1. 引言

1.1 文档目的

本文档作为《邮驿四方》Web应用程序的软件设计报告，旨在详细、清晰且准确地阐述本项目的软件设计方案。其核心目的为以下三点：

- **记录设计方案：** 详细记录系统架构、模块划分、关键类设计、数据模型设计以及所采用的技术选型和设计原则等方面的决策过程和最终方案。
- **支持评审与交流：** 作为项目设计评审验收的技术文档，方便课程教师、助教以及其他相关人员理解系统设计。

- **奠定维护与扩展基础：** 为项目的后续维护、功能迭代和系统扩展提供重要的参考依据。

1.2 项目背景与概述

- **项目名称：** 《邮驿四方—衣食学行全指南》Web 应用
- **项目说明：** 本项目是 [基于Java语言开发一个AI应用网站] --2025面向对象程序设计课程的一项大作业。
- **项目目标：** 在当前信息分散、获取效率不高的背景下，本项目旨在为北京邮电大学的学生群体开发一个集衣、食、学、行——即时尚穿搭、美食探索、学习笔记管理和智能行程规划——于一体的综合性Web应用。通过提供一个统一、便捷的信息平台和规划辅助工具，帮助学生更有效地获取、管理和利用与校园及都市生活相关的信息，提升生活和学习的便利性。
- 项目名称“邮驿四方”寓意着汇聚四方信息，这四方信息也对应这上述衣、食、学、行四个模块，旨在为用户指引方向。
- **核心功能回顾（简述）：** 本应用主要包含以下核心功能模块：

- **衣 (时尚穿搭):** 提供穿搭推荐列表，集成实时天气信息，支持基于天气的智能“一键穿搭”推荐，以及按场合和风格进行穿搭筛选和详情查看。
- **食 (美食推荐):** 展示丰富的美食信息列表，提供按菜系和评分的筛选、多种排序方式，并支持查看美食详细信息和统计热度（浏览次数）。
- **学 (学习笔记):** 提供完整的学习笔记生命周期管理功能，包括在线创建、查看、编辑、删除，支持按分类组织和筛选笔记，并提供按标题的搜索功能。此外，集成了“邮学家”AI助手，提供学习相关的智能问答服务。
- **行 (行程规划):** 以北京邮电大学西土城路校区为起点，为北邮学子提供到北京市内热门景点的路线规划服务，通过集成高德地图API在地图上直观展示路线，并提供距离、耗时和分段导航指引。

1.3 参考资料

本文档的撰写参考了以下资料：

- **[基于Java语言开发一个AI应用网站] --2025面向对象程序设计课程作业要求.txt**
- **Spring Boot** 官方文档
- **Spring Data JPA** 官方文档
- **Thymeleaf** 官方文档
- 高德地图 **Web** 服务 **API** 文档

- **OpenWeatherMap API** 文档

---

2. 总体设计

2.1 系统架构

《邮驿四方》Web 应用采用了典型的 **三层架构** 与 **MVC模式** 相结合的设计。

系统被逻辑上划分为以下主要层次：

- **表现层：**

- 职责：负责处理用户界面（View）的渲染和用户输入（Controller）。它接收来自用户的请求，调用业务逻辑层提供的服务，并将处理结果呈现给用户。
- 实现方式：主要由 Spring MVC 控制器（`@Controller` 注解的类，如 `WebController` 和各模块中的 `@RestController`）和 Thymeleaf 模板引擎（用于渲染 HTML 页面）组成。前端使用 HTML, CSS 和 JavaScript 来构建用户界面和处理客户端交互。

- **业务逻辑层：**

- 职责：包含系统的核心业务逻辑和规则。它处理表现层传递来的请求，调用数据访问层进行数据操作，并进行必要的数据处理、计算和验证。
- 技术实现：主要由带有 `@Service` 注解的类（如 `StudyNoteService`, `FoodRecommendationService` 等）组成。这些服务类封装了具体的业务流程，不直接与用户界面或数据库交互。

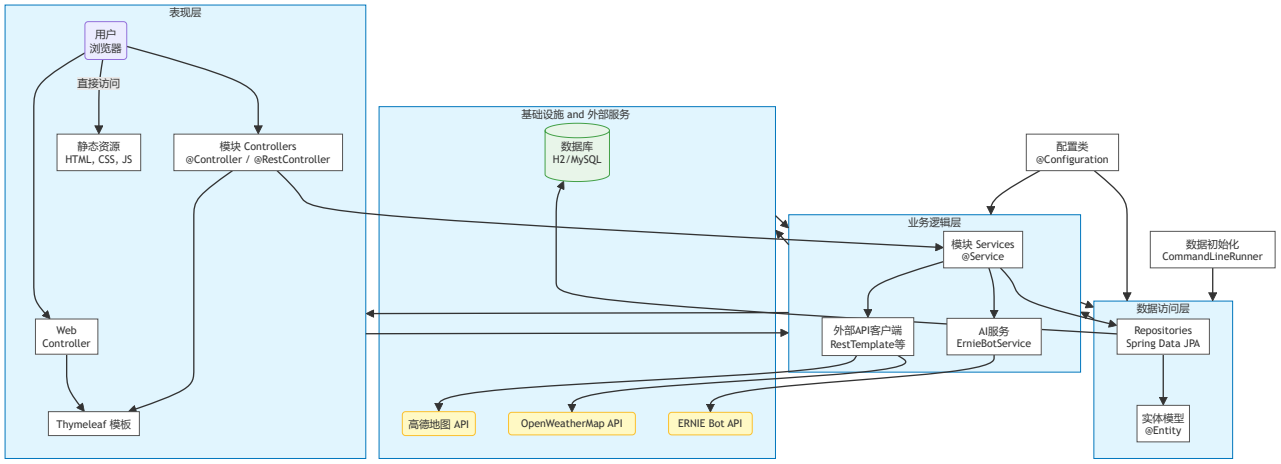
- **数据访问层：**

- 职责：负责与数据库进行交互，执行数据的增、删、改、查操作。它向业务逻辑层提供抽象的数据访问接口，屏蔽了底层数据库的具体实现细节。
- 实现方式：主要由 Spring Data JPA 的 Repository 接口（继承 `JpaRepository`）定义数据操作方法，并通过 JPA 框架将 Java 对象映射到数据库表。为了方便验收，项目使用了 H2 内存数据库作为开发环境，并预留了切换到 MySQL 等关系型数据库的能力。

- **外部服务集成：**

- 职责：负责与外部第三方 API 服务进行通信，获取或发送数据。这部分功能通常由业务逻辑层或专门的服务类负责调用。
- 实现方式：通过 Spring 的 `RestTemplate` 类发起 HTTP 请求，调用高德地图 API（用于路线规划、地理编码、天气）和 OpenWeatherMap API（作为备用天气源），以及 文心一言大模型 API（用于学习助手功能）。

**



- **系统架构图说明：**

- 该图展示了《邮驿四方》Web应用的主要逻辑分层。
- 客户端浏览器通过 HTTP 请求与表现层交互。
- 表现层（由 Spring MVC 控制器）处理用户请求，将请求转发给业务逻辑层。
- 业务逻辑层包含核心业务逻辑，它调用数据访问层进行数据库操作，或调用外部 API 服务获取外部数据。
- 数据访问层通过 Spring Data JPA 与数据库进行持久化交互。
- 外部 API 服务提供特定的功能（地图、天气、文心一言AI），被业务逻辑层调用。
- 静态资源（HTML, CSS, JS 文件）由表现层直接提供给客户端浏览器。

2.2 模块划分

基于“衣、食、学、行”的核心功能，系统在代码层面主要划分为以下业务模块：

- `**com.example.demo.fashion**` **(衣 - 穿搭推荐):**

- 核心功能：管理和展示时尚穿搭方案，集成天气信息，提供基于天气的智能穿搭推荐，支持按场合和风格筛选穿搭。

- 包含组件：`FashionOutfit` (Model), `FashionOutfitRepository` (Repository), `FashionOutfitService` (Service), `FashionOutfitController` (Controller), `WeatherController` (天气相关Controller), `FashionDataInitializer` (用于初始化)。

- `**com.example.demo.food**` **(食 - 美食推荐):**

- 核心功能：管理和展示美食信息，提供按菜系和评分筛选，按多种条件排序，查看美食详情及统计浏览次数。
- 包含组件：`FoodRecommendation` (Model), `FoodRecommendationRepository` (Repository), `FoodRecommendationService` (Service), `FoodRecommendationController` (Controller), `FoodDataInitializer` (数据初始化)。

- `**com.example.demo.study**` **(学 - 学习笔记):**

- 核心功能：提供学习笔记的创建、查看、编辑、删除、分类筛选和搜索功能，以及与AI助手（邮学家）的交互。
- 包含组件：`StudyNote` (Model), `StudyNoteRepository` (Repository), `StudyNoteService` (Service), `StudyNoteController` (Controller), `StudyAssistantController` (AI助手Controller), `StudyDataInitializer` (数据初始化), `ErnieBotService` (AI服务客户端), `ErnieBotConfig`, `RestTemplateConfig` (相关配置)。

- `**com.example.demo.travel**` **(行 - 行程规划):**

- 核心功能：提供从指定起点（北京邮电大学海淀校区）到目的地的路线规划，地图展示路线，显示路线信息。
- 包含组件：`TravelItinerary` (Model - 用于存储示例行程信息，尽管路线规划功能不直接使用此模型), `TravelItineraryRepository` (Repository), `TravelItineraryService` (Service), `TravelApiController` (路线规划API Controller), `Route` (Model - 用于表示路线数据结构), `RouteController` (模拟路线API Controller), `TravelDataInitializer` (数据初始化)。

2.3 技术栈与设计原则

本项目主要基于以下技术栈进行开发：

- **后端:**

- **Java:** 作为主要的编程语言。
- **Spring Boot**
- **Spring Data JPA :** 简化了 Repository 层的开发
- **Gradle:** 项目构建和依赖管理工具。

- **前端:**

- **三件套：HTML5, CSS3，JavaScript**

- **数据库:**

- **H2 数据库:** 轻量级数据库，方便开发和测试。
- **MySQL (可选):**`pom.xml` 中包含连接器，可以切换到MySQL数据库。

- **外部 API:**

- **高德地图 API (Web 服务, 路线规划, 天气):** 提供地理编码、多种交通方式路线规划和实时天气信息。
- **OpenWeatherMap API:** 作为高德地图天气 API 的备用方案。
- **ERNIE Bot API/文心一眼:** 用于支持学习笔记模块的学习AI助手功能。

