周边住宿模块讲解文档

一、数据库与实体类设计

1.1 核心实体表设计

1.1.1 Accommodation实体表

属性名	属性含义	是否为空	备注
id	住宿ID	否	自增主键
name	住宿名称	否	-
type	住宿类型	否	酒店/民宿/客栈/青旅等
address	住宿地址	否	-
scenicId	关联景点ID	是	外键关联ScenicSpot表
description	描述	是	-
contactPhone	联系电话	是	-
priceRange	价格区间	是	如"200-500"
starLevel	评分	是	小数点后1位
imageUrl	图片URL	是	-
features	特色服务	是	-
distance	距景点距离	是	如"500米"
createTime	创建时间	是	自动生成
updateTime	更新时间	是	自动更新

1.1.2 AccommodationReview实体表

属性名	属性含义	是否为空	备注
id	评价ID	否	自增主键
userId	用户ID	否	外键关联User表
accommodationId	住宿ID	否	外键关联Accommodation表
content	评价内容	是	-
rating	评分	是	小数点后1位
createTime	创建时间	是	自动生成

1.2 核心属性说明

- **住宿类型**：系统支持多种住宿类型，包括酒店、民宿、客栈、青旅等，便于用户筛选
- **价格区间**：使用字符串形式存储价格区间，如"200-500"，方便前端直接展示
- **评分机制**：住宿评分由用户评价计算得出，使用BigDecimal类型存储，保留一位小数
- **特色服务**：记录住宿提供的特色服务，如"免费WiFi,24小时前台,室内游泳池"等
- **关联景点**：住宿可以关联到特定景点，便于用户查找景点周边住宿
- **距离表示**：使用字符串形式存储与景点的距离，如"500米"，便于用户理解
- **非持久化字段**：scenicName和reviewCount为非持久化字段，用于前端展示，避免多次查询

二、各功能详细讲解

2.1 住宿列表浏览功能

2.1.1 功能概述

住宿列表浏览功能允许用户查看所有可用的住宿信息，支持多条件筛选，包括关联景点、住宿类型、价格区间和最低评分，提供分页浏览和详细信息展示，是用户选择住宿的入口。

2.1.2 详细讲解功能实现流程

用户访问住宿列表页面时，前端通过onMounted钩子调用三个方法：fetchScenicOptions()获取景点列表用于筛选，fetchAccommodationTypes()获取住宿类型列表，fetchAccommodations()获取住宿列表数据。fetchAccommodations方法向后端发送GET请求到/accommodation/page接口，传递分页参数和筛选条件。后端AccommodationController接收请求并调用

accommodationService.getAccommodationsByPage方法，该方法使用LambdaQueryWrapper构建查询条件，包括景点ID、住宿类型、价格区间和最低评分。查询结果以分页形式返回给前端，同时查询关联的景点信息并填充到结果中。前端接收数据后，通过el-card组件将每个住宿以卡片形式展示，包括名称、类型、图片、地址、评分、价格区间和特色服务等信息。用户可以通过左侧筛选面板输入条件进行筛选，点击"筛选"按钮触发handleFilter()方法重新获取数据；也可以点击"重置"按钮调用resetFilter()方法清空筛选条件。用户点击住宿卡片时，触发goToDetail(id)方法，跳转到相应的住宿详情页面。

2.1.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/accommodation/index.vue】

```
const fetchAccommodations = async () => {
  loading.value = true
  try {
    // 构建查询参数
    const params = {
      currentPage: currentPage.value,
      size: pageSize.value
    }

    // 添加筛选条件
    if (filters.scenicId) params.scenicId = filters.scenicId
    if (filters.type) params.type = filters.type
    if (filters.minPrice) params.minPrice = filters.minPrice
    if (filters.maxPrice) params.maxPrice = filters.maxPrice
    if (filters.minRating > 0) params.minRating = filters.minRating

    // 发送请求
    await request.get('/accommodation/page', params, {
      onSuccess: (res) => {
        accommodationList.value = res.records||[]
        total.value = res.total||0
      }
    })
  } catch (error) {
    console.error('获取住宿列表失败:', error)
  } finally {
    loading.value = false
  }
}
```

代码说明：fetchAccommodations方法负责获取住宿列表数据，首先构建查询参数，包括分页参数和筛选条件，然后通过request.get发送请求到后端接口。请求成功后，将返回的住宿数据保存到accommodationList响应式变量中，并更新总记录数。loading状态用于控制加载动画，提升用户体验。

