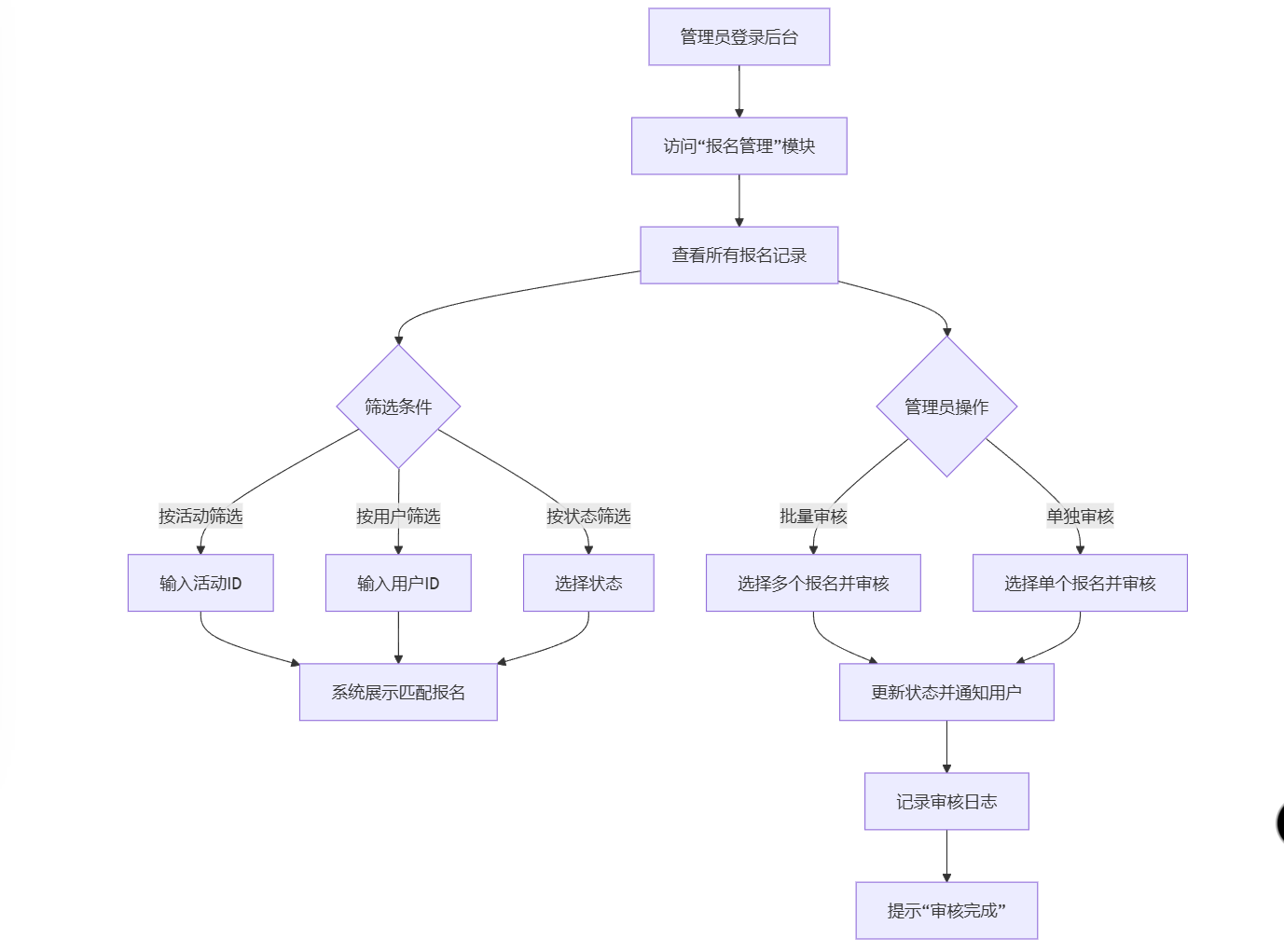
P（连通图数量）= 1

M = 10 - 6 + 2\*1 = 6

圈复杂度：6

## 报名管理模块

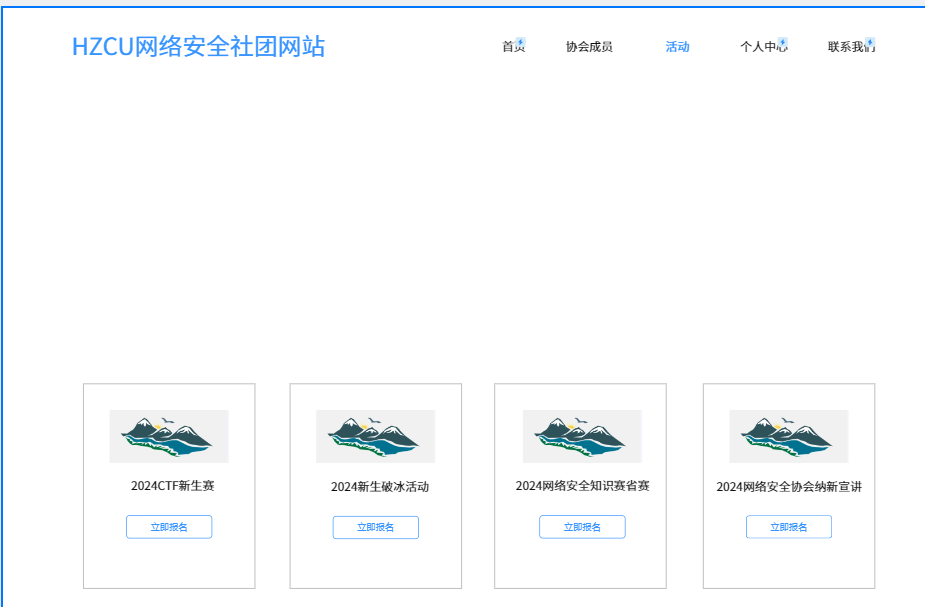
### 功能流程图



### 功能描述

1. 管理员登录后台管理系统。
2. 访问“报名管理”模块。
3. 查看所有用户的报名记录，包括待审核、已通过和已拒绝状态。
4. 管理员可以筛选报名记录，根据活动、用户或状态进行过滤。
5. 管理员可以批量审核报名或单独审核。