在软件设计过程中，我按照课程群ppt，遵循了以下核心设计原则：

- **单一职责原则 :** 每个类或模块应只有一个主要职责。
- **关注点分离 :** 系统的分层架构和模块划分（衣、食、学、行）体现了这一原则。
- **依赖倒置原则 :** 高层模块不应依赖于低层模块，它们都应该依赖于抽象。抽象不应该依赖于细节，细节应该依赖于抽象。
- **开放封闭原则**

---

3. 详细设计

详细设计阶段基于总体设计（即上一部分）确定的系统架构和模块划分，对每个模块、数据结构以及通用服务进行具体的设计描述。

3.1 数据模型设计

系统的数据存储通过关系型数据库实现，使用 Spring Data JPA 进行对象关系映射。以下是系统中的主要数据实体及其关键属性、数据类型和约束：

- **StudyNote**

- 对应表名：`study_notes` (通过 `@Table` 注解明确)
- 职责：存储用户的学习笔记信息。
- 属性：

- `id`: `Long`，主键 (`@Id`), 自增 (`@GeneratedValue(strategy = GenerationType.IDENTITY)`), 非空。数据库对应 `BIGINT` 类型。
- `title`: `String`，笔记标题，非空 (`@Column(nullable = false)`), 数据库对应 `VARCHAR` 类型。
- `content`: `String`，笔记内容，非空 (`@Column(nullable = false)`), 长度设置为 1000 (`length = 1000`)，数据库对应 `VARCHAR` 或 `TEXT` 类型。
- `category`: `String`，笔记分类，非空 (`@Column(nullable = false)`), 数据库对应 `VARCHAR` 类型。
- `createdAt`: `LocalDateTime`，创建时间，非空 (`@Column(name = "created_at", nullable = false)`), `@PrePersist` 自动设置，数据库对应 `TIMESTAMP` 类型。
- `updatedAt`: `LocalDateTime`，最后更新时间，非空 (`@Column(name = "updated_at", nullable = false)`), `@PrePersist` 和 `@PreUpdate` 自动设置，数据库对应 `TIMESTAMP` 类型。

- 关系：独立实体，与其他实体无直接关系。

- **FoodRecommendation**

- 对应表名：`food_recommendation` (由 `@Table` 指定)
- 职责：存储美食推荐信息。
- 属性：

- `id`: `Long`，主键 (`@Id`), 自增 (`@GeneratedValue(strategy = GenerationType.IDENTITY)`), 非空。数据库对应 `BIGINT` 类型。
- `name`: `String`，美食名称，非空，数据库对应 `VARCHAR` 类型。

- `description`: `String`，描述，数据库对应 `VARCHAR` 或 `TEXT` 类型。
- `cuisine`: `String`，菜系，非空，数据库对应 `VARCHAR` 类型。
- `location`: `String`，地点，非空，数据库对应 `VARCHAR` 类型。
- `imageUrl`: `String`，图片URL，数据库对应 `VARCHAR` 类型。
- `rating`: `Double`，评分，数据库对应 `DOUBLE` 或 `DECIMAL` 类型。
- `createdAt`: `LocalDateTime`，创建时间，`@PrePersist` 自动设置，数据库对应 `TIMESTAMP` 类型。
- `viewCount`: `Integer`，浏览次数，默认为0，数据库对应 `INT` 类型。

- 关系：独立实体，与其他实体无直接关系。

- **TravelItinerary**

- 对应表名：`travel_itinerary` (由 `@Table` 指定)
- 职责：存储预设或推荐的旅行行程信息。
- 属性：

- `id`: `Long`，主键 (`@Id`), 自增 (`@GeneratedValue(strategy = GenerationType.IDENTITY)`), 非空。数据库对应 `BIGINT` 类型。
- `title`: `String`，行程标题，数据库对应 `VARCHAR` 类型。
- `destination`: `String`，目的地，数据库对应 `VARCHAR` 类型。
- `description`: `String`，描述，数据库对应 `VARCHAR` 或 `TEXT` 类型。
- `startDate`: `LocalDate`，开始日期，数据库对应 `DATE` 类型。
- `endDate`: `LocalDate`，结束日期，数据库对应 `DATE` 类型。
- `durationDays`: `Integer`，持续天数，可以根据开始/结束日期计算，数据库对应 `INT` 类型。
- `imageUrl`: `String`，图片URL，数据库对应 `VARCHAR` 类型。
- `budget`: `Double`，预算，数据库对应 `DOUBLE` 或 `DECIMAL` 类型。
- `createdAt`: `LocalDateTime`，创建时间，`@PrePersist` 自动设置，数据库对应 `TIMESTAMP` 类型。

- 关系：独立实体，与其他实体无直接关系。

- **FashionOutfit**

- 对应表名：`fashion_outfit` (由 `@Table` 指定)
- 职责：存储时尚穿搭方案信息。
- 属性：

- `id`: `Long`，主键 (`@Id`), 自增 (`@GeneratedValue(strategy = GenerationType.IDENTITY)`), 非空。数据库对应 `BIGINT` 类型。
- `title`: `String`，穿搭标题，数据库对应 `VARCHAR` 类型。
- `description`: `String`，描述，数据库对应 `VARCHAR` 或 `TEXT` 类型。
- `season`: `String`，适用季节，数据库对应 `VARCHAR` 类型。
- `occasion`: `String`，适用场合，非空，数据库对应 `VARCHAR` 类型。
- `style`: `String`，风格，非空，数据库对应 `VARCHAR` 类型。
- `imageUrl`: `String`，图片URL，数据库对应 `VARCHAR` 类型。
- `items`: `String`，搭配单品列表（存储为文本，可以理解为格式化后的JSON字符串或纯文本列表），非空，数据库对应 `VARCHAR` 或 `TEXT` 类型。
- `createdAt`: `LocalDateTime`，创建时间，`@PrePersist` 自动设置，数据库对应 `TIMESTAMP` 类型。

- 关系：独立实体，与其他实体无直接关系。

3.2 模块详细设计

3.2.1 学习笔记模块

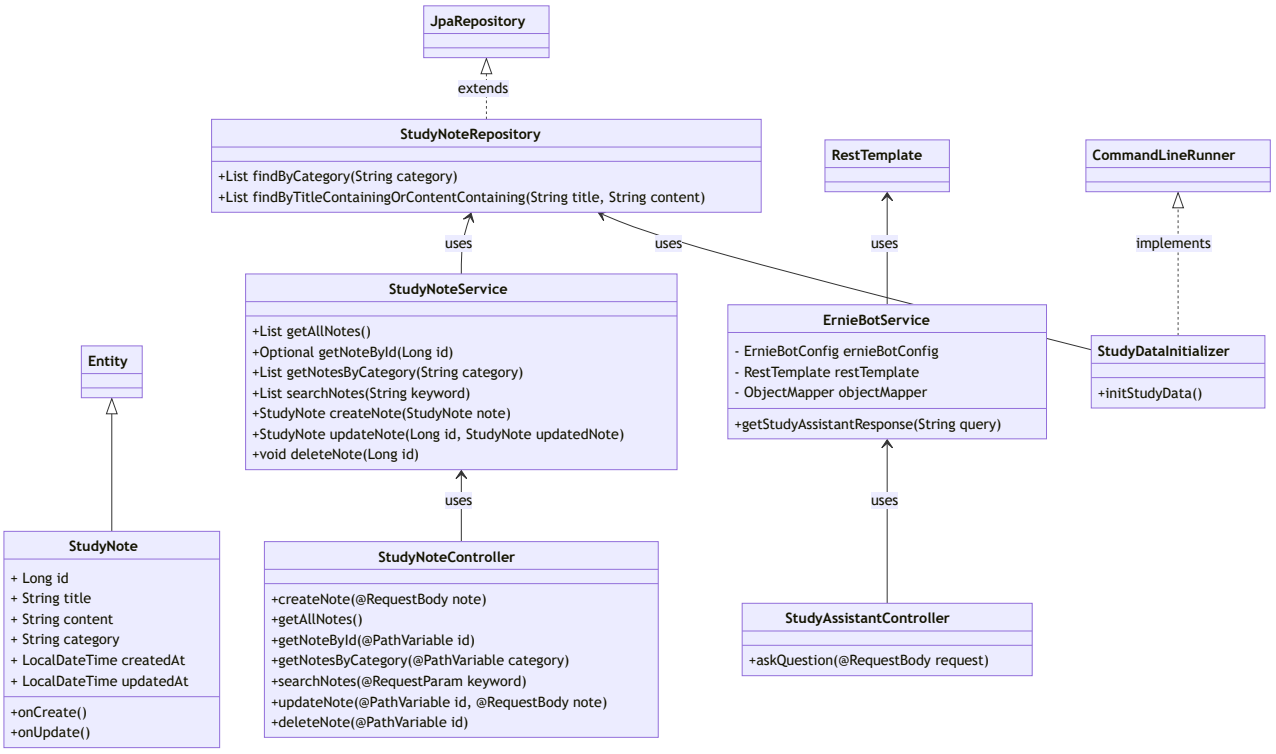
- **模块功能:**
- 用户在线创建新的学习笔记，包括填写标题、选择分类和输入内容。

- 用户可以浏览所有已创建的学习笔记列表，每条笔记以卡片形式展示摘要信息。
- 用户点击笔记卡片可查看笔记的完整内容详情。
- 用户可以在笔记详情界面选择编辑已有的笔记，修改其标题、分类或内容。
- 用户可以在笔记详情界面选择删除笔记，删除前提供确认提示。
- 用户可以通过页面左侧的分类列表筛选显示特定分类的笔记。
- 用户可以在页面顶部的搜索框中输入关键词，按标题模糊搜索笔记。
- 用户可以在“邮学家”聊天界面输入问题，获取 AI 助手（文心一眼大模型）的学习相关回答。

