攻略收藏模块讲解文档

一、数据库与实体类设计

1.1 核心实体表设计

1.1.1 Collection实体表

属性名	属性含义	是否为空	备注
id	收藏ID	否	自增主键
userId	用户ID	否	外键关联User表
guideId	攻略ID	否	外键关联TravelGuide表
createTime	创建时间	是	自动生成

1.2 核心属性说明

- **外键关联**：userId和guideId分别与用户表和攻略表建立外键关系，保证数据一致性
- **非持久化字段**：实体类中包含多个非持久化字段，用于前端展示关联的攻略和用户信息
  - guideTitle：攻略标题
  - guideCoverImage：攻略封面图片
  - guideViews：攻略浏览量
  - username：用户名
  - userNickname：用户昵称
  - userAvatar：用户头像
- **索引优化**：为userId和guideId创建了索引，提高查询效率
- **时间记录**：自动记录收藏创建时间，便于按时间排序展示

二、各功能详细讲解

2.1 攻略收藏功能

2.1.1 功能概述

攻略收藏功能允许登录用户将感兴趣的旅游攻略添加到个人收藏列表中，方便后续查看和管理，是用户个性化体验的重要组成部分。

2.1.2 详细讲解功能实现流程

用户在攻略详情页面看到感兴趣的攻略时，可以点击页面上方的"收藏"按钮进行收藏操作。前端首先通过checkIsCollected方法检查该攻略是否已被当前用户收藏，根据结果显示不同的按钮状态（已收藏/未收藏）。当用户点击收藏按钮时，前端调用handleCollectionToggle方法，该方法首先检查用户是否已登录，未登录则弹出提示框引导用户登录。如果用户已登录，则根据当前收藏状态执行不同的操作：如果已收藏，则发送DELETE请求到/collection/cancel接口取消收藏；如果未收藏，则发送POST请求到/collection/add接口添加收藏。后端CollectionController接收请求，从JWT令牌中获取当前用户ID，然后调用collectionService的相应方法处理收藏操作。添加收藏时，后端会验证用户和攻略是否存在以及是否已收藏，通过验证后将收藏记录保存到数据库。取消收藏时，后端会删除对应的收藏记录。操作成功后，前端更新收藏按钮状态，提供即时反馈。

2.1.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/guide/GuideDetail.vue】

```
// 处理收藏/取消收藏
const handleCollectionToggle = async () => {
  if (!userStore.isLoggedIn) {
    ElMessage.warning('请先登录')
    router.push('/login?redirect=' + route.fullPath)
    return
  }

  try {
    if (isCollected.value) {
      // 取消收藏
      await request.delete('/collection/cancel?guideId=' +
guide.value.id, {
        successMsg: '取消收藏成功',
        onSuccess: () => {
          isCollected.value = false
        }
      })
    } else {
      // 添加收藏
      await request.post('/collection/add', { guideId: guide.value.id
}, {
```

```
          successMsg: '收藏成功',
          onSuccess: () => {
            isCollected.value = true
          }
        })
      }
    } catch (error) {
      console.error('收藏操作失败:', error)
    }
  }
```

代码说明：handleCollectionToggle方法负责处理用户的收藏/取消收藏操作。首先检查用户是否已登录，未登录则引导用户前往登录页面。然后根据当前收藏状态发送不同的请求：已收藏状态下发送DELETE请求取消收藏，未收藏状态下发送POST请求添加收藏。请求成功后手动更新前端状态，确保界面及时反映最新的收藏状态。整个过程有完善的错误处理，提升用户体验。

2. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/CollectionService.java】

```
/**
 * 添加收藏
 */
public void addCollection(Collection collection) {
    // 验证用户是否存在
    if (userMapper.selectById(collection.getUserId()) == null) {
        throw new ServiceException("用户不存在");
    }

    // 验证攻略是否存在
    if (travelGuideMapper.selectById(collection.getGuideId()) == null)
{
        throw new ServiceException("攻略不存在");
    }

    // 检查是否已收藏
    LambdaQueryWrapper<Collection> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper.eq(Collection::getUserId, collection.getUserId())
                .eq(Collection::getGuideId, collection.getGuideId());
    if (collectionMapper.selectOne(queryWrapper) != null) {
        throw new ServiceException("已经收藏过该攻略");
    }

    // 添加收藏
    if (collectionMapper.insert(collection) <= 0) {
        throw new ServiceException("收藏失败");
    }
```

```
    }
```