### 界面设计



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字段名称** | **长度** | **录入方式** | **是否非空项** | **数据检验** | **默认显示** |
| **活动名称** | 20 | 文本框 | Y | Y | Y |
| **活动信息** | 50 | 文本框 | Y | Y | Y |
| **活动时间** | 20 | 文本框 | Y | Y | Y |
| **地点** | 50 | 文本框 | Y | Y | Y |
| **新闻时间** | 20 | 文本框 | Y | Y | Y |
| **新闻信息** | 50 | 文本框 | Y | Y | Y |
| **管理员编号** | 20 | 文本框 | Y | Y | N |
| **用户编号** | 20 | 文本框 | Y | Y | N |

### 伪代码

# 主过程：处理用户报名

Procedure HandleUserRegistration(userID, eventID, registrationData):

# 步骤 1: 验证报名数据

if not ValidateRegistrationData(registrationData) then

return "Invalid registration data"

end if

# 步骤 2: 检查报名截止日期

event = Database.Query("Events", {"eventID": eventID})

if event is None then

return "Event not found"

end if

if CurrentDate() > event.registrationDeadline then

return "Registration deadline has passed"

end if

# 步骤 3: 检查用户是否已报名

existingRegistration = Database.Query("Registrations", {"userID": userID, "eventID": eventID})

if existingRegistration is not None then

return "Already registered for this event"

end if

# 步骤 4: 创建报名记录

Database.Insert("Registrations", registration)

# 步骤 5: 发送报名成功通知

SendNotification(userID, "Registration Submitted", "Your registration has been submitted and is pending approval.")

return "Registration submitted successfully"

#验证报名数据

Procedure ValidateRegistrationData(registrationData):

# 示例验证：检查必填字段

requiredFields = ["fullName", "contactNumber", "department"]