- 关键类及其职责：

- `StudyNote` (`com.example.demo.study.model.StudyNote`): 数据实体，表示一篇学习笔记。
- `StudyNoteRepository` (`com.example.demo.study.repository.StudyNoteRepository`): 数据访问接口 (`@Repository`)，继承 `JpaRepository`，提供对 `StudyNote` 实体进行数据库操作的标准方法。通过自定义方法名 (`findByCategory`, `findByTitleContainingOrContentContaining`) 实现特定查询。
- `StudyNoteService` (`com.example.demo.study.service.StudyNoteService`): 业务逻辑服务 (`@Service`)，封装学习笔记的业务处理逻辑。接收 Controller 的请求，调用 Repository 执行数据库操作，并进行必要的业务判断和数据转换。
- `StudyNoteController` (`com.example.demo.study.controller.StudyNoteController`): RESTful API 控制器 (`@RestController`)，处理来自前端 `/api/study-notes` 路径下的 HTTP 请求。负责接收请求参数、调用 `StudyNoteService` 处理业务、并返回 JSON 格式的响应。
- `StudyAssistantController` (`com.example.demo.study.controller.StudyAssistantController`): RESTful API 控制器 (`@RestController`)，处理 AI 助手相关的请求 (`/api/study-assistant/ask`)。接收用户问题，调用 `ErnieBotService` 获取 AI 回答，并返回 JSON 响应。
- `ErnieBotService` (`com.example.demo.study.service.ErnieBotService`): AI 服务客户端 (`@Service`)，负责与外部 ERNIE Bot API 进行交互。封装了构建请求、发送 HTTP 请求 (`RestTemplate`)、解析 API 响应的逻辑。
- `StudyDataInitializer` (`com.example.demo.study.config.StudyDataInitializer`): 数据初始化类 (`@Configuration`, `CommandLineRunner`)，在应用启动时检查数据库，如果为空则插入示例学习笔记数据。

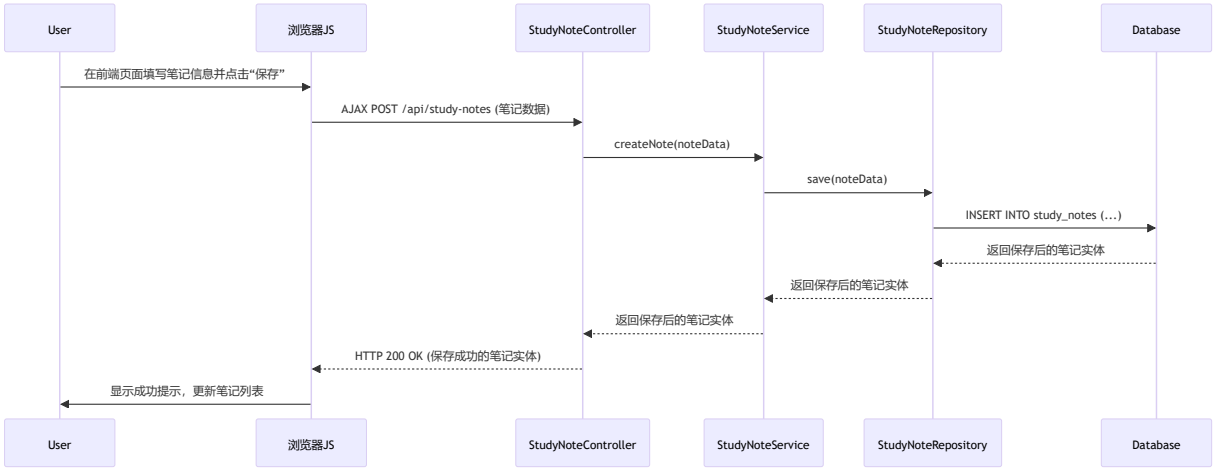
- 类图：



- 图 学习笔记模块类图说明:

- 该图展示了学习笔记模块中的核心类及其相互依赖关系。
- `StudyNote` 是数据实体类，与数据库表直接对应。
- `StudyNoteRepository` 是数据访问接口，定义了与数据库交互的方法，`StudyNoteService` 依赖并调用这些方法。
- `StudyNoteController` 是表现层入口，依赖 `StudyNoteService` 完成业务逻辑。
- `StudyAssistantController` 依赖 `ErnieBotService` 与外部 AI 服务通信。
- `ErnieBotService` 依赖 `RestTemplate` 和 `ErnieBotConfig` 来构建和发送 API 请求。
- `StudyDataInitializer` 在应用启动时使用 `StudyNoteRepository` 填充初始数据。

- 核心流程 介绍- 以“创建新笔记”为例的时序图:



- **时序图说明：**

- 该图 展示了用户通过前端界面创建一篇新学习笔记的流程。
- 用户在浏览器中操作，前端 JavaScript 捕捉用户行为并构造 AJAX 请求。
- HTTP POST 请求被发送到后端 `StudyNoteController` 的相应端点。
- Controller 调用 `StudyNoteService` 的 `createNote` 方法处理业务。
- Service 调用 `StudyNoteRepository` 的 `save` 方法将数据持久化到数据库。
- 数据库执行 INSERT 操作并返回结果。
- Service 将结果返回给 Controller，Controller 封装为 HTTP 响应返回给前端。
- 前端 JavaScript 处理响应，向用户显示提示并更新界面

前端展示

3.2.2 美食推荐模块

- **模块功能:**

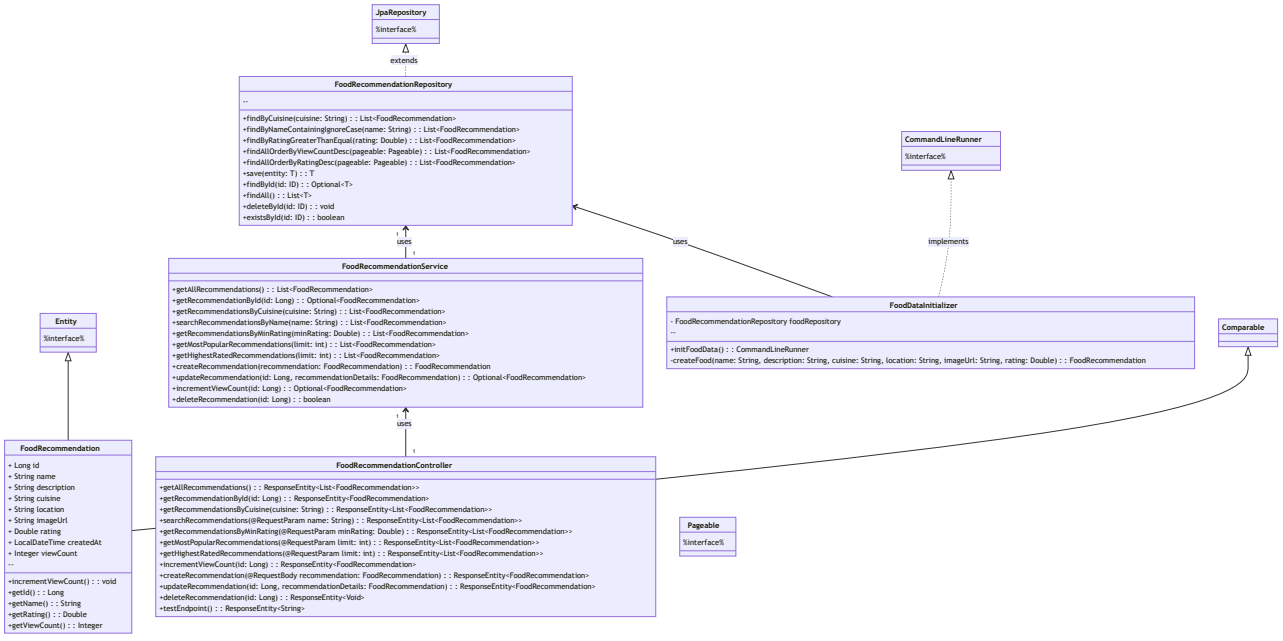
- 用户可以浏览所有美食推荐列表，列表以卡片形式展示摘要信息。
- 用户可以通过页面左侧的筛选面板，按菜系和最低评分筛选美食列表。
- 用户可以通过下拉菜单选择按热度、评分、名称或发布时间对美食列表进行升序或降序排序。
- 用户点击美食卡片可以查看该美食的详细信息，并触发后端记录一次浏览次数以统计热度。
- （如果前端有添加/编辑功能）用户可以添加新的美食推荐，编辑已有的美食推荐，或删除某个美食推荐。

- **关键类及其职责 (Key Classes and Responsibilities):**

- `FoodRecommendation` (`com.example.demo.food.model.FoodRecommendation`):
- 实体类 (`@Entity`), 映射到数据库表，定义美食推荐的数据结构。包含 `incrementViewCount()` 方法用于增加浏览次数。
- `FoodRecommendationRepository` (`com.example.demo.food.repository.FoodRecommendationRepository`):

- 数据访问接口 (`@Repository`), 继承 `JpaRepository<FoodRecommendation, Long>`。提供标准的 CRUD 方法。定义了多种查询方法：`findByCuisine()`, `findByNameContainingIgnoreCase()`, `findByRatingGreaterThanEqual()`, 以及通过 `@Query` 定义的 `findAllOrderByViewCountDesc()` 和 `findAllOrderByRatingDesc()`。
- `FoodRecommendationService` (`com.example.demo.food.service.FoodRecommendationService`):
- 业务逻辑服务类 (`@Service`), 封装美食推荐的业务处理逻辑。包括接收 Controller 请求、调用 Repository 查询/保存数据、处理筛选/排序逻辑（尽管排序由 Repository 的 `@Query` 直接完成）、以及 `incrementViewCount()` 方法中更新浏览次数并保存的逻辑。
- `FoodRecommendationController` (`com.example.demo.food.controller.FoodRecommendationController`):
- **RESTful API** 控制器类 (`@RestController`, `@RequestMapping("/api/food-recommendations")`), 处理与美食推荐资源相关的 HTTP 请求。提供获取所有、根据 ID、根据菜系、根据关键词搜索、根据最低评分、根据热度排序、根据评分排序、统计浏览次数、创建、更新、删除等接口。
- `FoodDataInitializer` (`com.example.demo.food.config.FoodDataInitializer`):
- 数据初始化组件 (`@Configuration`, `CommandLineRunner`)，在应用启动时向空的数据库表中填充示例美食数据。