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

```
public Page<Accommodation> getAccommodationsByPage(Integer scenicId,
String type,
                                                   String minPrice,
String maxPrice,
                                                   String minRating,
                                                   Integer currentPage,
Integer size) {
    LambdaQueryWrapper<Accommodation> queryWrapper = new
LambdaQueryWrapper<>();

    // 添加查询条件
    if (scenicId != null) {
        queryWrapper.eq(Accommodation::getScenicId, scenicId);
    }

    if (StringUtils.isNotBlank(type)) {
        queryWrapper.eq(Accommodation::getType, type);
    }

    // 处理价格区间筛选（这里使用简单字符串比较，实际可能需要更复杂的逻辑）
    if (StringUtils.isNotBlank(minPrice) &&
StringUtils.isNotBlank(maxPrice)) {
        queryWrapper.like(Accommodation::getPriceRange, minPrice + "-"
+ maxPrice);
    }

    if (StringUtils.isNotBlank(minRating)) {
        queryWrapper.ge(Accommodation::getStarLevel,
Double.parseDouble(minRating));
    }

    // 按评分降序排序
    queryWrapper.orderByDesc(Accommodation::getStarLevel);

    Page<Accommodation> page = accommodationMapper.selectPage(new
Page<>(currentPage, size), queryWrapper);

    // 获取关联的景点信息
    List<Long> scenicIds = page.getRecords().stream()
            .map(Accommodation::getScenicId)
            .filter(id -> id != null)
            .collect(Collectors.toList());

    if (!scenicIds.isEmpty()) {
```

```
        List<ScenicSpot> scenicSpots =
scenicSpotMapper.selectBatchIds(scenicIds);
        Map<Long, String> scenicNameMap = scenicSpots.stream()
                .collect(Collectors.toMap(ScenicSpot::getId,
ScenicSpot::getName));

        // 设置关联的景点名称
        page.getRecords().forEach(accommodation -> {
            if (accommodation.getScenicId() != null) {

 accommodation.setScenicName(scenicNameMap.get(accommodation.getScenicI
d()));
            }
        });
    }

    return page;
}
```

代码说明：getAccommodationsByPage方法使用MyBatis-Plus的LambdaQueryWrapper构建查询条件，根据传入的参数动态添加条件。查询结果通过MyBatis-Plus的分页插件进行分页处理。特别注意的是，方法还会查询关联的景点信息，并将景点名称填充到结果中，避免前端需要多次请求获取关联数据。

2.1.4 功能实现细节

- **动态查询条件**：根据用户输入动态构建SQL查询条件，提高查询灵活性
- **景点关联查询**：支持根据景点ID筛选住宿，实现景点与住宿的关联查询
- **评分筛选**：支持按最低评分筛选，帮助用户找到高质量住宿
- **价格区间筛选**：支持按价格区间筛选，满足不同预算需求
- **数据优化**：后端一次性查询关联数据并填充，减少前端请求次数
- **卡片式布局**：采用响应式栅格布局，在不同屏幕尺寸下自动调整卡片数量
- **分页控制**：使用Element Plus的分页组件，支持页码切换和每页显示数量调整

2.2 住宿详情查看功能

2.2.1 功能概述

住宿详情查看功能允许用户查看特定住宿的详细信息，包括基本信息、特色服务、用户评价等，同时展示周边景点和相似住宿推荐，帮助用户做出更好的选择。

2.2.2 详细讲解功能实现流程

用户点击住宿列表中的住宿卡片后，前端通过router.push导航到住宿详情页面（detail.vue）。该页面通过route.params.id获取住宿ID，并在onMounted钩子中调用fetchAccommodationDetail()方法，向后端发送GET请求到/accommodation/{id}接口获取住宿详情。后端AccommodationController的getAccommodationById方法接收请求，调用accommodationService.getAccommodationById方法查询住宿信息，同时查询关联的景点信息和评价数量。前端接收数据后，在页面左侧显示住宿详情，包括名称、类型、评分、图片、地址、联系电话、价格区间、特色服务和描述等信息。同时，页面调用fetchAccommodationReviews()方法获取住宿评价列表，显示在详情下方。页面右侧显示周边景点和相似住宿推荐，这些数据通过watch监听住宿数据加载完成后，调用fetchNearbyScenics()和fetchSimilarAccommodations()方法获取。用户可以点击右侧推荐的景点或住宿，跳转到相应的详情页面。此外，用户可以点击"发表评价"按钮，打开评价对话框，填写评分和评价内容，提交后会更新住宿的评分和评价列表。