for each field in requiredFields do

if registrationData[field] is empty then

return False

end if

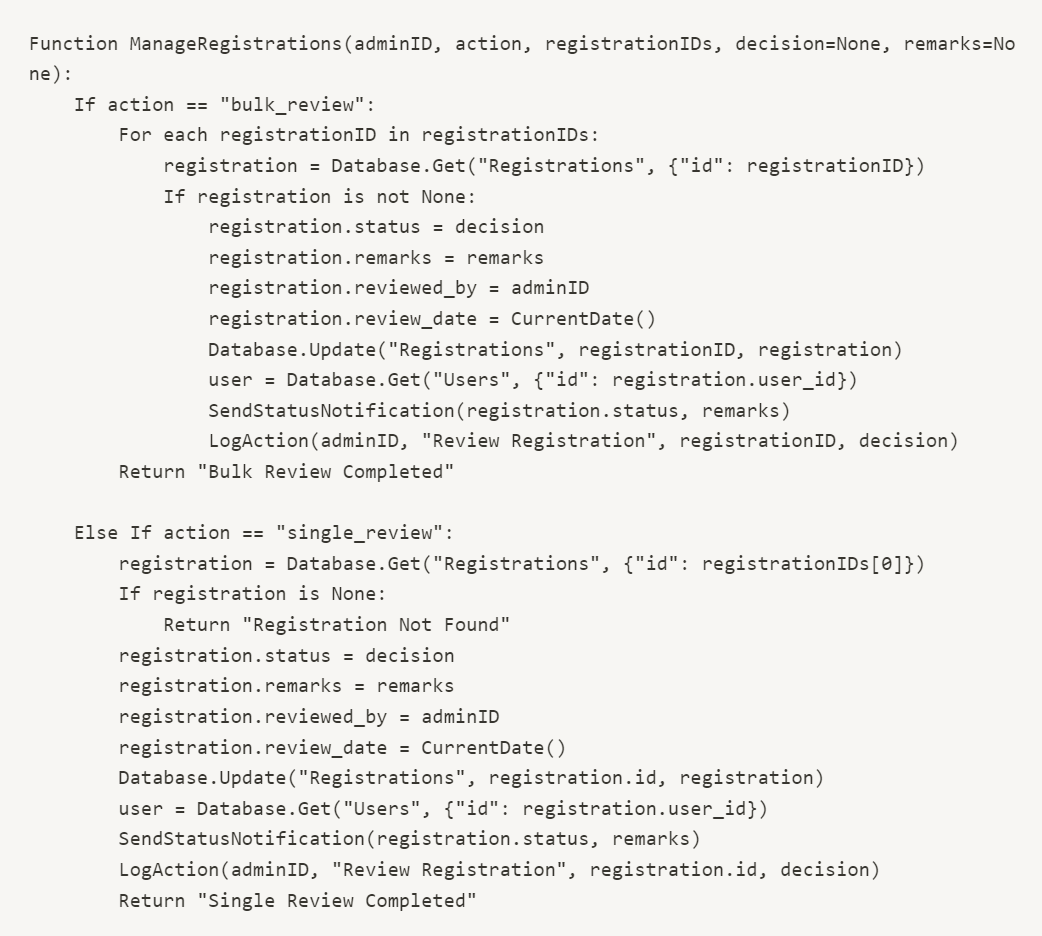
end for

# 其他数据格式验证可以在此添加

return True

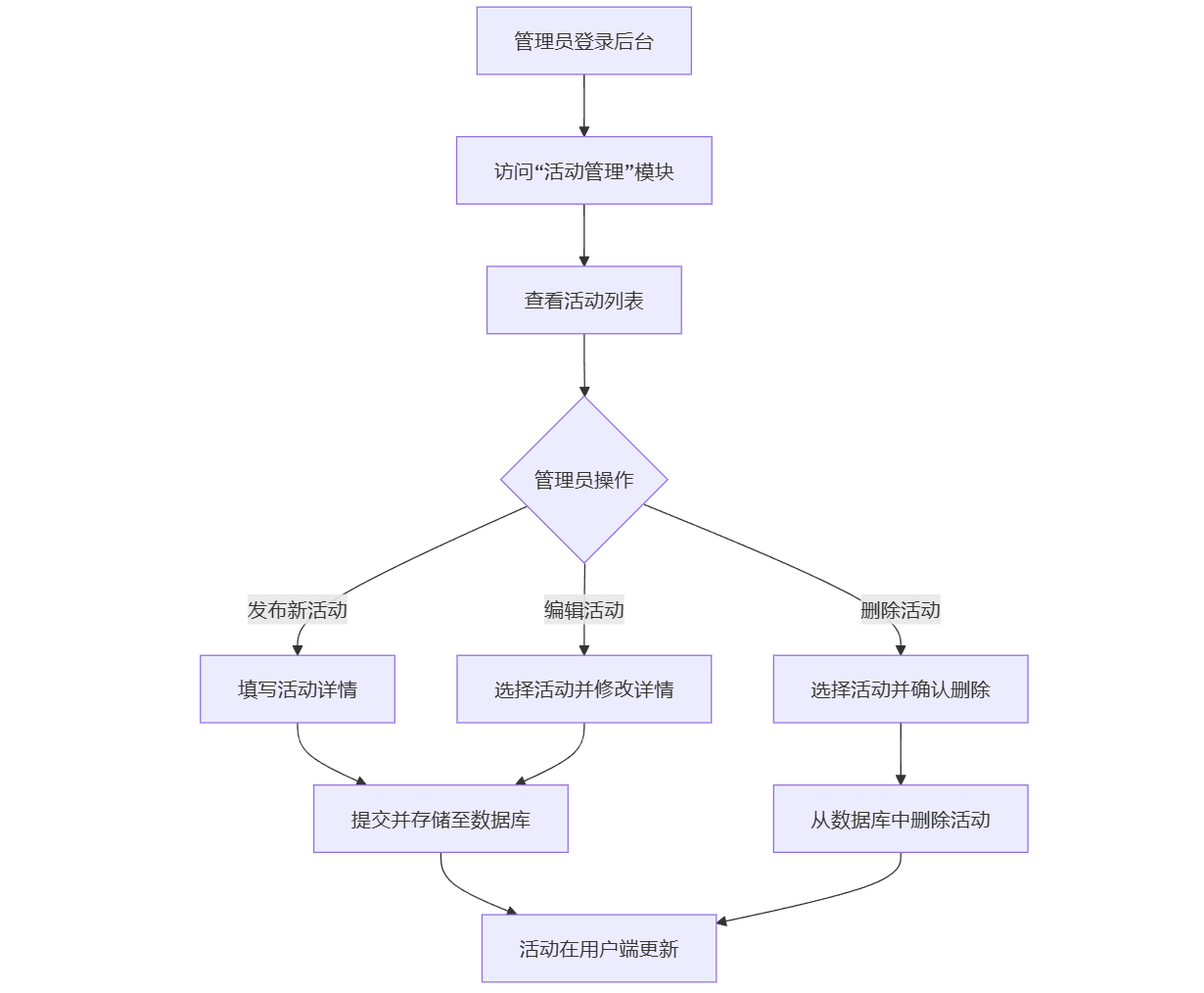
End Procedure

### 关键算法



## 活动管理模块

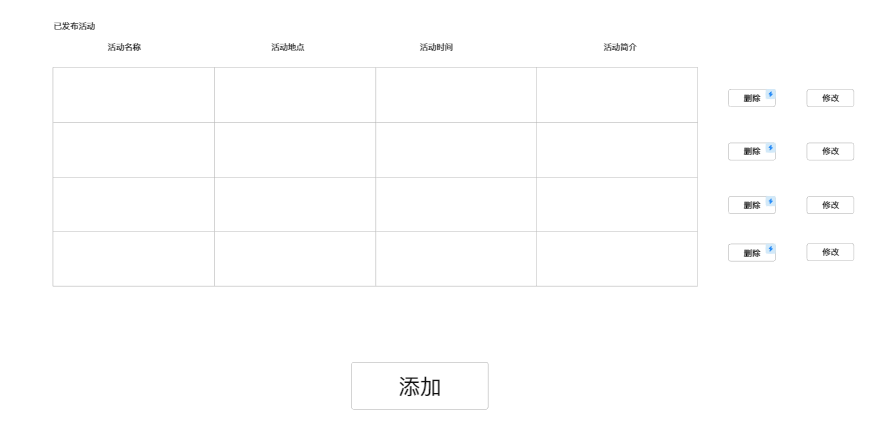
### 功能流程图



### 功能描述

1. 管理员登录后台管理系统。
2. 访问“活动管理”模块。
3. 查看当前所有活动列表。
4. 管理员可以选择发布新活动、编辑现有活动或删除活动。
5. 提交更改后，系统更新数据库，并在用户端展示最新活动信息。