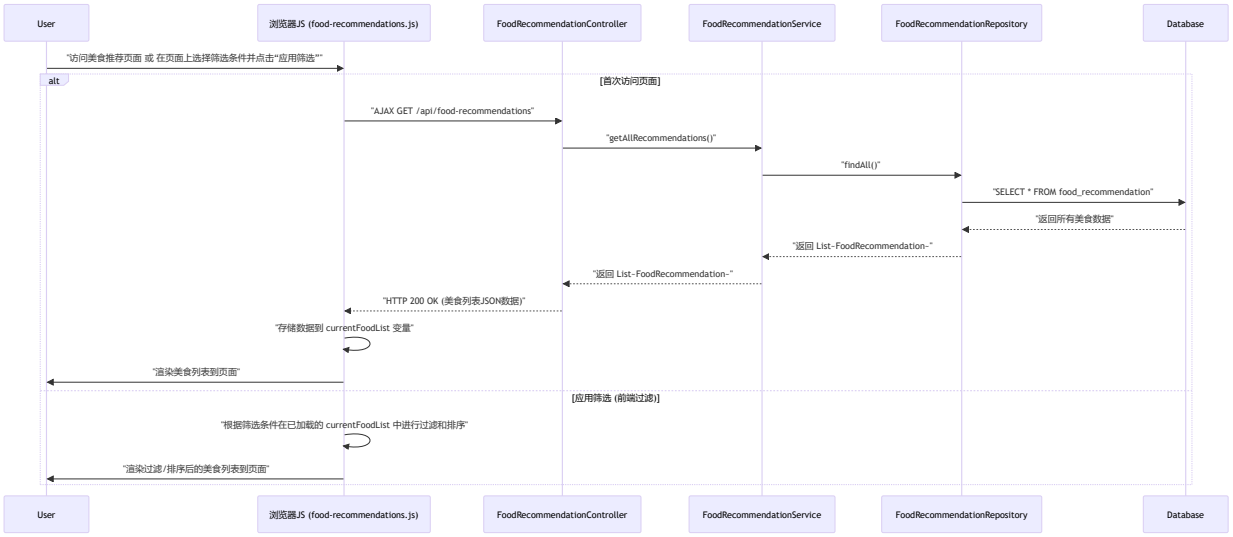
- 类图：



- 美食推荐模块类图说明：

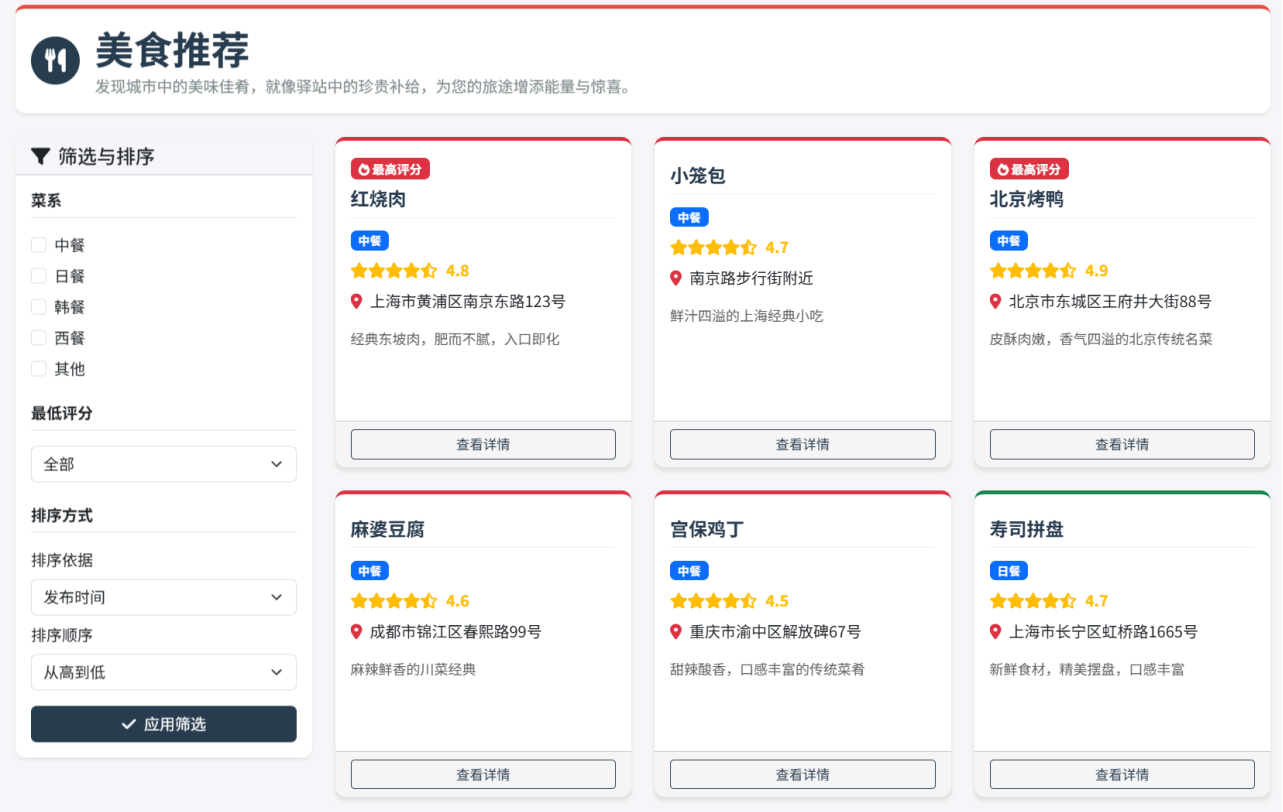
- 该图展示了美食推荐模块中的核心类及其关系。
- `FoodRecommendation` 是数据实体。
- `FoodRecommendationRepository` 接口继承 `JpaRepository`，提供数据访问方法，并定义了多种查找美食的方法，包括基于属性的派生查询和基于 `@Query` 的复杂查询（如按浏览次数和评分排序）。
- `FoodRecommendationService` 依赖 `FoodRecommendationRepository` 实现业务逻辑，如获取推荐列表、筛选、搜索、统计浏览次数等。
- `FoodRecommendationController` 依赖 `FoodRecommendationService` 处理来自前端的 HTTP 请求，并将数据格式化为 JSON 返回。
- `FoodDataInitializer` 负责初始化示例美食数据。

- **核心流程 - 以“查看美食列表及应用筛选”为例的时序图：**



- 该图展示了用户首次访问美食推荐页面或在页面上应用筛选条件后，前端获取并显示美食列表的流程。
- 用户操作触发前端 JavaScript (BrowserJS) 发起异步 AJAX 请求.
- 首次访问页面时，前端发起 GET 请求 `/api/food-recommendations` 获取**所有**美食数据.
- 请求通过 Controller 调用 Service，再调用 Repository，从数据库获取所有数据.
- 后端返回所有美食数据给前端 JavaScript.
- 前端 JavaScript 将获取到的所有数据存储在 `currentFoodList` 变量中.
- **应用筛选/排序功能主要在前端实现.** 用户选择筛选或排序条件时，前端 JavaScript 直接操作 `currentFoodList` 变量进行过滤和排序，然后更新页面的显示，而不再向后端发送新的筛选/排序请求。

前端展示



3.2.3 行程规划模块

- **模块功能:**

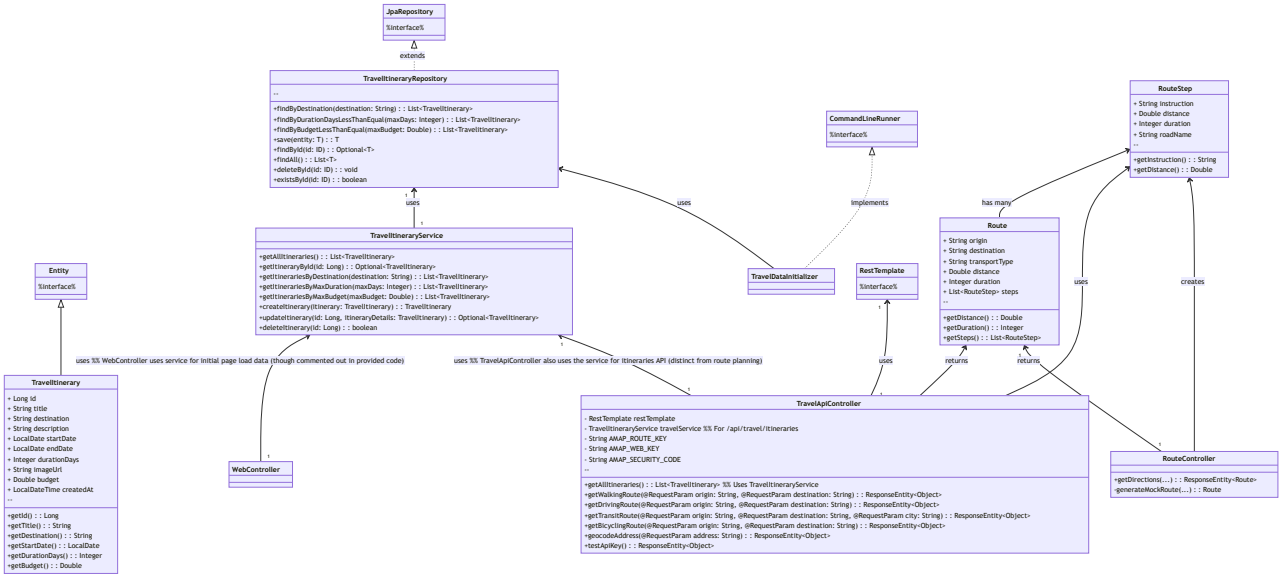
- 展示从北京邮电大学到热门景点的路线规划界面。
- 用户可以选择目的地。
- 系统调用外部高德地图 API 进行路线规划。
- 在页面集成的地图上直观显示规划出的路线几何图形。
- 在信息面板显示路线的总距离、预计耗时和分段导航指引。

- **关键类及其职责：**

- `TravelItinerary` (`com.example.demo.travel.model.TravelItinerary`):
- **实体类** (`@Entity`), 映射到数据库表，定义预设旅行行程方案的数据结构（如标题、目的地、预算等）。
- `TravelItineraryRepository` (`com.example.demo.travel.repository.TravelItineraryRepository`):
- **数据访问接口** (`@Repository`), 继承 `JpaRepository<TravelItinerary, Long>`，提供对 `TravelItinerary` 实体的 CRUD 操作以及根据目的地、天数、预算等属性的查询方法。
- `TravelItineraryService` (`com.example.demo.travel.service.TravelItineraryService`):

