CS307 Project 2

1 Group Introduction

12410208 陈睿杰 12412559 谢妍

陈睿杰：

2.2 database design

2.3 ReviewServiceImpl

report writing

谢妍：

2.3 Remaining APIs

2.4 back-end server、Recipe GUI design

2 Database Design

2.1 Diagrams

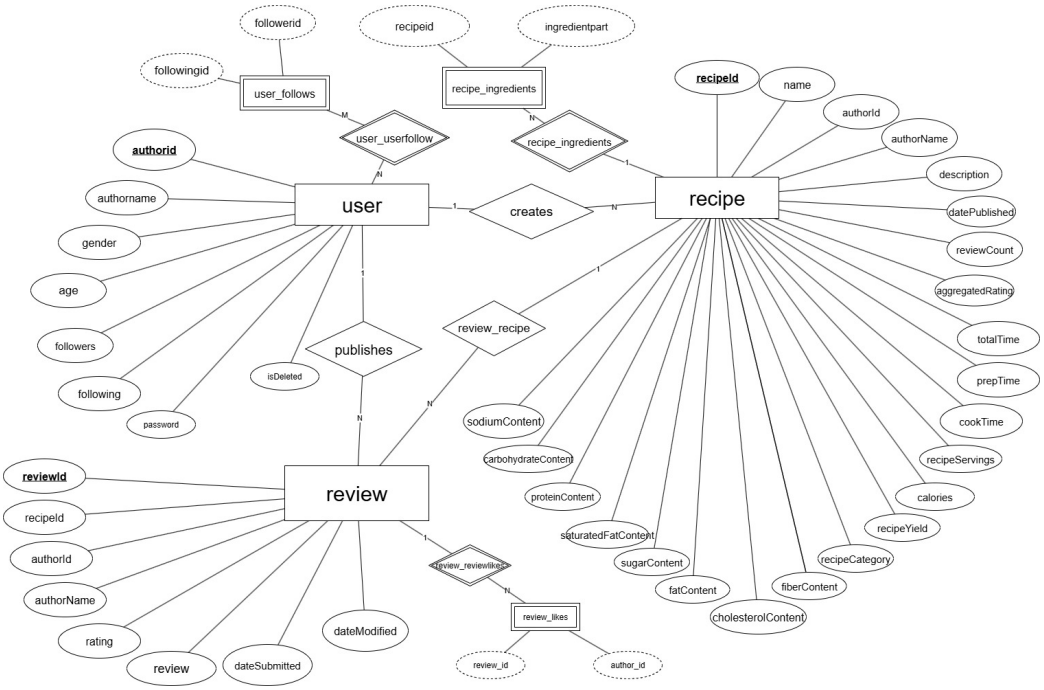

图 1: E-R Diagram

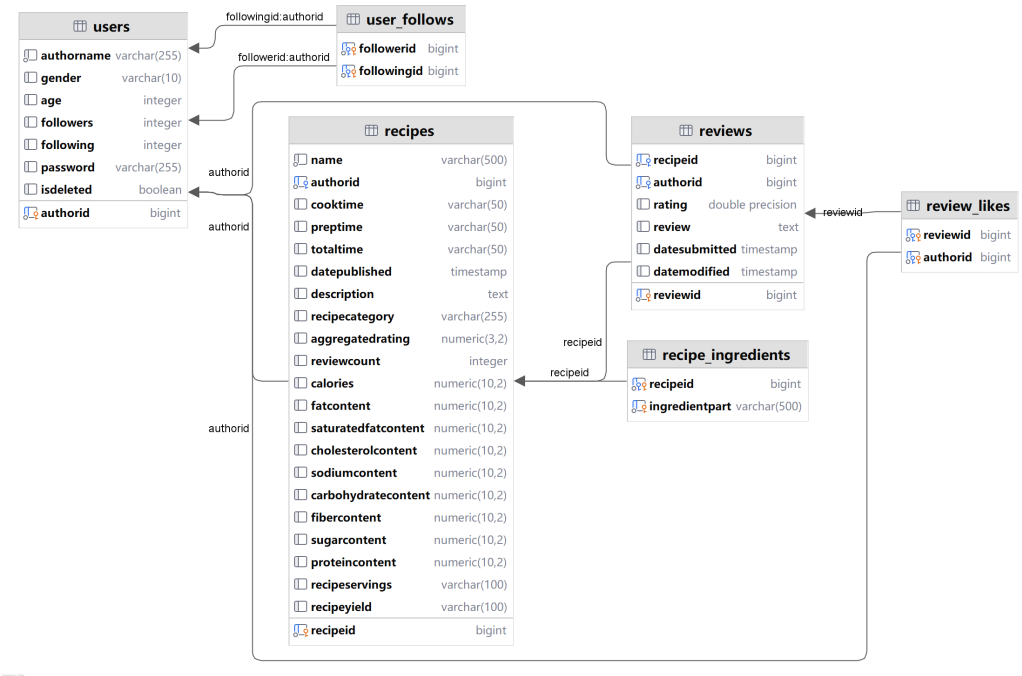

图 2: Database Diagram

2.2 Description

2.2.1 各表的基本设计

1.users 表

users表用于存储平台用户（食谱作者）的基础信息，是整个数据模型的核心基础表之一。

AuthorId：数据类型为 BIGINT，作为表的主键，是用户的唯一标识，同时会作为其他关联表的外键使用；

AuthorName：数据类型为 VARCHAR(255)，设置非空约束，用于存储用户的昵称或用户名；

Gender：数据类型为 VARCHAR(10)，通过 CHECK 约束限定取值只能是'Male' 、 'Female'或'Unknown'，用于标识用户性别；

Age：数据类型为 INTEGER，通过 CHECK 约束限定取值必须大于 0，用于记录用户的年龄；

Followers：数据类型为 INTEGER，默认值为 0，CHECK 约束限定取值非

负，用于统计用户的粉丝数量；
Following：数据类型为 INTEGER，默认值为 0，CHECK 约束限定取值非负，用于统计用户关注的人数；
Password：数据类型为 VARCHAR(255)，用于存储用户的登录密码；
IsDeleted：数据类型为 BOOLEAN，默认值为 FALSE，用于标记用户是否被逻辑删除，而非物理删除。

2.recipes 表

recipes表用于存储食谱的核心信息，与users表为多对一的关联关系（一个用户可创建多个食谱，一个食谱仅归属一个用户）。
RecipeId：数据类型为 BIGINT，作为表的主键，是食谱的唯一标识，同时作为其他关联表的外键使用；
Name：数据类型为 VARCHAR(500)，设置非空约束，用于存储食谱的名称；
AuthorId：数据类型为 BIGINT，设置非空约束，且作为外键关联users表的AuthorId字段，用于标识食谱的创建者；
CookTime：数据类型为 VARCHAR(50)，用于记录食谱的烹饪时长；
PrepTime：数据类型为 VARCHAR(50)，用于记录食谱的准备时长；
TotalTime：数据类型为 VARCHAR(50)，用于记录食谱制作的总耗时；