### 界面设计



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **类型** | **长度** | **精度** | **小数位数** | **默认值** | **允许空** | **主键** | **外键** | **说明** |
| **1** | 用户账号 | Variable characters | 10 |  |  |  |  | √ |  |  |
| **2** | 管理员账号 | Variable characters (10) | 10 |  |  |  |  |  | √ |  |
| **3** | 活动编号 | Variable characters (10) | 10 |  |  |  |  |  | √ |  |
| **4** | 活动时间 | Datatime | 8 |  |  |  |  |  | √ |  |
| **5** | 地点 | Variable characters (20) | 30 |  |  |  |  | √ |  |  |

### 伪代码

# 主过程：发布新活动

Procedure PublishEvent(adminID, eventDetails):

# 步骤 1: 验证活动详情

if not ValidateEventDetails(eventDetails) then

return "Invalid event details"

end if

# 步骤 2: 创建活动记录

event = {

"eventID": GenerateUUID(),

"name": eventDetails.name,

"description": eventDetails.description,

"date": eventDetails.date,

"location": eventDetails.location,

"registrationDeadline": eventDetails.registrationDeadline,

"status": "published",

"createdBy": adminID,

"createdAt": CurrentDateTime()

}

Database.Insert("Events", event)

End Procedure

# 主过程：编辑活动

Procedure EditEvent(adminID, eventID, updatedDetails):

# 步骤 1: 获取活动记录

event = Database.Query("Events", {"eventID": eventID})

if event is None then

return "Event not found"

end if

# 步骤 2: 更新活动详情

if ValidateEventDetails(updatedDetails) then

event.name = updatedDetails.name

event.description = updatedDetails.description

event.date = updatedDetails.date

event.location = updatedDetails.location

event.registrationDeadline = updatedDetails.registrationDeadline

Database.Update("Events", eventID, event)

end if

End Procedure

# 主过程：删除活动

Procedure DeleteEvent(adminID, eventID):

# 步骤 1: 获取活动记录

event = Database.Query("Events", {"eventID": eventID})

if event is None then

return "Event not found"

end if

# 步骤 2: 删除活动记录

Database.Delete("Events", {"eventID": eventID})

return "Event deleted successfully"

End Procedure

# 主过程：列出所有活动

Procedure ListEvents(filterCriteria=None):

# 步骤 1: 查询活动

if filterCriteria is not None then

events = Database.QueryAll("Events", filterCriteria)

else

events = Database.QueryAll("Events", {})