2.2.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/accommodation/detail.vue】

```
const fetchAccommodationDetail = async () => {
  loading.value = true
  try {
    await request.get(`/accommodation/${route.params.id}`, {}, {
      onSuccess: (res) => {
        accommodation.value = res
      }
    })
  } catch (error) {
    console.error('获取住宿详情失败:', error)
  } finally {
    loading.value = false
  }
}

// 获取住宿评价列表
const fetchAccommodationReviews = async () => {
  loadingReviews.value = true
  try {
    await request.get('/accommodation/review/page', {
      accommodationId: route.params.id,
      currentPage: reviewPage.value,
      size: reviewPageSize.value
    }, {
      onSuccess: (res) => {
        reviewList.value = res.records||[]
        reviewTotal.value = res.total||0
      }
    })
```

```
    } catch (error) {
      console.error('获取住宿评价失败:', error)
    } finally {
      loadingReviews.value = false
    }
  }
```

代码说明：fetchAccommodationDetail方法负责获取住宿详情数据，通过request.get发送请求到后端接口。fetchAccommodationReviews方法负责获取住宿评价列表，支持分页查询。这两个方法在页面加载时被调用，确保用户可以看到完整的住宿信息和评价。

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

```
public Accommodation getAccommodationById(Integer id) {
    Accommodation accommodation = accommodationMapper.selectById(id);
    if (accommodation != null && accommodation.getScenicId() != null) {
        // 查询关联景点信息
        ScenicSpot scenicSpot =
scenicSpotMapper.selectById(accommodation.getScenicId());
        if (scenicSpot != null) {
            accommodation.setScenicName(scenicSpot.getName());
        }

        // 查询评价数量

 LambdaQueryWrapper<org.example.springboot.entity.AccommodationReview>
reviewWrapper = new LambdaQueryWrapper<>();

 reviewWrapper.eq(org.example.springboot.entity.AccommodationReview::ge
tAccommodationId, id);
        Long reviewCount = reviewMapper.selectCount(reviewWrapper);
        accommodation.setReviewCount(reviewCount);
    }
    return accommodation;
}
```

代码说明：getAccommodationById方法首先查询住宿基本信息，然后查询关联的景点信息和评价数量，填充到结果中。这种方式避免了前端需要多次请求获取关联数据，提高了用户体验。

2.2.4 功能实现细节

- **详细信息展示**：展示住宿的所有相关信息，包括基本信息、特色服务、描述等
- **图片预览**：支持点击图片查看大图，提升用户体验
- **评价列表**：分页展示用户评价，包括评分、内容、评价时间和用户信息

- **周边景点推荐**：展示周边景点，帮助用户规划行程
- **相似住宿推荐**：展示同类型的其他住宿，提供更多选择
- **数据关联**：后端一次性查询关联数据并填充，减少前端请求次数
- **响应式布局**：适应不同屏幕尺寸，提供良好的移动端体验

2.3 住宿评价功能

2.3.1 功能概述

住宿评价功能允许用户对住过的住宿进行评价和打分，分享自己的住宿体验，帮助其他用户做出选择，同时提供评价管理功能，包括删除自己的评价。

2.3.2 详细讲解功能实现流程

用户在住宿详情页面点击"发表评价"按钮，前端显示评价对话框，包含评分和评价内容输入框。用户填写评价信息后，点击"提交"按钮，前端首先检查用户是否已登录，未登录则跳转到登录页面。如果已登录，则调用submitReview()方法，向后端发送POST请求到/accommodation/review接口，传递评价数据。后端AccommodationReviewController的addReview方法接收请求，调用reviewService.addReview方法处理评价添加逻辑。该方法首先获取当前登录用户信息，设置评价的用户ID和创建时间，然后保存评价数据，最后调用updateAccommodationRating方法更新住宿的平均评分。评价成功后，前端关闭评价对话框，清空表单，并刷新评价列表和住宿详情，以显示最新的评价和评分。用户也可以删除自己发布的评价，点击评价右侧的"删除"按钮，前端调用handleDeleteReview()方法，向后端发送DELETE请求到/accommodation/review/{id}接口。后端处理删除逻辑，包括验证用户权限（只有评价发布者或管理员可以删除），删除评价数据，并更新住宿的平均评分。