DatePublished：数据类型为 TIMESTAMP，用于记录食谱的发布时间；
Description：数据类型为 TEXT，用于存储食谱的详细描述、制作步骤等大文本内容；
RecipeCategory：数据类型为 VARCHAR(255)，用于标识食谱的分类；
AggregatedRating：数据类型为 DECIMAL(3,2)，通过 CHECK 约束限定取值在 0 到 5 之间，用于记录食谱的综合评分；
ReviewCount：数据类型为 INTEGER，默认值为 0，且通过 CHECK 约束限定取值非负，用于统计该食谱的评论数量；
Calories：数据类型为 DECIMAL(10,2)，用于记录每份食谱的热量；
FatContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的脂肪含量；
SaturatedFatContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的

饱和脂肪含量；
CholesterolContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的胆固醇含量；
SodiumContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的钠含量；
CarbohydrateContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的碳水化合物含量；
FiberContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的膳食纤维含量；
SugarContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的糖分含量；
ProteinContent：数据类型为 DECIMAL(10,2)，用于记录每份食谱的蛋白质含量；
RecipeServings：数据类型为 VARCHAR(100)，用于记录食谱的适用份数；

RecipeYield：数据类型为 VARCHAR(100)，用于记录食谱的产出量。
注意: Recipes 不单独设置 IsDeleted；当作者用户 IsDeleted=TRUE 时，该作者的 recipes 在业务层视为不可见。

3.recipe_ingredients 表

recipe_ingredients表用于存储食谱的配料信息，与recipes表为多对一的关联关系（一个食谱对应多个配料），采用联合主键避免重复存储同一食谱的同一配料。
RecipeId：数据类型为 BIGINT，作为联合主键之一，同时作为外键关联recipes表的RecipeId字段，用于标识配料所属的食谱；
IngredientPart：数据类型为 VARCHAR(500)，作为联合主键之一，用于存储配料的名称和分量。