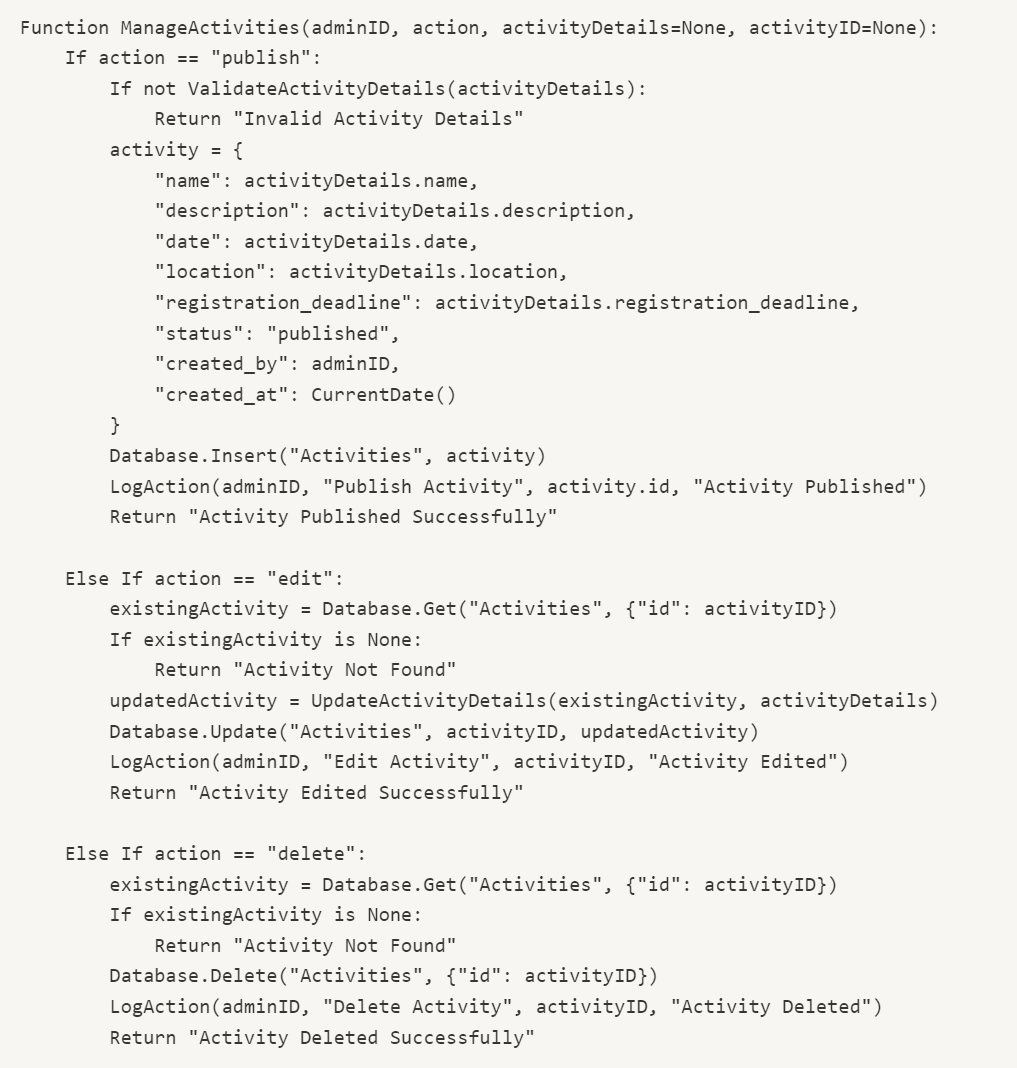
end if

# 步骤 2: 返回活动列表

return events

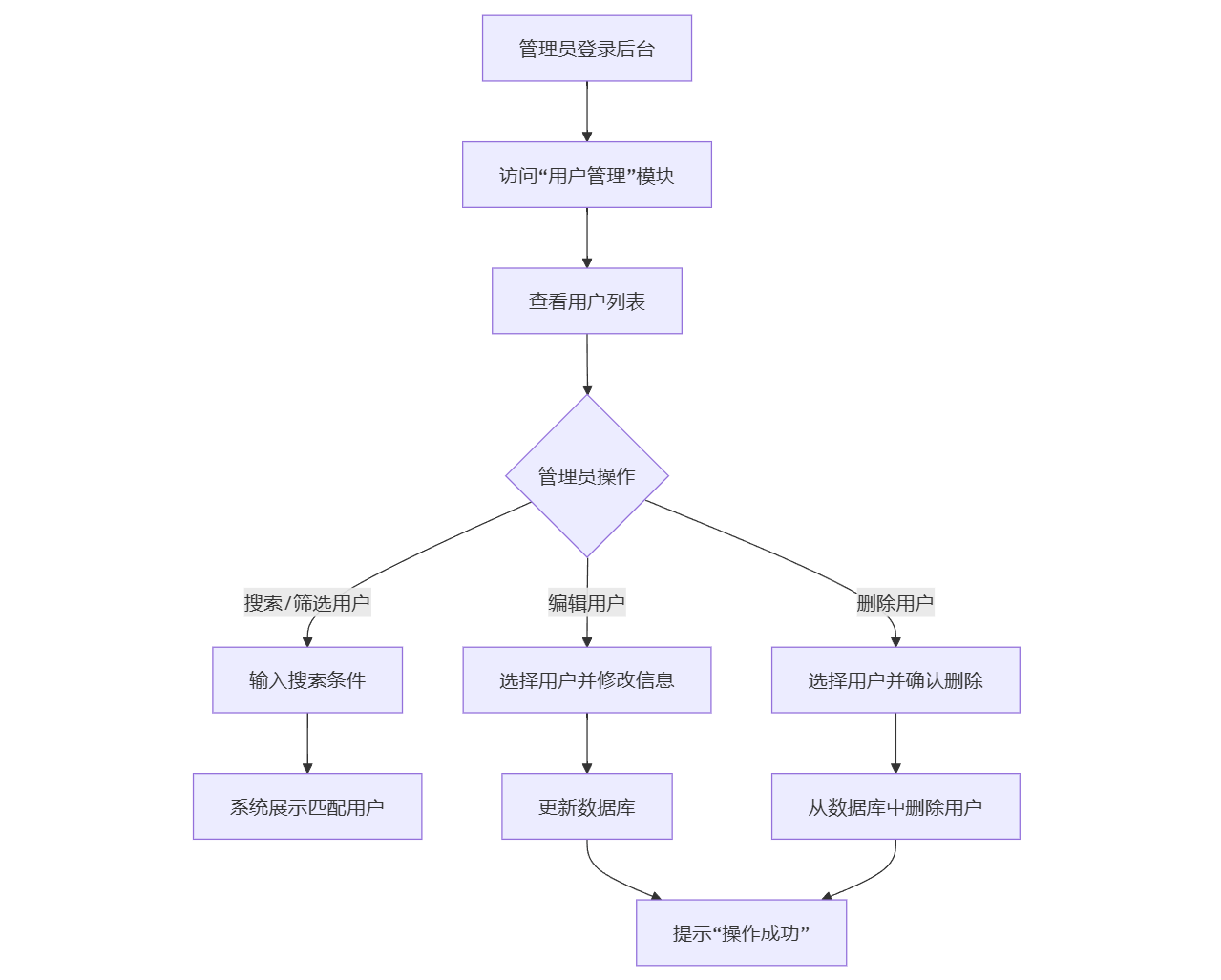
End Procedure

### 关键算法



## 用户管理模块

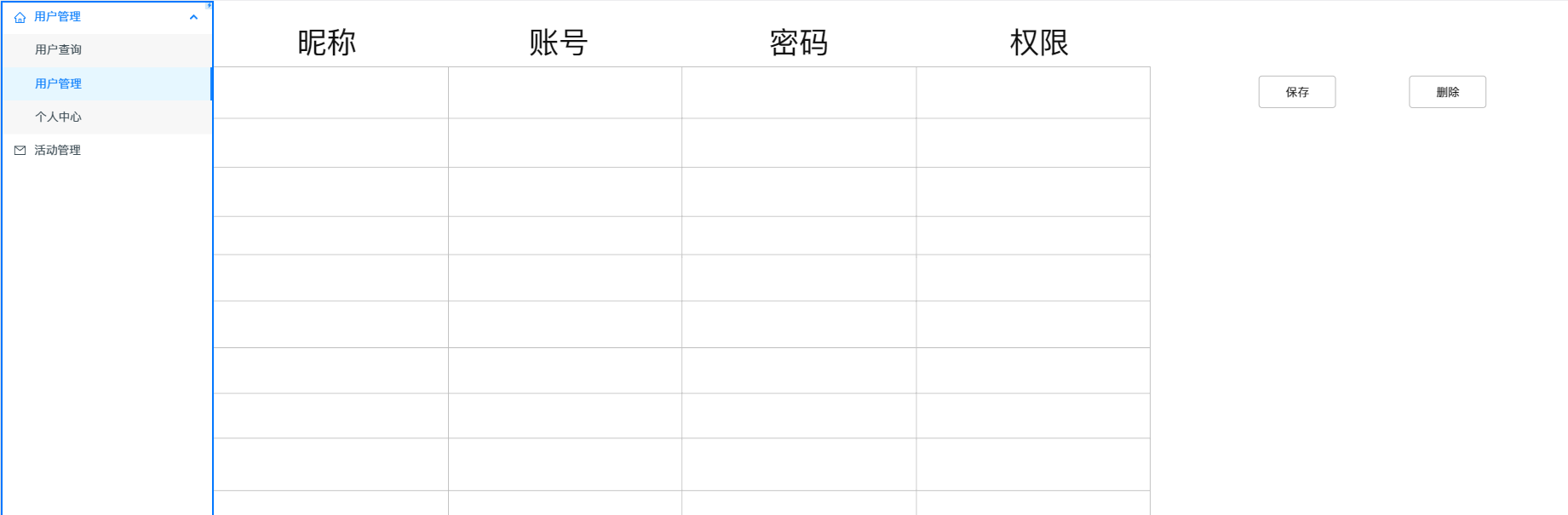
### 功能流程图



### 功能描述

1. 管理员登录后台管理系统。
2. 访问“用户管理”模块。
3. 查看所有注册用户列表，包括普通用户和管理员。
4. 管理员可以搜索、筛选用户，编辑用户信息或删除用户。
5. 管理员可查看用户的详细信息和活动报名记录。

### 界面设计



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **类型** | **长度** | **精度** | **小数位数** | **默认值** | **允许空** | **主键** | **外键** |
| **1** | 用户账号 | Variable characters | 10 |  |  |  |  | √ |  |
| **2** | 管理员账号 | Variable characters (10) | 10 |  |  |  |  | √ |  |
| **3** | 用户名 | Variable characters (20) | 20 |  |  |  |  | √ |  |
| **4** | 用户密码 | Variable characters (20) | 20 |  |  |  |  | √ |  |

### 伪代码

# 主过程：搜索用户

Procedure SearchUser(adminID, searchCriteria):

# 步骤 1: 验证管理员权限

if not IsAdmin(adminID) then

return "Permission Denied"

end if

# 步骤 2: 执行搜索

users = Database.QueryAll("Users", searchCriteria)

# 步骤 3: 返回搜索结果

return users

End Procedure

# 主过程：编辑用户信息

Procedure EditUser(adminID, userID, updatedInfo):

# 步骤 1: 验证管理员权限

if not IsAdmin(adminID) then