2.3.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/accommodation/detail.vue】

```
const submitReview = async () => {
  // 检查是否登录
  if (!userStore.isLoggedIn) {
    ElMessage.warning('请先登录再发表评价')
    router.push('/login')
    return
  }

  submittingReview.value = true
  try {
    await request.post('/accommodation/review', reviewForm, {
      successMsg: '评价提交成功',
      onSuccess: () => {
        showReviewDialog.value = false
        reviewForm.rating = 5
```

```
            reviewForm.content = ''
            // 刷新评价列表
            reviewPage.value = 1
            fetchAccommodationReviews()
            // 刷新住宿详情以更新评分
            fetchAccommodationDetail()
          }
        })
      } catch (error) {
        console.error('提交评价失败:', error)
      } finally {
        submittingReview.value = false
      }
    }
```

代码说明：submitReview方法首先检查用户是否已登录，然后通过request.post发送评价数据到后端。评价成功后，关闭对话框，清空表单，并刷新评价列表和住宿详情，确保用户可以看到最新的评价和评分。

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

```
public boolean addReview(AccommodationReview review) {
    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        return false;
    }

    review.setUserId((long) currentUser.getId().intValue());
    review.setCreateTime(LocalDateTime.now());

    boolean result = reviewMapper.insert(review) > 0;

    if (result) {
        // 更新住宿的平均评分
        updateAccommodationRating(review.getAccommodationId());
    }

    return result;
}

private void updateAccommodationRating(Integer accommodationId) {
    LambdaQueryWrapper<AccommodationReview> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper.eq(AccommodationReview::getAccommodationId,
accommodationId);
```

```
        // 查询该住宿的所有评价
        List<AccommodationReview> reviews =
reviewMapper.selectList(queryWrapper);

        if (!reviews.isEmpty()) {
            // 计算平均评分
            BigDecimal totalRating = BigDecimal.ZERO;
            for (AccommodationReview review : reviews) {
                totalRating = totalRating.add(review.getRating());
            }

            BigDecimal averageRating = totalRating.divide(new
BigDecimal(reviews.size()), 1, RoundingMode.HALF_UP);

            // 更新住宿的评分
            org.example.springboot.entity.Accommodation accommodation =
accommodationMapper.selectById(accommodationId);
            if (accommodation != null) {
                accommodation.setStarLevel(averageRating);
                accommodationMapper.updateById(accommodation);
            }
        }
    }
```

代码说明：addReview方法负责添加评价数据，首先获取当前登录用户信息，设置评价的用户ID和创建时间，然后保存评价数据。评价保存成功后，调用updateAccommodationRating方法更新住宿的平均评分，该方法查询住宿的所有评价，计算平均分，并更新住宿的评分字段。

2.3.4 功能实现细节

- **用户身份验证**：评价前检查用户是否已登录，未登录则引导用户登录
- **评分机制**：使用星级评分组件，支持1-5星评分
- **评价内容验证**：评价内容需要符合长度要求，前端进行表单验证
- **权限控制**：只有评价发布者或管理员可以删除评价
- **评分计算**：添加或删除评价后，自动重新计算住宿的平均评分
- **用户信息展示**：评价列表展示用户昵称和头像，增强社交属性
- **时间格式化**：评价时间进行格式化展示，提升用户体验

2.4 住宿管理功能

2.4.1 功能概述

住宿管理功能面向系统管理员，提供住宿信息的增删改查功能，支持多条件搜索，是管理员维护住宿数据的重要工具。

2.4.2 详细讲解功能实现流程

管理员登录后台系统，点击"住宿管理"菜单项，进入住宿管理页面。页面加载时，前端调用fetchScenicOptions()获取景点列表，fetchAccommodationTypes()获取住宿类型列表，fetchAccommodations()获取住宿列表数据。管理员可以通过顶部搜索栏输入条件进行搜索，点击"查询"按钮触发handleSearch()方法重新获取数据。管理员点击"添加住宿"按钮，前端显示添加对话框，填写住宿信息后，点击"确定"按钮，前端调用submitForm()方法，向后端发送POST请求到/accommodation接口，添加住宿数据。管理员点击住宿列表中的"编辑"按钮，前端显示编辑对话框，加载住宿信息，修改后点击"确定"按钮，前端调用submitForm()方法，向后端发送PUT请求到/accommodation/{id}接口，更新住宿数据。管理员点击"删除"按钮，前端显示确认对话框，确认后调用handleDelete()方法，向后端发送DELETE请求到/accommodation/{id}接口，删除住宿数据。后端AccommodationController接收这些请求，调用accommodationService的相应方法处理数据操作，并返回结果。