代码说明：addCollection方法负责处理添加攻略收藏的业务逻辑。首先验证用户和攻略是否存在；然后检查用户是否已收藏该攻略，避免重复收藏；最后创建收藏记录并保存到数据库。整个过程包含多项业务校验，确保数据的有效性和一致性，并通过异常机制处理各种错误情况。

2.1.4 功能实现细节

- **登录状态检查**：收藏操作需要用户登录，未登录用户会被引导到登录页面
- **重复收藏检查**：系统会检查用户是否已收藏该攻略，避免重复收藏
- **即时反馈**：操作成功或失败都有明确的提示信息，提升用户体验
- **按钮状态管理**：根据收藏状态显示不同的按钮样式和文字，直观反映当前状态
- **异常处理**：完善的异常处理机制，确保系统稳定性
- **数据校验**：后端对用户和攻略的存在性进行校验，确保数据一致性

####

2.2 攻略收藏列表查看功能

2.2.1 功能概述

攻略收藏列表查看功能允许用户查看自己收藏的所有旅游攻略，支持分页浏览和取消收藏操作，帮助用户管理个人收藏内容。

2.2.2 详细讲解功能实现流程

用户点击个人中心的"我的收藏"菜单项，进入收藏管理页面。页面加载时，前端通过onMounted钩子调用fetchCollections方法，该方法向后端发送GET请求到/collection/page接口，传递用户ID、当前页码和每页数量参数。后端CollectionController的getCollectionsByPage方法接收请求，调用collectionService.getCollectionsByPage或getCurrentUserCollections方法查询用户的收藏记录。该方法首先验证用户身份，然后查询数据库获取收藏记录，并通过fillGuideInfo方法填充关联的攻略信息。查询结果以分页形式返回给前端，前端接收数据后，通过el-card组件将每个收藏的攻略以卡片形式展示，包括攻略封面、标题、浏览量等信息。用户可以点击攻略卡片查看详情，也可以点击"取消收藏"按钮取消收藏。点击"取消收藏"按钮时，前端显示确认对话框，用户确认后调用handleCancelCollection方法，向后端发送DELETE请求到/collection/cancel接口。后端处理取消收藏逻辑，操作成功后前端刷新收藏列表。

2.2.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/collection/CollectionList.vue】

```
const fetchCollections = async () => {
```

```
  loading.value = true
  try {
    await request.get('/collection/page', {
      currentPage: currentPage.value,
      size: pageSize.value,
      userId: userStore.userInfo.id
    }, {
      showDefaultMsg: false,
      onSuccess: (res) => {
        tableData.value = res.records||[]
        total.value = res.total||0
      }
    })
  } catch (error) {
    console.error('获取收藏列表失败:', error)
  } finally {
    loading.value = false
  }
}

const handleCancelCollection = (item) => {
  ElMessageBox.confirm('确定要取消收藏这个攻略吗?', '提示', {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
  }).then(() => {
    request.delete('/collection/cancel?guideId=' + item.guideId, {
      successMsg: '取消收藏成功',
      onSuccess: () => {
        // 刷新列表
        fetchCollections()
      }
    })
  }).catch(() => {})
}
```

代码说明：fetchCollections方法负责获取用户收藏的攻略列表，通过request.get发送请求到后端接口，请求成功后将返回的数据保存到tableData响应式变量中。handleCancelCollection方法处理取消收藏操作，首先显示确认对话框，用户确认后发送DELETE请求到后端接口，操作成功后刷新列表。这两个方法共同实现了收藏列表的查看和管理功能。

2. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/CollectionService.java】

```
/**
 * 分页查询用户的收藏列表
 */
```

```
public Page<Collection> getCollectionsByPage(Long userId, Integer
currentPage, Integer size) {
    // 检查用户是否存在
    if (userMapper.selectById(userId) == null) {
        throw new ServiceException("用户不存在");
    }

    // 查询用户收藏
    LambdaQueryWrapper<Collection> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper.eq(Collection::getUserId, userId)
                .orderByDesc(Collection::getCreateTime);

    Page<Collection> page = collectionMapper.selectPage(new Page<>
(currentPage, size), queryWrapper);

    // 填充攻略信息
    fillGuideInfo(page.getRecords());

    return page;
}

/**
 * 填充攻略信息
 */
private void fillGuideInfo(List<Collection> collections) {
    if (collections.isEmpty()) return;

    List<Long> guideIds = collections.stream()
                        .map(Collection::getGuideId)
                        .distinct()
                        .collect(Collectors.toList());

    if (!guideIds.isEmpty()) {
        Map<Long, TravelGuide> guideMap =
travelGuideMapper.selectBatchIds(guideIds)
                                        .stream()

.collect(Collectors.toMap(TravelGuide::getId, guide -> guide));

        for (Collection collection : collections) {
            TravelGuide guide = guideMap.get(collection.getGuideId());
            if (guide != null) {
                collection.setGuideTitle(guide.getTitle());
                collection.setGuideCoverImage(guide.getCoverImage());
                collection.setGuideViews(guide.getViews());
            }
        }
    }
}
```

代码说明：getCollectionsByPage方法负责查询用户收藏的攻略列表，支持分页查询。首先验证用户是否存在，然后查询数据库获取收藏记录，并按收藏时间降序排序。查询完成后，调用fillGuideInfo方法填充关联的攻略信息，该方法通过批量查询优化了性能，避免了N+1查询问题。这种设计既满足了功能需求，又保证了查询效率。

2.2.4 功能实现细节

- **分页查询**：支持分页查询收藏列表，提高大数据量下的查询效率
- **时间排序**：按收藏时间降序排序，最新收藏的攻略显示在前面
- **数据关联**：自动填充关联的攻略信息，避免前端多次请求
- **批量查询优化**：使用批量查询攻略信息，避免N+1查询问题
- **卡片式布局**：采用卡片式布局展示收藏的攻略，视觉效果好
- **空数据处理**：当没有收藏攻略时，显示友好的空状态提示
- **点击跳转**：点击攻略卡片可以直接跳转到攻略详情页面

2.3 攻略收藏状态查询功能

2.3.1 功能概述

攻略收藏状态查询功能允许在攻略详情页面查询当前用户是否已收藏该攻略，根据查询结果显示不同的按钮状态，提供更好的用户体验。

2.3.2 详细讲解功能实现流程

用户访问攻略详情页面时，前端通过fetchGuide方法获取攻略详情数据。当数据加载完成后，如果用户已登录，前端调用checkIsCollected方法，该方法向后端发送GET请求到/collection/isCollected接口，传递攻略ID参数。后端CollectionController的isCollected方法接收请求，从JWT令牌中获取当前用户ID，然后调用collectionService.isCollected方法查询收藏状态。该方法通过查询数据库判断用户是否已收藏该攻略，并返回布尔结果。前端接收结果后，将收藏状态保存到isCollected响应式变量中，然后根据收藏状态显示不同的按钮样式和文字，已收藏状态下显示"已收藏"，未收藏状态下显示"收藏"。这种状态查询方式确保了界面显示与实际收藏状态的一致性。

2.3.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/guide/GuideDetail.vue】

```
// 查询是否已收藏
const checkIsCollected = async (guideId) => {
  if (!userStore.isLoggedIn) return

  try {
    await request.get('/collection/isCollected', { guideId }, {
```

```
        showDefaultMsg: false,
        onSuccess: (data) => {
          isCollected.value = data
        }
      })
    } catch (error) {
      console.error('查询收藏状态出错:', error)
    }
  }
```

代码说明：checkIsCollected方法负责查询用户是否已收藏当前攻略，首先检查用户是否已登录，然后通过request.get发送请求到后端接口，请求成功后将返回的收藏状态保存到isCollected响应式变量中。这种状态查询方式确保了界面显示与实际收藏状态的一致性。

2. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/CollectionService.java】

```
/**
 * 检查用户是否已收藏某攻略
 */
public boolean isCollected(Long userId, Long guideId) {
    LambdaQueryWrapper<Collection> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper.eq(Collection::getUserId, userId)
                .eq(Collection::getGuideId, guideId);
    return collectionMapper.selectCount(queryWrapper) > 0;
}
```

代码说明：isCollected方法负责检查用户是否已收藏某攻略，通过构建查询条件并执行查询，根据查询结果返回布尔值。这种实现方式简单高效，直接通过数据库查询判断收藏状态。

2.3.4 功能实现细节

- **登录状态检查**：只有登录用户才会查询收藏状态，未登录用户直接显示默认状态
- **条件查询**：使用精确条件查询，确保查询结果准确
- **响应式更新**：查询结果保存到响应式变量中，自动触发界面更新
- **错误处理**：完善的错误处理机制，确保系统稳定性
- **视觉反馈**：根据收藏状态显示不同的按钮样式和文字，提供直观的视觉反馈

2.4 攻略收藏管理功能（管理员）

2.4.1 功能概述

攻略收藏管理功能面向系统管理员，允许管理员查看和管理所有用户的收藏记录，支持多条件搜索和删除操作，是后台管理系统的重要组成部分。

2.4.2 详细讲解功能实现流程

管理员登录后台系统，点击"收藏管理"菜单项，进入收藏管理页面。页面加载时，前端通过onMounted钩子调用fetchCollections方法，该方法向后端发送GET请求到/collection/admin/page接口，获取所有用户的收藏记录。管理员可以通过顶部搜索栏输入用户名或攻略标题进行搜索，点击"搜索"按钮触发handleSearch方法重新获取数据。后端CollectionController的getAdminCollectionList方法接收请求，调用collectionService.getCollectionsByAdmin方法查询符合条件的收藏记录。该方法首先根据用户名和攻略标题查询符合条件的用户ID和攻略ID，然后使用这些ID构建查询条件，最后查询数据库获取收藏记录，并通过fillGuideAndUserInfo方法填充关联的攻略和用户信息。查询结果以分页形式返回给前端，前端接收数据后，通过el-table组件将收藏记录以表格形式展示，包括用户信息、攻略信息和收藏时间等。管理员可以点击"删除"按钮删除收藏记录，点击后前端显示确认对话框，管理员确认后调用handleDelete方法，向后端发送DELETE请求到/collection/admin/{id}接口。后端处理删除逻辑，操作成功后前端刷新收藏列表。

2.4.3 关键代码讲解

1. 代码位置：【vue3/src/views/backend/collection/CollectionList.vue】

```
const fetchCollections = async () => {
  loading.value = true
  try {
    await request.get('/collection/admin/page', {
      username: searchForm.username,
      guideTitle: searchForm.guideTitle,
      currentPage: currentPage.value,
      size: pageSize.value
    }, {
      showDefaultMsg: false,
      onSuccess: (res) => {
        tableData.value = res.records||[]
        total.value = res.total||0
      }
    })
  } catch (error) {
    console.error('获取收藏列表失败:', error)
  } finally {
    loading.value = false
  }
}

const handleDelete = (row) => {
```

```
  ElMessageBox.confirm('确定要删除这条收藏记录吗？', '提示', {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
  }).then(() => {
    request.delete(`/collection/admin/${row.id}`,  {
      successMsg: '删除成功',
      onSuccess: () => {
        fetchCollections()
      }
    })
  }).catch(() => {})
}
```

代码说明：fetchCollections方法负责获取所有用户的收藏记录，通过request.get发送请求到后端接口，支持按用户名和攻略标题搜索。handleDelete方法处理删除收藏记录操作，首先显示确认对话框，管理员确认后发送DELETE请求到后端接口，操作成功后刷新列表。这两个方法共同实现了收藏管理功能。