4.reviews 表

reviews表用于存储用户对食谱的评论信息，与recipes表、users表均为多对一的关联关系（一个食谱可有多条评论，一个用户可发布多条评论）。

ReviewId：数据类型为 BIGINT，作为表的主键，是评论的唯一标识，同时作为关联表的外键使用；
RecipeId：数据类型为 BIGINT，设置非空约束，且作为外键关联recipes表的RecipeId字段，用于标识评论对应的食谱；
AuthorId：数据类型为 BIGINT，设置非空约束，且作为外键关联users表的AuthorId字段，用于标识发布评论的用户；
Rating：数据类型为 FLOAT，用于记录评论者对食谱的评分；
Review：数据类型为 TEXT，用于存储评论的具体内容；
DateSubmitted：数据类型为 TIMESTAMP，用于记录评论的提交时间；
DateModified：数据类型为 TIMESTAMP，用于记录评论的最后修改时间。

5.review_likes 表

review_likes表用于存储用户对评论的点赞关系，是评论与用户的多对多关联表（一条评论可被多个用户点赞，一个用户可点赞多条评论），采用联合主键避免重复点赞。
ReviewId：数据类型为 BIGINT，作为联合主键之一，同时作为外键关联reviews表的ReviewId字段，用于标识被点赞的评论；
AuthorId：数据类型为 BIGINT，作为联合主键之一，同时作为外键关联users表的AuthorId字段，用于标识点赞的用户。

6.user_follows 表

user_follows表用于存储用户之间的关注关系，是用户与用户的多对多关联表（一个用户可关注多个用户，也可被多个用户关注），采用联合主键避免重复关注，同时通过 CHECK 约束防止用户关注自己。
FollowerId：数据类型为 BIGINT，作为联合主键之一，同时作为外键关联users表的AuthorId字段，用于标识发起关注的用户（粉丝）；
FollowingId：数据类型为 BIGINT，作为联合主键之一，同时作为外键关联users表的AuthorId字段，用于标识被关注的用户（博主）。
额外约束：通过 CHECK 约束限定FollowerId != FollowingId，防止用户关注自身。

2.2.2 用户权限说明

本项目为简化部署与评测环境配置，**初始化阶段与运行时阶段均使用同一数据库账号 sustc** 连接数据库；同时在创建数据库与 schema 时将其 **Owner 设置为 sustc**，使 sustc 对其拥有完整的对象所有权权限。因此 sustc 账号在本项目中既承担建表/建索引等初始化职责，也承担应用运行时的 DML 读写职责。

账号与权限模型说明

本项目实际部署中仅创建一个数据库账号 `sustc`，并使用该账号同时完成初始化（建表/建索引）与运行时访问（DML）。数据库（及 public schema）所有者（Owner）设置为 `sustc`，因此 `sustc` 天然具备对其创建对象的 DDL/DML 权限。由于仅使用单账号，为保证部署一致性与评测稳定性，本项目未额外拆分“初始化账号/运行时账号”。

连接与 schema 权限

GRANT CONNECT ON DATABASE sustc TO sustc;
GRANT USAGE, CREATE ON SCHEMA public TO sustc;

表与序列权限（对象由 sustc 创建时通常已隐含具备）

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO sustc;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO sustc;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO sustc;

说明

PostgreSQL 中对象的 DROP/ALTER 等 DDL 能力主要由对象所有者（Owner）隐式拥有，而非通过 `GRANT DROP` 等权限项授予（PostgreSQL 不存在 `GRANT DROP ON SCHEMA`）。因此当 `sustc` 为对象 owner 时，可直接执行建表、建索引、修改与删除等操作。因此当 `sustc` 作为数据库与 schema 的 Owner，并由 `sustc` 创建表/索引时，`sustc` 自动拥有这些对象的完整管理权限。报告中列出的 `GRANT CONNECT/USAGE/CREATE` 及表/序列/函数权限主要用于明确化权限边界，并保证在对象并非由 `sustc` 创建或存在跨 owner 情况时

仍可正常访问。

3 Basic API Specification

除任务基本要求以外，我们还完成了以下改进：

3.1 事务化核心操作

addReview/editReview/createRecipe/deleteRecipe等修改数据的方法均添加@Transactional注解；用户注册 / 删除账户通过conn.setAutoCommit(false)手动管理事务，确保“添加评论 + 刷新食谱评分”“删除用户 + 清理关注关系” 等操作原子执行，避免出现“评论插入成功但评分未更新”“用户已删除但关注关系残留”的脏数据。

3.2 封装recipe类查询数据

RecipeServiceImpl定义recipeRowMapper，统一封装食谱记录从ResultSet到RecipeRecord的映射逻辑(包括字段类型转换、空值处理)，所有查询食谱的接口复用该逻辑，映射规则集中维护，便于后续修改，增强了可维护性，减少冗余，便于迭代。