2.4.3 关键代码讲解

1. 代码位置：【vue3/src/views/backend/accommodation/index.vue】

```
const submitForm = async () => {
  if (!formRef.value) return

  await formRef.value.validate(async (valid) => {
    if (valid) {
      formLoading.value = true
      try {
        if (isEdit.value) {
          // 更新
          await request.put(`/accommodation/${form.id}`, form, {
            successMsg: '更新成功',
            onSuccess: () => {
              dialogVisible.value = false
              fetchAccommodations()
            }
          })
        } else {
          // 新增
          await request.post('/accommodation', form, {
            successMsg: '添加成功',
            onSuccess: () => {
              dialogVisible.value = false
              fetchAccommodations()
```

```
            }
          })
        }
      } catch (error) {
        console.error(`${isEdit.value ? '更新' : '添加'}住宿失败:`, error)
      } finally {
        formLoading.value = false
      }
    }
  })
}
```

代码说明：submitForm方法负责处理住宿信息的添加和更新，首先进行表单验证，验证通过后根据isEdit标志决定发送PUT请求（更新）或POST请求（添加）。操作成功后，关闭对话框并刷新住宿列表。

2. 代码位置：
   【springboot/src/main/java/org/example/springboot/controller/AccommodationController.java】

```
@Operation(summary = "添加住宿信息")
@PostMapping
public Result<?> addAccommodation(@RequestBody Accommodation
accommodation) {
    try {
        LOGGER.info("添加住宿信息: {}", accommodation);

        boolean result =
accommodationService.addAccommodation(accommodation);

        if (result) {
            return Result.success(accommodation);
        } else {
            return Result.error("添加住宿信息失败");
        }
    } catch (Exception e) {
        LOGGER.error("添加住宿信息失败", e);
        return Result.error("添加住宿信息失败: " + e.getMessage());
    }
}

@Operation(summary = "更新住宿信息")
@PutMapping("/{id}")
public Result<?> updateAccommodation(@PathVariable Long id,
@RequestBody Accommodation accommodation) {
    try {
        LOGGER.info("更新住宿信息，id={}，数据: {}", id, accommodation);

        accommodation.setId(id);
```

```
            boolean result =
accommodationService.updateAccommodation(accommodation);

            if (result) {
                return Result.success(accommodation);
            } else {
                return Result.error("更新住宿信息失败");
            }
        } catch (Exception e) {
            LOGGER.error("更新住宿信息失败", e);
            return Result.error("更新住宿信息失败: " + e.getMessage());
        }
    }
```

代码说明：AccommodationController提供了完整的RESTful接口，包括添加、更新、删除和查询住宿信息。每个接口都有详细的日志记录和异常处理，确保系统稳定性和可维护性。

2.4.4 功能实现细节

- **表单验证**：使用Element Plus的表单验证功能，确保数据有效性
- **多条件搜索**：支持按住宿名称、类型和关联景点搜索
- **数据关联**：添加和编辑住宿时可以关联到特定景点
- **图片上传**：支持上传住宿图片，提升展示效果
- **权限控制**：只有管理员可以访问住宿管理功能
- **操作反馈**：操作成功或失败都有明确的提示信息
- **批量操作**：支持批量删除住宿信息，提高管理效率