2. 代码位置：
【springboot/src/main/java/org/example/springboot/service/CollectionService.java】

```
/**
 * 管理员查询所有收藏
 */
public Page<Collection> getCollectionsByAdmin(String username, String
guideTitle, Integer currentPage, Integer size) {
    // 首先获取符合条件的用户
    List<Long> userIds = new ArrayList<>();
    if (StringUtils.isNotBlank(username)) {
        LambdaQueryWrapper<User> userQuery = new LambdaQueryWrapper<>
();
        userQuery.like(User::getUsername, username);
        List<User> users = userMapper.selectList(userQuery);
        if (!users.isEmpty()) {
            userIds =
users.stream().map(User::getId).collect(Collectors.toList());
        } else {
            // 如果没找到用户，直接返回空结果
            return new Page<>(currentPage, size);
        }
    }

    // 查询符合条件的攻略
    List<Long> guideIds = new ArrayList<>();
    if (StringUtils.isNotBlank(guideTitle)) {
        LambdaQueryWrapper<TravelGuide> guideQuery = new
LambdaQueryWrapper<>();
```

```
            guideQuery.like(TravelGuide::getTitle, guideTitle);
            List<TravelGuide> guides =
travelGuideMapper.selectList(guideQuery);
            if (!guides.isEmpty()) {
                guideIds =
guides.stream().map(TravelGuide::getId).collect(Collectors.toList());
            } else {
                // 如果没找到攻略，直接返回空结果
                return new Page<>(currentPage, size);
            }
        }

        // 构建查询条件
        LambdaQueryWrapper<Collection> queryWrapper = new
LambdaQueryWrapper<>();
        if (!userIds.isEmpty()) {
            queryWrapper.in(Collection::getUserId, userIds);
        }
        if (!guideIds.isEmpty()) {
            queryWrapper.in(Collection::getGuideId, guideIds);
        }

        queryWrapper.orderByDesc(Collection::getCreateTime);
        Page<Collection> page = collectionMapper.selectPage(new Page<>
(currentPage, size), queryWrapper);

        // 填充攻略和用户信息
        fillGuideAndUserInfo(page.getRecords());

        return page;
    }
```

代码说明：getCollectionsByAdmin方法负责管理员查询所有用户的收藏记录，支持按用户名和攻略标题搜索。该方法首先根据用户名和攻略标题查询符合条件的用户ID和攻略ID，然后使用这些ID构建查询条件，最后查询数据库获取收藏记录，并填充关联的攻略和用户信息。这种实现方式支持灵活的条件搜索，提高了管理效率。