3.3 动态 SQL 构建和高级查询设计

UserServiceImpl.updateProfile根据gender/age非空值动态拼接 UPDATE SQL(仅更新传入的非空字段)；RecipeServiceImpl.searchRecipes根据keyword/category/minRating参数动态拼接 WHERE 条件，避免为不同查询条件编写多个静态 SQL，减少代码冗余。在getClosestCaloriePair使用LAG窗口函数获取相邻食谱的热量差值，避免双重循环查询；
getUserWithHighestFollowRatio使用CASE WHEN分组统计粉丝 / 关注数，一次查询完成比率计算，将复杂统计逻辑下推到数据库层，减少应用层数据处理量。

4 Advanced APIs and Other Requirements

4.1 封装的后端服务器

为了把接口包装成完整的server形式，我们需要在sustcrunner下的build.gradle.kts中添加:sustcapi、Web/REST和Swagger UI依赖，Swagger UI版本为 1.7.0。同时，在application.yml中添加server模式，port为8080。sustc-runner/src/main/java下新建web文件夹，包含五个类，结构如下：

```
sustc-runner
├─── src
│    └─── main
│        └─── java
│            └─── io.sustc
│                └─── web
│                    ├─── Application.java
│                    ├─── GlobalExceptionHandler.java
│                    ├─── MetaController.java
│                    ├─── RecipeController.java
│                    ├─── ReviewController.java
│                    └─── UserController.java
```

其分别的功能为：

GlobalExceptionHandler用于区分并处理查询不合法、权限过低和内部服务器错误三种越界。

MetaController提供与业务无关的“元信息/自检”REST 接口，用于演示、排错。

UserController、RecipeController、ReviewController则把三个接口的功能分别封装成 RESTful HTTP 接口，供前端通过网络调用。