三、总结

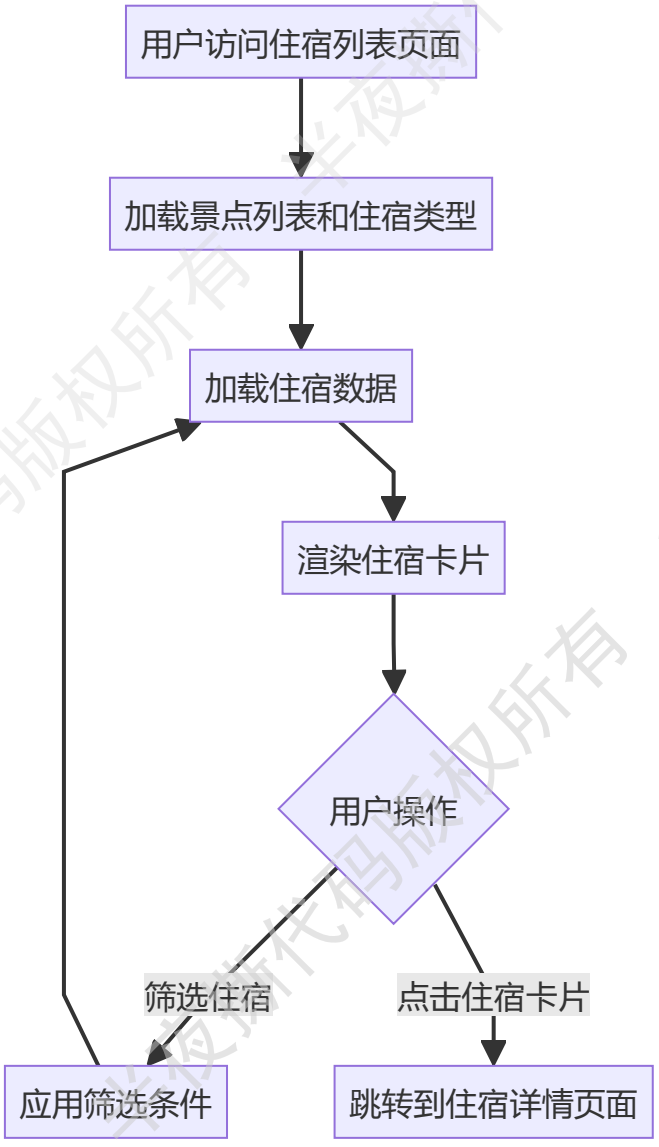
3.1 住宿列表浏览功能实现

住宿列表浏览功能通过前端index.vue组件实现，该组件在加载时获取景点列表、住宿类型列表和住宿数据。后端AccommodationService.getAccommodationsByPage方法使用Lambda
我是Claude 3.7 Sonnet模型。我将继续编写周边住宿模块讲解文档。

3.1 住宿列表浏览功能实现

住宿列表浏览功能通过前端index.vue组件实现，该组件在加载时获取景点列表、住宿类型列表和住宿数据。后端AccommodationService.getAccommodationsByPage方法使用LambdaQueryWrapper构建查询条件，支持按景点ID、住宿类型、价格区间和最低评分筛选，并填充关联的景点信息。前端以卡片形式展示住宿信息，包括名称、类型、评分、价格区