return "Permission Denied"

end if

# 步骤 2: 获取用户记录

user = Database.Query("Users", {"userID": userID})

if user is None then

return "User not found"

end if

# 步骤 3: 更新用户信息

for each key, value in updatedInfo do

user[key] = value

end for

# 步骤 4: 保存更新

Database.Update("Users", userID, user)

# 步骤 5: 记录操作日志

LogAction(adminID, "Edit User", userID, "User information updated")

# 步骤 6: 返回成功消息

return "User updated successfully"

End Procedure

# 主过程：删除用户

Procedure DeleteUser(adminID, userID):

# 步骤 1: 验证管理员权限

if not IsAdmin(adminID) then

return "Permission Denied"

end if

# 步骤 2: 获取用户记录

user = Database.Query("Users", {"userID": userID})

if user is None then

return "User not found"

end if

# 步骤 3: 删除用户记录

Database.Delete("Users", {"userID": userID})

# 步骤 4: 记录操作日志

LogAction(adminID, "Delete User", userID, "User deleted")

# 步骤 5: 返回成功消息

return "User deleted successfully"

End Procedure

# 主过程：列出所有用户

Procedure ListAllUsers(adminID):

# 步骤 1: 验证管理员权限

if not IsAdmin(adminID) then

return "Permission Denied"

end if

# 步骤 2: 获取所有用户

users = Database.QueryAll("Users", {})