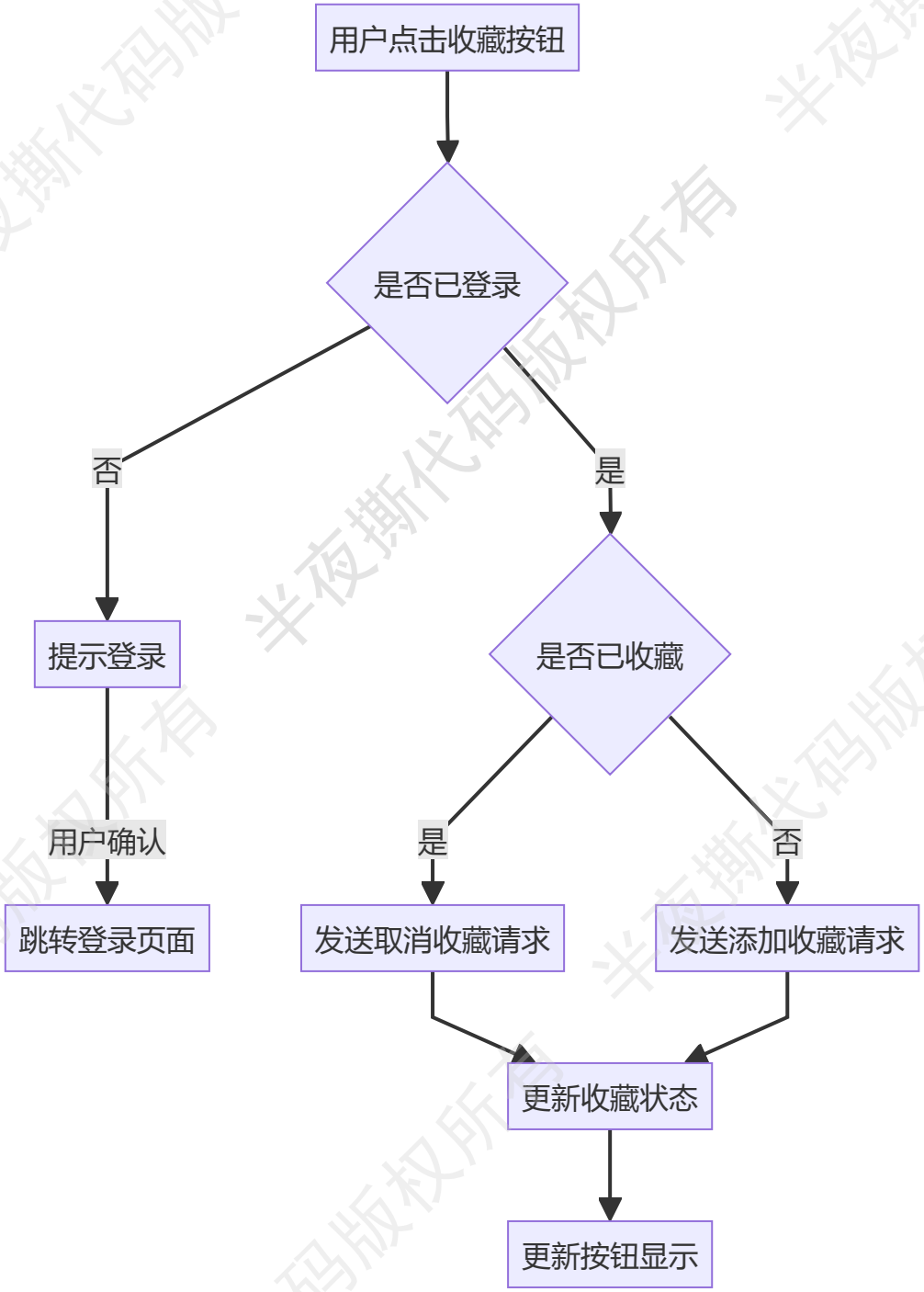
2.4.4 功能实现细节

- **多条件搜索**：支持按用户名和攻略标题搜索，提高查询灵活性
- **分页查询**：支持分页查询收藏列表，提高大数据量下的查询效率
- **时间排序**：按收藏时间降序排序，最新收藏的记录显示在前面
- **数据关联**：自动填充关联的攻略和用户信息，提供完整的数据展示
- **批量查询优化**：使用批量查询攻略和用户信息，避免N+1查询问题
- **权限控制**：只有管理员可以访问收藏管理功能，确保系统安全性
- **操作确认**：删除操作需要确认，避免误操作

三、总结

3.1 攻略收藏功能实现

攻略收藏功能通过前端GuideDetail.vue组件中的handleCollectionToggle方法实现，该方法根据当前收藏状态发送不同的请求：未收藏时发送POST请求添加收藏，已收藏时发送DELETE请求取消收藏。后端CollectionService.addCollection方法处理添加收藏逻辑，包括验证用户和攻略是否存在、避免重复收藏等。整个功能设计注重用户体验，提供了即时反馈和友好的交互，同时通过完善的异常处理确保系统稳定性。



3.2 攻略收藏列表查看功能实现

攻略收藏列表查看功能通过CollectionList.vue组件实现，该组件在加载时调用fetchCollections方法获取用户收藏的攻略列表。后端CollectionService.getCollectionsByPage方法查询用户的收藏记录，并通过fillGuideInfo方法填充关联的攻略信息。前端以卡片形式展示收藏的攻略，支持查看详情和取消收藏操作。取消收藏通过handleCancelCollection方法实现，该方法发送DELETE请求到后端接口，操作成功后刷新列表。整个功能设计注重数据关联和用户体验，通过批量查询优化了性能，避免了N+1查询问题。

3.3 攻略收藏状态查询功能实现

攻略收藏状态查询功能通过GuideDetail.vue组件中的checkIsCollected方法实现，该方法在攻略详情加载完成后调用，向后端发送GET请求查询收藏状态。后端CollectionService.isCollected方法通过查询数据库判断用户是否已收藏该攻略，并返回布尔结果。前端根据查询结果显示不同的按钮样式和文字，提供直观的视觉