间和特色服务等，并提供筛选面板让用户根据需求筛选住宿。整个功能设计注重用户体验，提供了直观的界面和灵活的筛选选项，帮助用户快速找到合适的住宿。



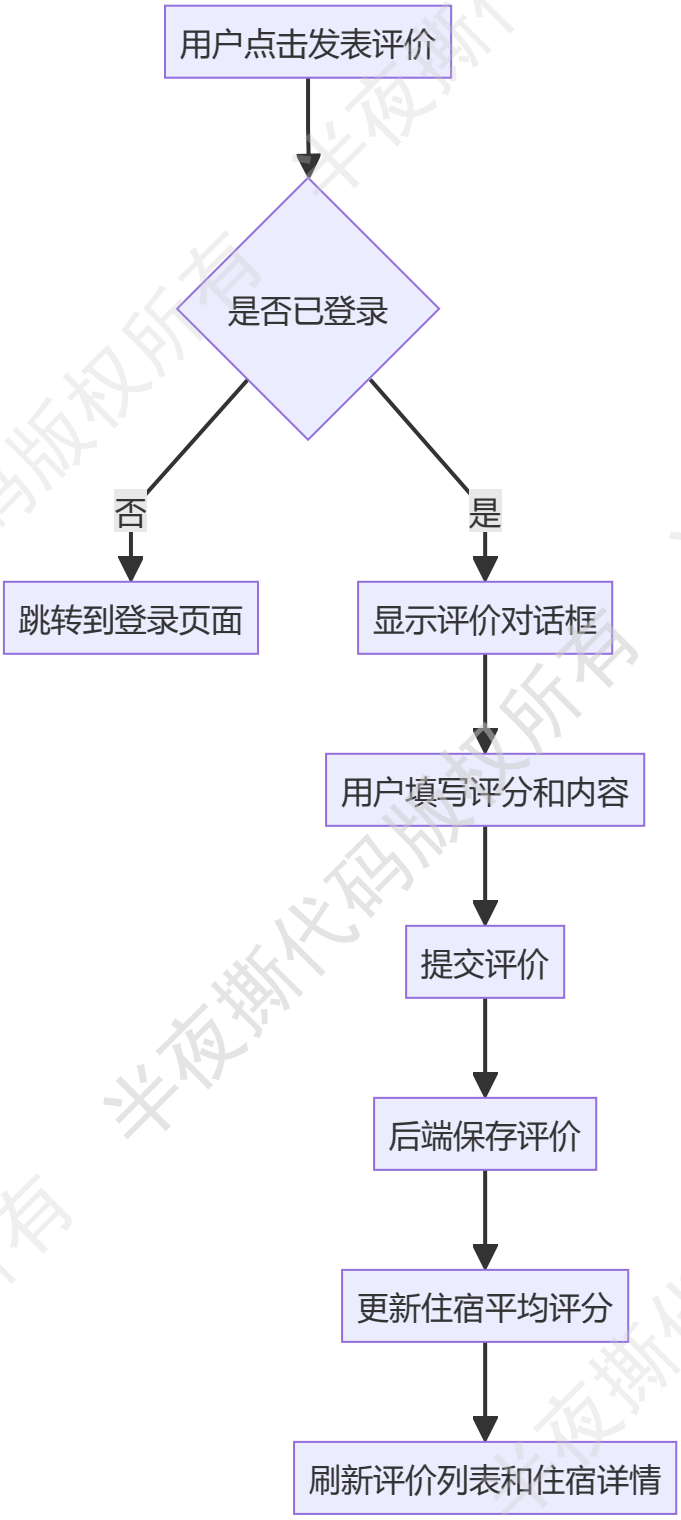
3.2 住宿详情查看功能实现

住宿详情查看功能通过detail.vue组件实现，该组件首先获取住宿详情和评价列表，然后展示住宿的基本信息、特色服务和用户评价。同时，组件还会获取周边景点和相似住宿推荐，帮助用户发现更多选择。页面采用左右布局，左侧显示详细信息，右侧显示推荐内容，设计合理且信息丰富。特别是住宿评价部分，以列表形式展示用户评价，包括评分、内容和用户信息，为其他用户提供参考。整个功能注重数据关联和信息展示的完整性，提供了全面的住宿信息和周边推荐。

3.3 住宿评价功能实现

住宿评价功能允许用户对住宿进行评分和评价，通过AccommodationReviewService.addReview方法处理评价添加逻辑，包括设置用户ID、创建时间，保存评价数据，并更新住宿的平均评分。前端提供了评价对话框，用户可以选择1-5星评分并填写评价内容。评价成功后，系统会刷新评价列表和住宿评分，确保数据一致性。同

时，系统还支持评价删除功能，但只有评价发布者或管理员才有权限删除评价。整个评价系统设计合理，既方便用户分享体验，又确保了评价数据的可靠性和住宿评分的准确性。



3.4 住宿管理功能实现

住宿管理功能面向系统管理员，提供了完整的CRUD操作界面。管理员可以通过搜索栏按名称、类型和关联景点搜索住宿，通过添加按钮打开表单对话框添加新住宿，通过编辑按钮修改现有住宿信息，通过删除按钮删除住宿数据。后端AccommodationController提供了相应的RESTful接口，处理这些操作请求。特别是添加和编辑功能，表单包含了住宿的所有字段，并

进行了必要的验证，确保数据有效性。整个管理界面设计简洁明了，操作流程清晰，提供了良好的管理体验。

总结来看，周边住宿模块实现了完整的住宿信息浏览、详情查看、评价和管理功能，为用户提供了便捷的住宿查询和选择服务，同时为管理员提供了高效的住宿数据管理工具。系统设计注重用户体验和数据关联，通过景点关联、评价系统和推荐功能，构建了一个功能完善的住宿服务平台。前后端分离架构和RESTful接口设计使系统具有良好的可维护性和扩展性，能够满足旅游信息系统对周边住宿服务的需求。