# 步骤 3: 返回用户列表

return users

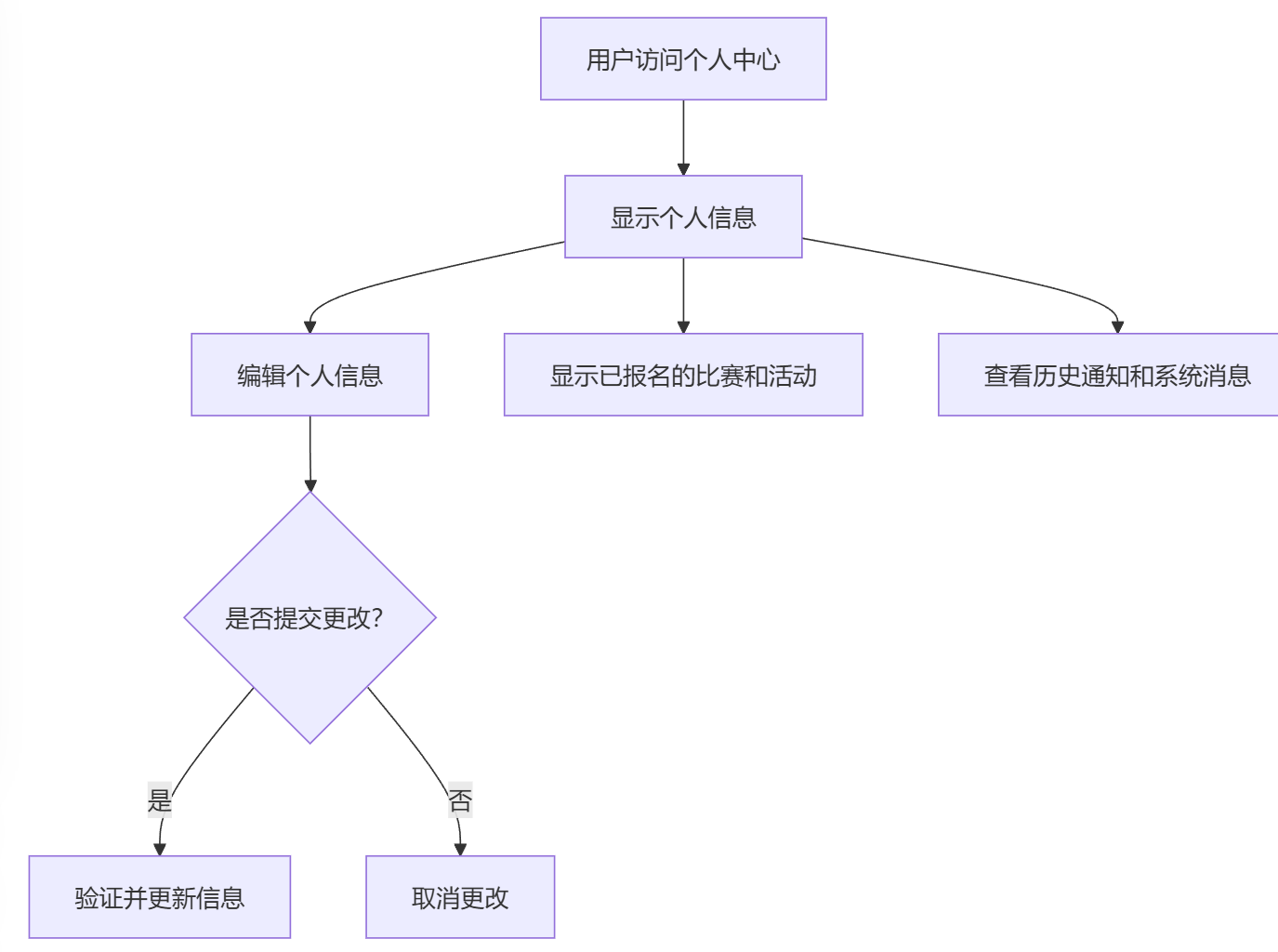
End Procedure

### 关键算法



## 个人中心

### 功能流程图



### 功能描述

1. 用户登录后，访问个人中心页面。

2. 系统显示用户的个人信息、已报名的比赛和活动记录。

3. 用户可以编辑个人信息（如更改密码、更新联系方式）。

4. 用户可以查看历史通知和系统消息。

### 界面设计



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **类型** | **长度** | **精度** | **小数位数** | **默认值** | **允许空** | **主键** | **外键** |
| **1** | 用户账号 | Variable characters | 10 |  |  |  |  | √ |  |
| **2** | 管理员账号 | Variable characters (10) | 10 |  |  |  |  | √ |  |
| **3** | 用户名 | Variable characters (20) | 20 |  |  |  |  |  | √ |
| **4** | 用户密码 | Variable characters (20) | 20 |  |  |  |  | √ |  |
| **5** | 活动编号 | Variable characters (10) | 10 |  |  |  |  |  | √ |  |
| **6** | 活动时间 | Datatime | 8 |  |  |  |  |  | √ |  |
| **7** | 地点 | Variable characters (20) | 30 |  |  |  |  |  | √ |  |

### 伪代码

# 主过程：查看个人信息

Procedure ViewPersonalInfo(userID):

# 步骤 1: 获取用户信息

user = Database.Query("Users", {"userID": userID})

# 步骤 2: 检查用户是否存在

if (user == None) then

return "用户不存在"

end if

# 步骤 3: 返回用户信息

return user

End Procedure

# 主过程：编辑个人信息

Procedure EditPersonalInfo(userID, updatedInfo):

# 步骤 1: 获取用户记录

user = Database.Query("Users", {"userID": userID})

# 步骤 2: 检查用户是否存在

if (user == None) then

return "用户不存在"

end if

# 步骤 3: 更新用户信息

for each key, value in updatedInfo do

user[key] = value

end for

# 步骤 5: 保存更新

Database.Update("Users", userID, user)

# 步骤 6: 返回成功消息

return "个人信息更新成功"

End Procedure

# 主过程：查看报名状态

Procedure ViewRegistrationStatus(userID):