- 业务逻辑服务类 (`@Service`), 封装预设行程方案的业务逻辑，如获取列表、根据属性查询、创建、更新、删除。
- `TravelApiController` (`com.example.demo.travel.controller.TravelApiController`):
- **RESTful API** 控制器类 (`@RestController`, `@RequestMapping("/api/travel")`), 负责处理与路线规划 API 相关的请求。作为高德地图 API 的后端代理，接收前端请求（如 `/api/travel/walking-route`），使用 `RestTemplate` 调用相应的高德地图 Web 服务 API，并将高德 API 返回的结果直接或稍作处理后返回给前端。也包含地理编码 `/api/travel/geocode` 和 API Key 测试 `/api/travel/test-key` 接口。
- `Route` (`com.example.demo.travel.model.Route`) 和 `Route.RouteStep` (`com.example.demo.travel.model.Route.RouteStep`):
- **数据传输对象 (DTO)**。`Route` 类用于封装从高德地图 API 获取的路线规划结果数据结构（起点、终点、类型、距离、时长、步骤列表），`RouteStep` 表示路线中的每个步骤。这些类主要用于后端接收高德 API 响应和向前端发送数据。
- `RouteController` (`com.example.demo.travel.controller.RouteController`):
- **RESTful API** 控制器类 (`@RestController`, `@RequestMapping("/api/routes")`), 它提供了基于模拟数据生成路线的功能 (`/api/routes/directions`)。
- `TravelDataInitializer` (`com.example.demo.travel.config.TravelDataInitializer`):
- 数据初始化组件 (`@Configuration`, `CommandLineRunner`)

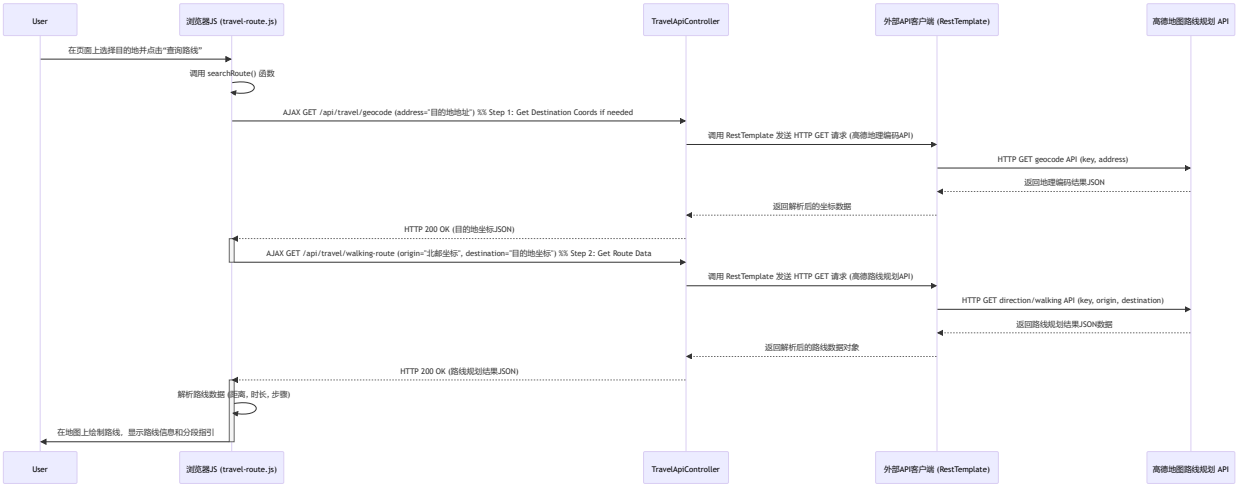
- 类图



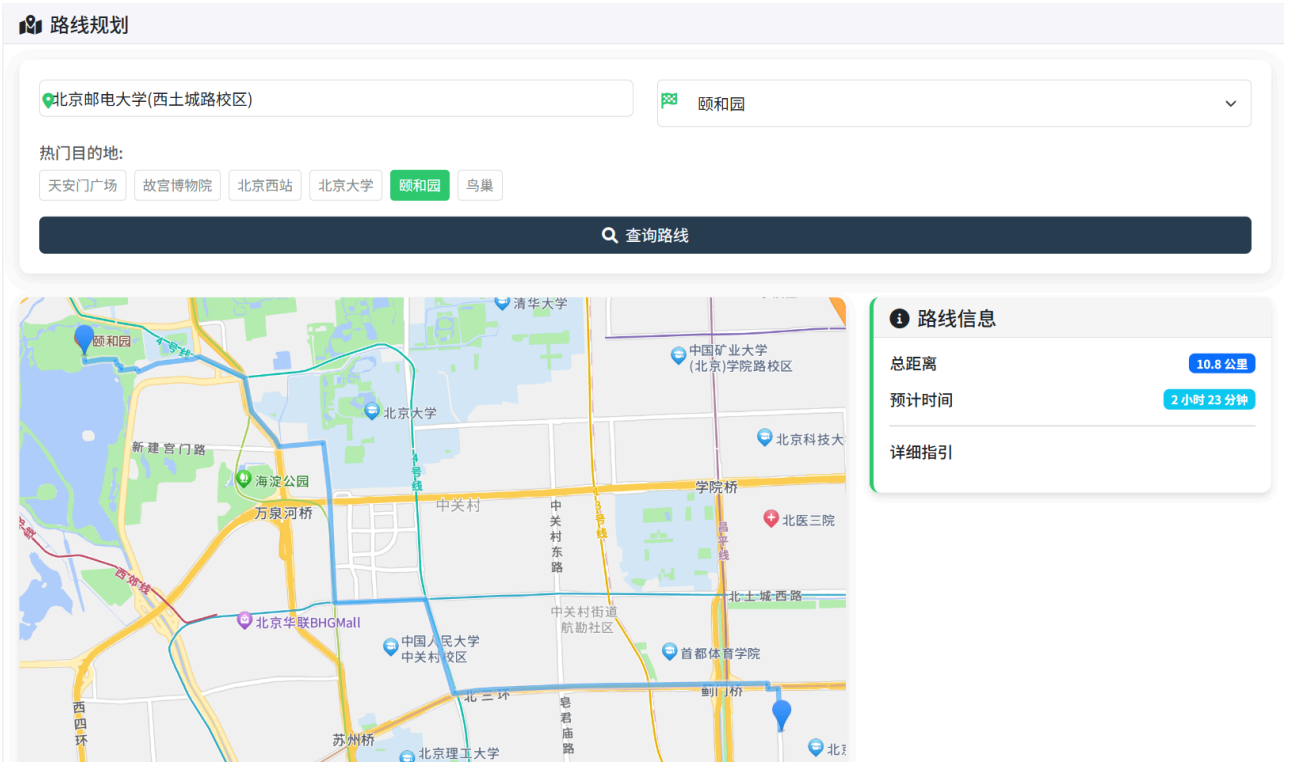
- **行程规划模块类图说明：**

- 该图展示了行程规划模块相关的类及其关系。
- `TravelItinerary` 和 `TravelItineraryRepository` / `Service` / `Controller` 负责预设行程方案的管理和 API 。
- 核心的路线规划功能由 `TravelApiController` 负责，它通过 `RestTemplate` 调用外部高德地图 API获取路线数据。
- `Route` 和 `Route.RouteStep` 是用于封装从高德 API 获取的路线结果的 DTO，在 `TravelApiController` 中被使用和返回。
- `TravelDataInitializer` 用于初始化预设行程方案数据。

- **核心流程 以“根据目的地获取路线规划”为例的时序图：**



前端展示



3.2.4 穿搭推荐模块

- 模块功能:

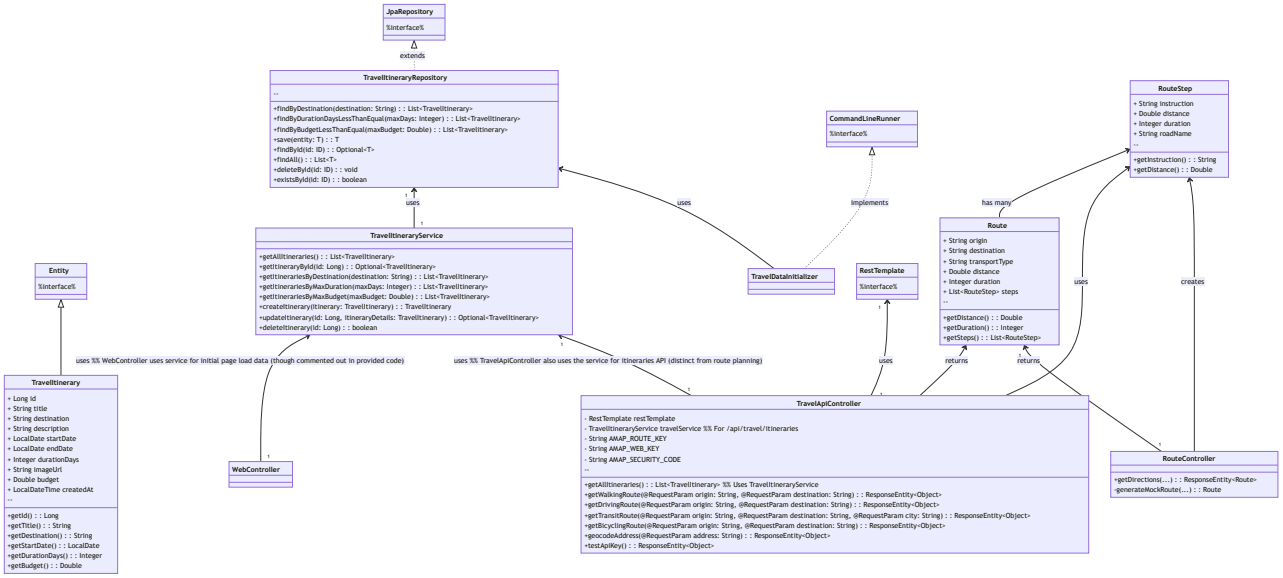
- 展示多种穿搭方案列表，每个方案包含图片、标题、场合、风格、描述和搭配单品。
- 集成实时天气预报功能，显示北京的当前天气状况。
- 根据当前天气情况智能推荐合适的穿搭组合。
- 用户可以通过页面左侧的筛选面板，按一个或多个场合和风格筛选穿搭方案。
- 提供三种筛选模式：同时满足所有选择、仅场合匹配、仅风格匹配。
- 用户可以重置所有筛选条件。
- 用户点击穿搭卡片可以查看穿搭方案的详细信息。

- 关键类及其职责：

- `FashionOutfit` (`com.example.demo.fashion.model.FashionOutfit`):
- 实体类 (`@Entity`), 映射到数据库表，定义穿搭方案的数据结构。
- `FashionOutfitRepository` (`com.example.demo.fashion.repository.FashionOutfitRepository`): 数据访问接口 (`@Repository`), 继承 `JpaRepository<FashionOutfit, Long>`。提供对 `FashionOutfit` 实体的 CRUD 操作以及根据季节、场合、风格等属性的查询方法。

- `FashionOutfitService` (`com.example.demo.fashion.service.FashionOutfitService`):
- **业务逻辑服务类** (`@Service`), 封装穿搭推荐的业务逻辑，如获取列表、根据属性查询、创建、更新、删除。
- `FashionOutfitController` (`com.example.demo.fashion.controller.FashionOutfitController`): **RESTful API 控制器类** (`@RestController`, `@RequestMapping("/api/fashion-outfits")`), 处理与穿搭推荐资源相关的 HTTP 请求。提供获取所有、根据 ID、根据季节、根据场合、根据风格、创建、更新、删除等接口。
- `WeatherController` (`com.example.demo.fashion.controller.WeatherController`):
- **RESTful API 控制器类** (`@RestController`), 处理天气信息相关的请求 (`/api/weather/today`)。作为高德地图天气 API 和 OpenWeatherMap API 的后端代理，调用外部 API 获取天气数据，并将其返回给前端。
- `FashionDataInitializer` (`com.example.demo.fashion.config.FashionDataInitializer`):
- **数据初始化组件** (`@Configuration`, `CommandLineRunner`)，在应用启动时向空的数据库表中填充示例穿搭方案数据。

- 类图：

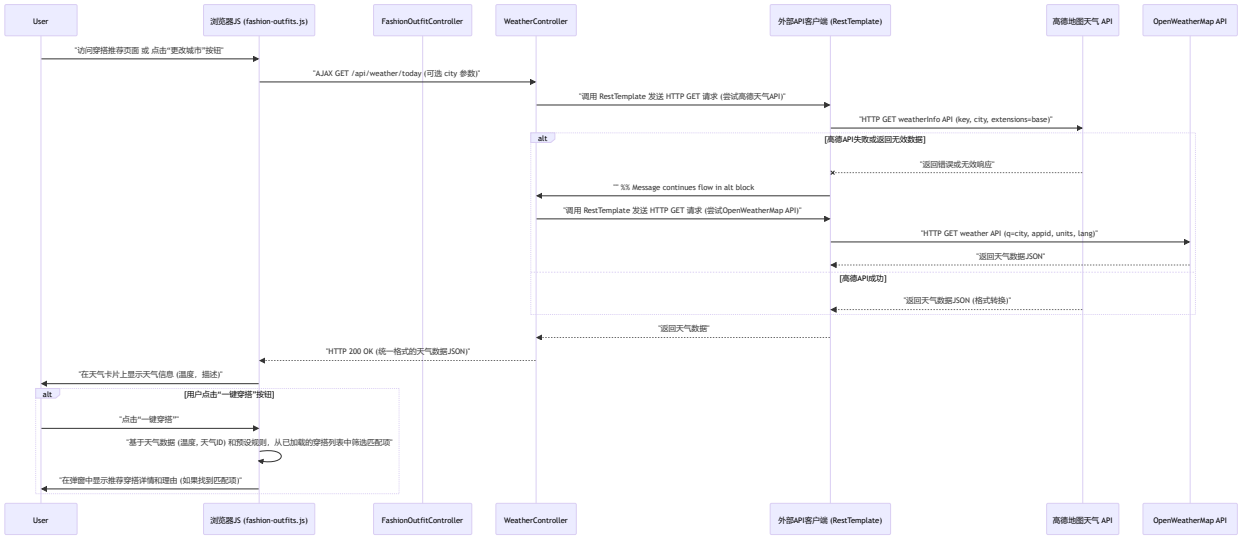


- 类图说明:

- 该图展示了穿搭推荐模块中的核心类及其关系。
- `FashionOutfit` 是数据实体。

- `FashionOutfitRepository` 接口继承 `JpaRepository`，提供数据访问方法，并定义了按季节、场合、风格查找穿搭的方法。
- `FashionOutfitService` 依赖 `FashionOutfitRepository` 实现业务逻辑。
- `FashionOutfitController` 依赖 `FashionOutfitService` 处理来自前端的 HTTP 请求。
- `WeatherController` 独立负责天气信息的获取，作为外部 API 的客户端，依赖 `RestTemplate`。它包含根据描述判断天气 ID 以及从备用 API 获取数据的方法。
- `FashionDataInitializer` 负责初始化示例穿搭数据。

- **核心流程 - 以“获取天气信息并推荐穿搭”为例：**



前端展示

4. 设计模式

在《邮驿四方》Web 应用的设计与实现过程中，我遵循并应用了多种设计模式，以提高代码的可维护性、可读性、灵活性和可扩展性。

4.1 总体设计模式

在系统的高层次架构上，我主要采用了以下两种模式：

- **三层架构：**

- 将应用程序的职责划分为水平层次，每一层只依赖于其下一层或同层。典型的分层包括表现层、业务逻辑层和数据访问层。
- **项目中的应用：**《邮驿四方》严格遵循了三层架构，将代码组织在 `controller` (表现层)、`service` (业务逻辑层) 和 `repository`/`model` (数据访问层) 包中。`WebController` 和模块的 `@Controller`/`@RestController` 位于表现层，处理用户请求；`@Service` 类位于业务逻辑层，实现核心业务逻辑；`@Repository` 接口和 `@Entity` 类位于数据访问层，负责数据持久化。外部 API 调用作为一种特殊的基础设施服务，通常在业务逻辑层通过专门的客户端（如 `ErnieBotService`）或直接由 Service/Controller (如 `TravelApiController`, `WeatherController`) 调用 `RestTemplate` 完成。

- **MVC 模式：**

- 将应用程序分为三个相互关联的部分：Model (数据模型和业务逻辑)、View (用户界面) 和 Controller (处理用户输入并更新模型和视图)。
- **项目中的应用：**在表现层，本项目采用了 Spring MVC 实现 MVC 模式：

- **Controller:** 由 `@Controller` 或 `@RestController` 注解的类实现，负责接收来自用户的 HTTP 请求，调用相应的 Service 处理业务逻辑，并决定返回哪一个 View 或 JSON 数据。
- **Model:** 在狭义上指业务逻辑处理过程中操作的数据对象（如 `StudyNote`, `FoodRecommendation` 实体类），在 Spring MVC 中也指用于在 Controller 和 View 之间传递数据的 `org.springframework.ui.Model` 对象或方法返回值（如 `ModelAndView`, `ResponseEntity`）。
- **View:** 由 Thymeleaf 模板 (`.html` 文件在 `src/main/resources/templates`) 实现，负责根据 Controller 传递过来的数据进行渲染，生成最终的 HTML 页面呈现给用户。静态资源 (CSS, JS) 也支持 View 的呈现和交互。

4.2 代码级设计模式

在具体的模块实现和代码编写层面，我也应用了一些常用的设计模式：

- 仓储模式

- 即在领域对象和数据映射层之间提供一个抽象层，用于数据访问操作。它看起来像一个对象集合，通过领域特定的方法访问对象，将底层数据存储细节（如数据库连接、SQL语句）封装起来。
- **项目中的应用:** Spring Data JPA 的 Repository 接口（如 `StudyNoteRepository`, `FoodRecommendationRepository` ）是仓储模式的典型实现。它们继承 `JpaRepository`，无需编写实现代码即可获得基本的 CRUD 功能。通过定义 `findBy...`, `searchBy...` 等方法，提供了领域特定的数据查询接口。

- 依赖注入

- 即不是由对象自身创建或查找其依赖的对象，而是由外部容器（如 Spring IoC 容器）负责创建依赖的对象，并通过构造函数、Setter 方法或字段等方式将其注入到需要它的对象中。
- **项目中的应用:** Spring 框架广泛应用于依赖注入。通过在类的成员变量或构造函数上使用 `@Autowired` 注解，Spring 容器会自动找到并注入相应的 Bean（如将 `StudyNoteRepository` Bean 注入到 `StudyNoteService` 中，将 `StudyNoteService` Bean 注入到 `StudyNoteController` 中）。`RestTemplate` Bean 也通过 `@Autowired` 注入到需要调用外部 API 的类中。

- 策略模式

- 即定义一系列算法，将每个算法封装起来，并使它们可以相互替换。策略模式使得算法可以独立于使用它的客户端而变化。
- **项目中的应用:**

- **美食筛选与排序:** 在 `food-recommendations.js` 中，前端根据用户选择的筛选条件（菜系、最低评分）和排序方式（依据、顺序），在客户端对已加载的 `currentFoodList` 数据进行过滤和排序。虽然后端没有使用单独的策略类，但前端 JavaScript 代码中包含了根据不同条件（如 `food.cuisine`, `food.rating`）应用不同逻辑（`filter`, `sort`）的实现，这体现了根据用户选择的“策略”来执行不同的数据处理行为。

- **穿搭推荐天气匹配:** 在 `fashion-outfits.js` 的“一键穿搭”功能中，前端 JavaScript 根据天气数据（温度、天气 ID）通过一系列 `if/else if` 语句来判断推荐哪种场合和风格的穿搭。这可以看作是一种简单的基于规则的“推荐策略”选择。

- **适配器模式**

- 即将一个类的接口转换成客户端期望的另一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。
- **项目中的应用:** `WeatherController` 在获取天气信息时扮演了适配器的角色。它需要从两个不同的外部 API (高德地图天气 API 和 OpenWeatherMap API) 获取数据，而这两个 API 返回的 JSON 数据格式可能不同。`WeatherController` 调用这两个 API，处理它们的响应，并尝试将数据转换为一个相对统一的格式（例如，一个包含 `weather` 数组和 `main` 对象的 Map）返回给前端。前端 JavaScript (`fashion-outfits.js`) 则期望接收这种统一格式的数据进行处理 (`handleWeatherData` 函数)。

- **服务门面模式 :**

- 即为子系统中的一组接口提供一个统一的接口。门面模式定义了一个高层接口，这个接口使得子系统更容易使用。
- **项目中的应用:** 业务逻辑层（Service Layer）的各个 Service 类（如 `StudyNoteService`, `FoodRecommendationService` 等）可以被视为对应模块的门面。
- 它们为表现层（Controller）提供了一组简化的、面向业务的接口（如 `createNote()`, `getAllRecommendations()`, `getWalkingRoute()` - 如果将路线规划逻辑放入 Service），隐藏了底层数据访问（Repository 调用）和外部服务调用（RestTemplate 调用）的复杂性。

- **单例模式 :**

- 即确保一个类只有一个实例，并提供一个全局访问点。
- **项目中的应用:** 在 Spring Boot 应用中，大部分组件（如 `@Service`, `@Repository`, `@RestController`, `@Component`, `@Configuration` 注解的类以及 `@Bean` 方法返回的对象）默认都是以单例模式由 Spring IoC 容器管理的。
- 例如，整个应用生命周期中通常只有一个 `StudyNoteService` 实例，被多个 Controller 共享使用。`RestTemplate` 也通常被配置为单例 Bean。