# 步骤 1: 获取用户的所有报名记录

registrations = Database.QueryAll("Registrations", {"userID": userID})

# 步骤 2: 检查是否有报名记录

if (registrations.isEmpty()) then

return "没有找到报名记录"

end if

# 步骤 3: 返回报名状态

return registrations

End Procedure

# 主过程：查看通知

Procedure ViewNotifications(userID):

# 步骤 1: 获取用户的所有通知

notifications = Database.QueryAll("Notifications", {"userID": userID}, order\_by="sentAt DESC")

# 步骤 2: 返回通知列表

return notifications

End Procedure

# 步骤 3: 更新通知状态

notification.isRead = True

Database.Update("Notifications", notificationID, notification)

# 步骤 4: 返回成功消息

return "通知已标记为已读"

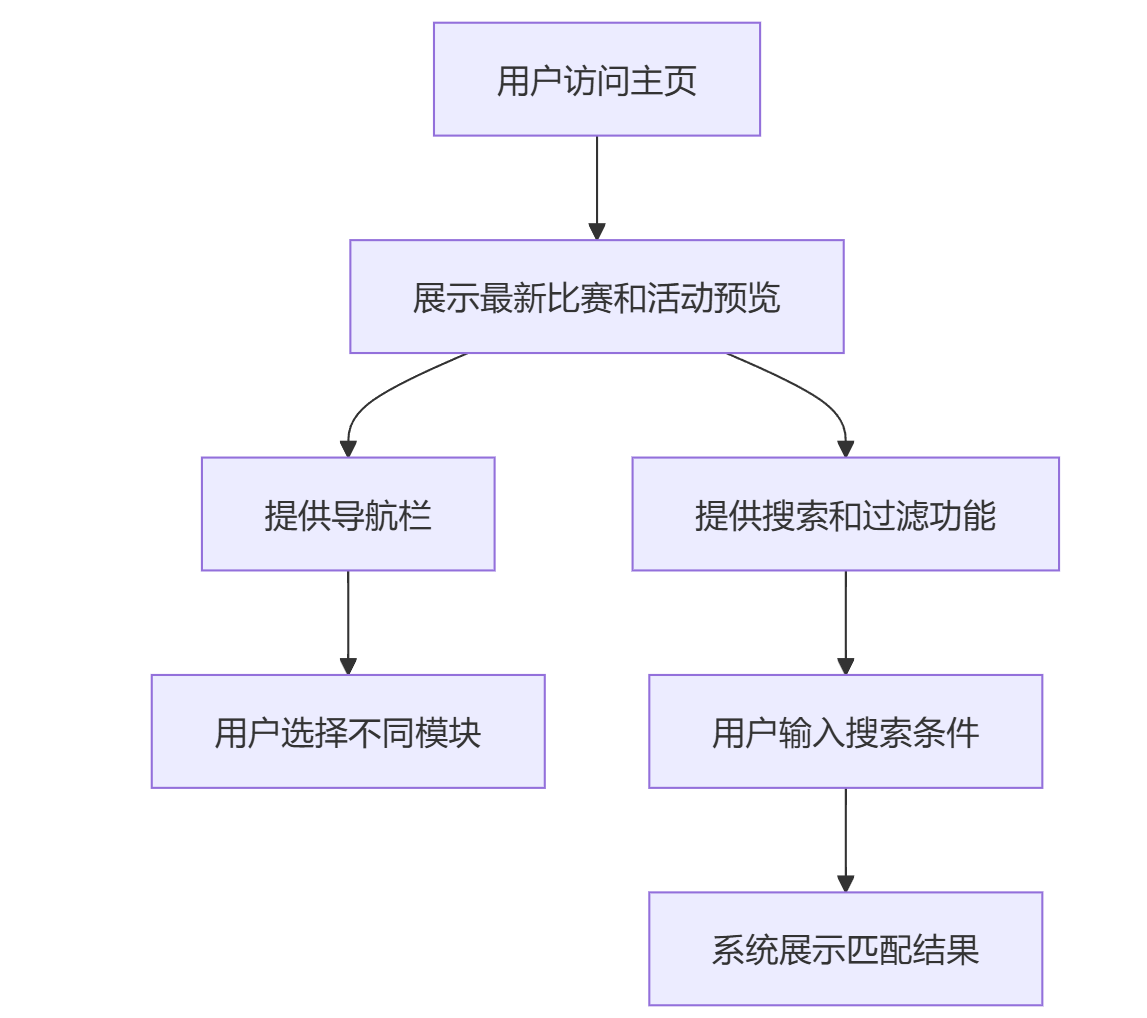
End Procedure

### 关键算法



## 浏览主页模块

### 功能流程图



### 功能描述

1. 用户访问系统主页。
2. 系统展示最新的比赛和活动预览。
3. 用户可以通过导航栏访问不同的模块（如活动信息、个人中心）。
4. 系统提供搜索和过滤功能，帮助用户快速找到感兴趣的比赛或活动。

### 界面设计



### 伪代码

# 主过程：浏览主页

Procedure BrowseHomepage(userID):

# 步骤 1: 检查缓存是否存在

cacheKey = "homepage\_data"

cachedData = Cache.Get(cacheKey)

if (cachedData != None) then

return cachedData

end if

# 步骤 2: 查询最新活动

latestEvents = Database.QueryAll("Events", {"status": "published"}, order\_by="date DESC", limit=5)

# 步骤 3: 返回主页数据

return homepageData

End Procedure

End Procedure

### 关键算法