- **种子数据模式:**

- 即在应用程序启动时自动执行一些初始化任务，如填充数据库初始数据。
- **项目中的应用:**`*DataInitializer` 类（如 `StudyDataInitializer`, `FoodDataInitializer` 等）通过实现 `CommandLineRunner` 接口，是初始化器模式的应用。
- 它们在应用启动后运行特定的逻辑（检查数据是否存在，如果不存在则创建并保存示例数据）。

- **拦截器模式:**

- **核心思想:** 在一个方法或流程执行前、执行后或异常发生时插入额外的逻辑（如日志记录、性能监控、事务管理）。
- **项目中的应用:**`HibernateSqlInterceptor` 实现了 `StatementInspector` 接口，并被配置到 Hibernate 属性中（通过 `HibernateConfig` 的 `hibernatePropertiesCustomizer` Bean）。它允许在执行 SQL 语句前后进行拦截。尽管在提供的代码中 `inspect()` 方法只返回原始 SQL，没有添加额外的日志，但这体现了使用拦截器来插入行为的设计意图（例如，如果需要记录所有执行的 SQL）。结合 `LoggingFilter` 和 `SqlPatternLogFilter` 等，共同构成了对数据库访问日志进行控制的拦截/过滤机制。

好的，这是一份软件设计报告的“8. 未来工作”部分的撰写内容，列出了当前版本的已知限制和未来可能的改进方向。

5.未来工作

当前版本的《邮驿四方》Web 应用作为一个面向对象程序设计课程的大作业，已初步实现了衣、食、学、行四个核心模块的基本功能。

然而，作为一个实际应用，它仍然存在一些限制和不足，也为未来的功能增强和系统演进提供了广阔的空间。

下面是我认为可以在未来继续优化的地方：

- **用户系统与个性化:**

- 集成 Spring Security，实现用户注册、登录、认证和基于角色的权限控制。
- 将用户数据与登录用户关联，实现数据隔离和个性化服务。
- 基于用户行为（浏览历史、收藏等）实现个性化的内容推荐（美食、穿搭、行程）。

- **数据管理与内容平台化：**

- 开发后台管理界面，方便管理员或用户管理（增、删、改、查）美食、旅行行程和穿搭方案数据，无需修改代码或数据库。
- 集成图片上传服务，允许用户上传美食或穿搭图片。
- 学习笔记模块集成 Markdown 编辑器或富文本编辑器，支持更丰富的笔记格式。

- **核心模块功能深化：**

- **行程规划：**

- 允许用户自由输入起点和目的地进行路线规划。
- 增加路线保存功能，用户可以保存常用路线。
- 集成更多地图功能，如周边搜索、导航语音播报等。

- **穿搭推荐：**

- 引入更复杂的后端推荐算法，考虑季节、温度、天气现象、场合、风格、用户偏好、甚至用户上传的已有衣物信息进行智能匹配。
- 实现穿搭方案的评分和评论功能。

- **美食推荐：**

- 增加用户对美食的评论和评分功能。

- **学习笔记：**

- 支持笔记的分享或协作功能。

---

6. 开发与运行环境

本节将详细说明开发和运行《邮驿四方》Web应用所需的软硬件环境以及依赖关系。

6.1 开发环境

项目的开发和构建需要以下环境：

- **集成开发环境 ：**

- 推荐使用支持 Java 和 Spring Boot 开发的现代 IDE，例如 **IntelliJ IDEA**, **Eclipse** 或 **VS Code**
- 项目要求使用 **Java Development Kit (JDK) 版本 17 或更高版本**。

- **构建工具:**

- 本项目主要使用 **Apache Maven** 作为项目构建和管理工具。`pom.xml` 文件定义了项目的依赖、构建配置等信息。

- **依赖管理 (Dependency Management):**

- 项目的依赖由 **Maven** 通过 `pom.xml` 文件进行管理。所有外部库和框架（如 Spring Boot 各个 Starter, Spring Data JPA, Thymeleaf, H2, MySQL 连接器, 外部API调用相关的库等）都在 `pom.xml` 的 `<dependencies>` 节中声明。
- Maven 会自动下载和管理这些依赖。

6.2 运行环境

成功构建后，《邮驿四方》Web 应用可以在以下环境运行：

- **操作系统:**

- 由于 Spring Boot 应用被打包成可执行的 JAR 文件，其中包含了嵌入式 Web 服务器和所有依赖，因此对底层操作系统要求不高。
- 只要该操作系统支持运行兼容版本的 Java 运行时环境 (JRE)，即可运行。。

- **Java 运行时环境:**

- 应用运行时需要安装 **Java Runtime Environment (JRE) 版本 17 或更高版本**。

- **部署方式 :**

- 项目被配置为打包成一个 **可执行的 JAR 文件** 。这是 Spring Boot 的标准部署方式。
- 运行方式: 在命令行中使用 `java -jar your-application-name.jar` 命令即可直接运行应用程序

- **外部服务依赖:**

- **数据库:**

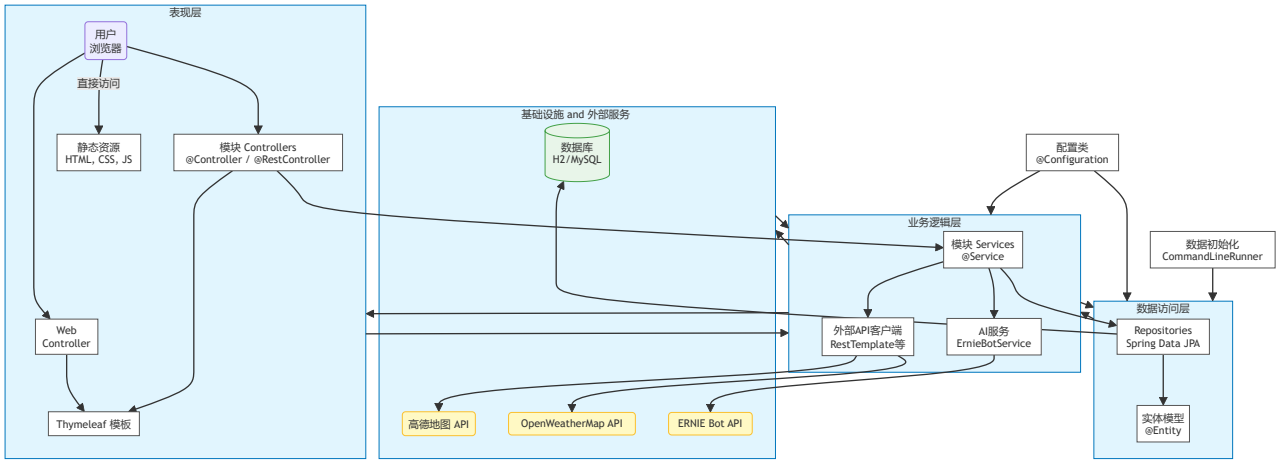
- 为了方便验收工作，开发和测试环境默认使用 **H2 内存数据库**，数据存储在内存中，应用停止后数据会丢失。无需额外安装数据库服务。
- 项目代码和配置也支持连接 **MySQL 数据库**。如果在 `application.properties` 中配置了 MySQL 连接信息并确保 MySQL 服务可用，应用可以连接外部 MySQL 数据库进行持久化存储。

- **外部 API 访问：**

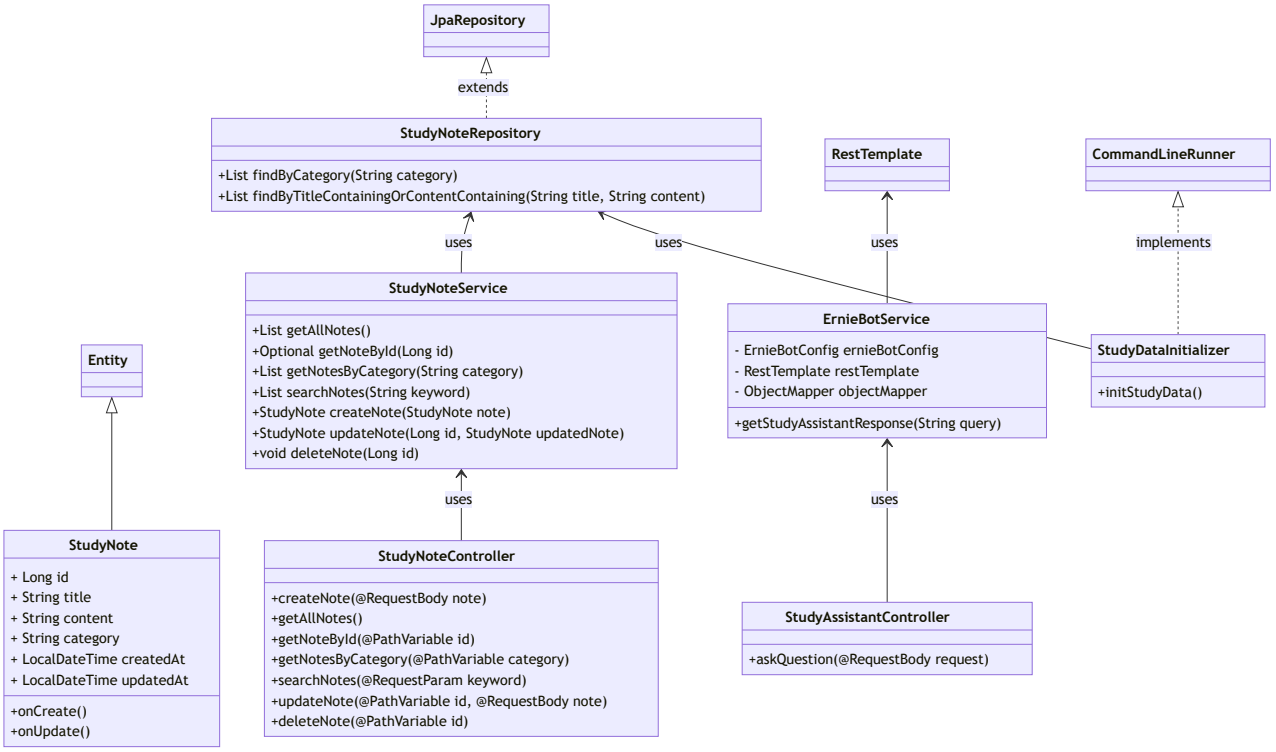
- 应用依赖对 **高德地图服务 (地理编码, 路线规划, 天气)** 和 **OpenWeatherMap 服务 (天气备用)** 的访问。
- 学习笔记模块依赖对 **ERNIE Bot API** （文心一言api）的访问。
- 运行环境需要能够访问互联网以连接这些外部 API 服务。
- 不用担心 我已经把key配置好了。

七、附录

系统架构图

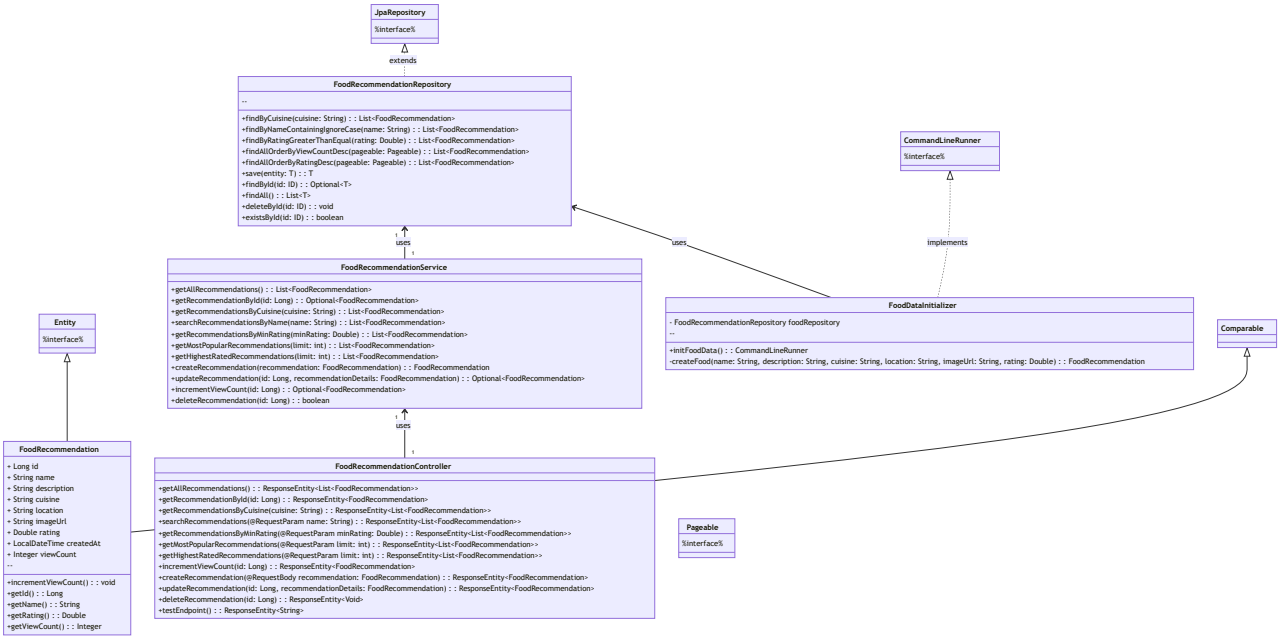


四大类图



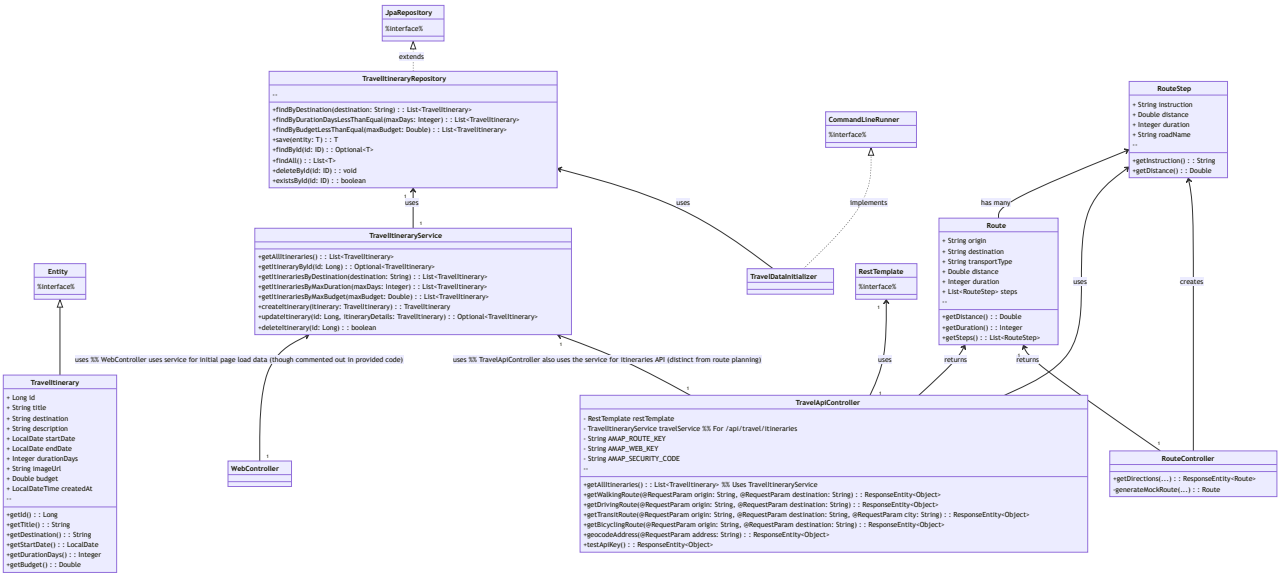
- **图 学习笔记模块类图说明：**

- 该图展示了学习笔记模块中的核心类及其相互依赖关系。
- `StudyNote` 是数据实体类，与数据库表直接对应。
- `StudyNoteRepository` 是数据访问接口，定义了与数据库交互的方法，`StudyNoteService` 依赖并调用这些方法。
- `StudyNoteController` 是表现层入口，依赖 `StudyNoteService` 完成业务逻辑。
- `StudyAssistantController` 依赖 `ErnieBotService` 与外部 AI 服务通信。
- `ErnieBotService` 依赖 `RestTemplate` 和 `ErnieBotConfig` 来构建和发送 API 请求。
- `StudyDataInitializer` 在应用启动时使用 `StudyNoteRepository` 填充初始数据。



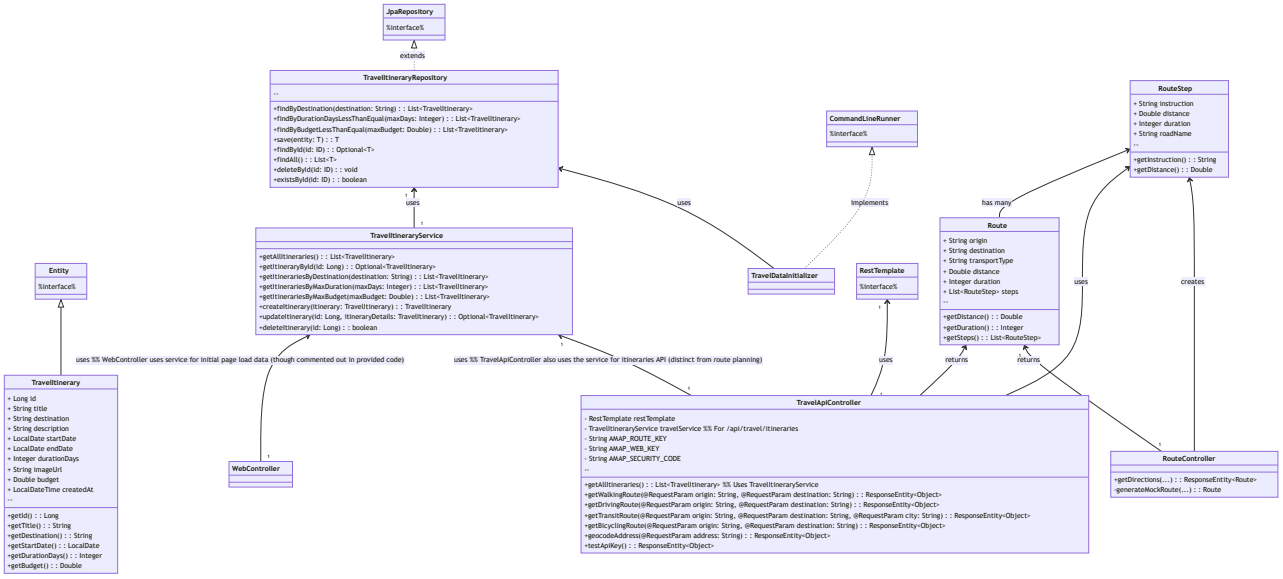
- 美食推荐模块类图说明：

- 该图展示了美食推荐模块中的核心类及其关系。
- `FoodRecommendation` 是数据实体。
- `FoodRecommendationRepository` 接口继承 `JpaRepository`，提供数据访问方法，并定义了多种查找美食的方法，包括基于属性的派生查询和基于 `@Query` 的复杂查询（如按浏览次数和评分排序）。
- `FoodRecommendationService` 依赖 `FoodRecommendationRepository` 实现业务逻辑，如获取推荐列表、筛选、搜索、统计浏览次数等。
- `FoodRecommendationController` 依赖 `FoodRecommendationService` 处理来自前端的 HTTP 请求，并将数据格式化为 JSON 返回。
- `FoodDataInitializer` 负责初始化示例美食数据。



- 行程规划模块类图说明：

- 该图展示了行程规划模块相关的类及其关系。
- `TravelItinerary` 和 `TravelItineraryRepository` / `Service` / `Controller` 负责预设行程方案的管理和 API 。
- 核心的路线规划功能由 `TravelApiController` 负责，它通过 `RestTemplate` 调用外部高德地图 API获取路线数据。
- `Route` 和 `Route.RouteStep` 是用于封装从高德 API 获取的路线结果的 DTO，在 `TravelApiController` 中被使用和返回。
- `TravelDataInitializer` 用于初始化预设行程方案数据。



- 类图说明：

- 该图展示了穿搭推荐模块中的核心类及其关系。
- `FashionOutfit` 是数据实体。
- `FashionOutfitRepository` 接口继承 `JpaRepository`，提供数据访问方法，并定义了按季节、场合、风格查找穿搭的方法。
- `FashionOutfitService` 依赖 `FashionOutfitRepository` 实现业务逻辑。
- `FashionOutfitController` 依赖 `FashionOutfitService` 处理来自前端的 HTTP 请求。
- `WeatherController` 独立负责天气信息的获取，作为外部 API 的客户端，依赖 `RestTemplate`。它包含根据描述判断天气 ID 以及从备用 API 获取数据的方法。
- `FashionDataInitializer` 负责初始化示例穿搭数据。