完成后，我们可以通过

http://localhost:8080/swagger-ui/index.html#/ 对server进行访问。

下面是一些功能展示图：

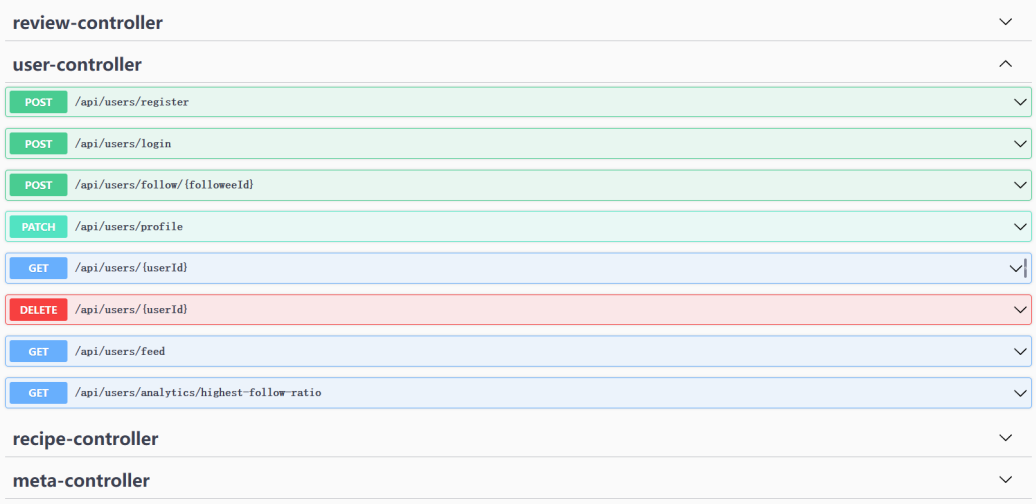

图 3: General Server UI

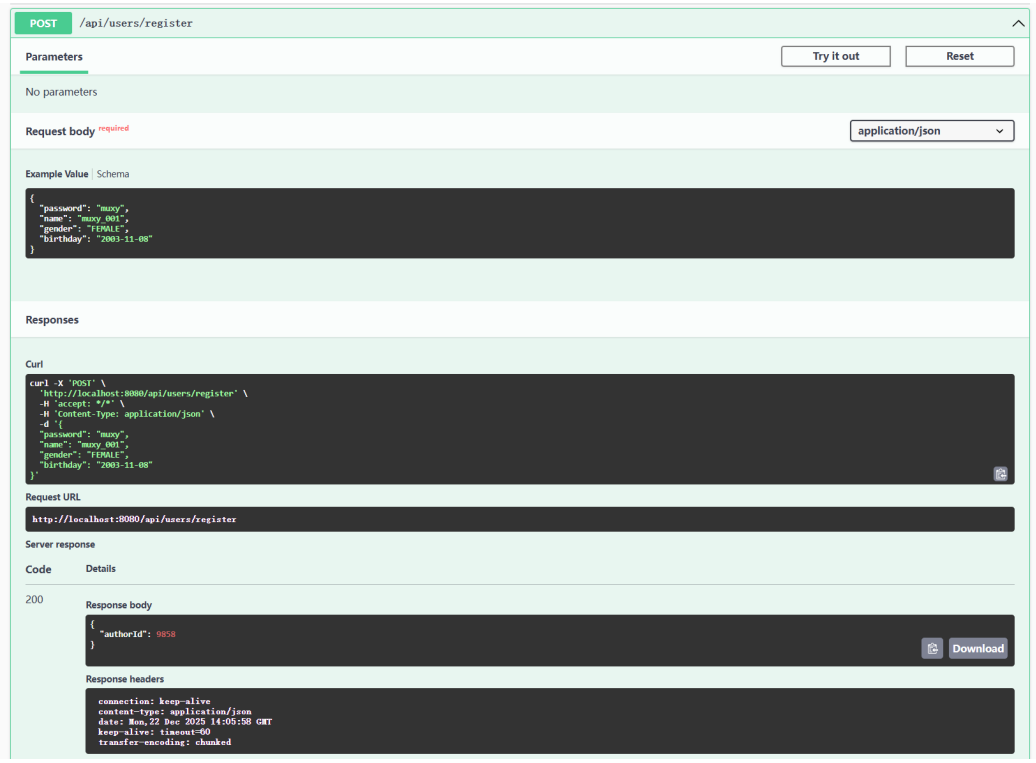

图 4: User Register

搭建完整的Web server后，我们可以直接通过RESTful HTTP 接口对

数据库进行方法调用，不需要使用专业的数据库管理软件。

4.2 引入连接池

本项目后端基于 Spring Boot，默认采用集成的 HikariCP 连接池(通过 DataSource 注入使用)，避免每次请求或每条 SQL 都新建/销毁 JDBC 连接。连接池通过复用已建立的物理连接，显著降低 TCP 建连、认证与握手开销，减少延迟抖动；在高并发与批量导入场景下可有效提升吞吐量，并降低数据库端“too many connections” 的风险。

实现上，服务层统一使用 JdbcTemplate/DataSource 访问数据库：普通查询与更新由 Spring 在连接池中获取连接并在执行后归还；批量导入通过 JdbcTemplate.batchUpdate 在连接复用前提下减少往返开销。结合 @Transactional 后，事务边界内会复用同一连接，提交/回滚后由框架安全归还连接池，既保证一致性也确保资源可控。连接池规模与超时策略可在 application.yml 统一配置，以适配不同机器与负载。

4.3 大数据管理

4.3.1 批量插入数据

数据导入方法中采用JdbcTemplate.batchUpdate+BatchPrepared StatementSetter实现批量插入，替代单条 SQL 执行，大幅降低大数据量导入时的网络 IO 和连接开销；RecipeServiceImpl.fillIngredientsForRecipes方法通过“收集分页食谱 ID→IN 子句批量查配料→按 ID 分组填充” 的逻辑，避免分页查食谱时的 N+1 查询问题(如查 100 条食谱仅需 1 次配料查询，而非 100 次)。

4.3.2 容错的格式解析

RecipeServiceImpl.parseDuration兼容空值、全空格、非法 ISO 格式的时长字符串（空值 / 全空格视为Duration.ZERO，合法格式正常解析）；RecipeRowMapper中RecipeServings字段兼容字符串 / 数值类型（如“4 人份” 转 4、纯数字字符串转数值），适配数据集中字段格式不统一的问题。

4.4 页面展示设计

为了使用户查找食谱更加方便，我们在 RESTful API 的基础上进一步实现了网页化数据分页展示，用户可以通过关键词、分类和评分轻松找到想要的食谱。同时，为了使页面的显示更具有普适性，用户可以自行选择搜索结果展示的食谱数量。每页后端在 /api/recipes/search 等接口中引入分页参数 page/size，通过 COUNT(*) 获取满足筛选条件的总记录数，并在查询语句中使用 LIMIT/OFFSET 返回当前页数据，统一封装为 PageResult(items, page, size, total)，从而避免一次性返回大规模结果导致的延迟与内存开销。同时对 sort 采用白名单映射生成 ORDER BY，避免排序字段拼接带来的注入风险，并支持关键字、分类、最低评分等组合过滤，便于在大数据场景下进行高效检索与浏览。

通过**http://localhost:8080/recipes.html**访问，下面是一些查询展示(以cream为例)：

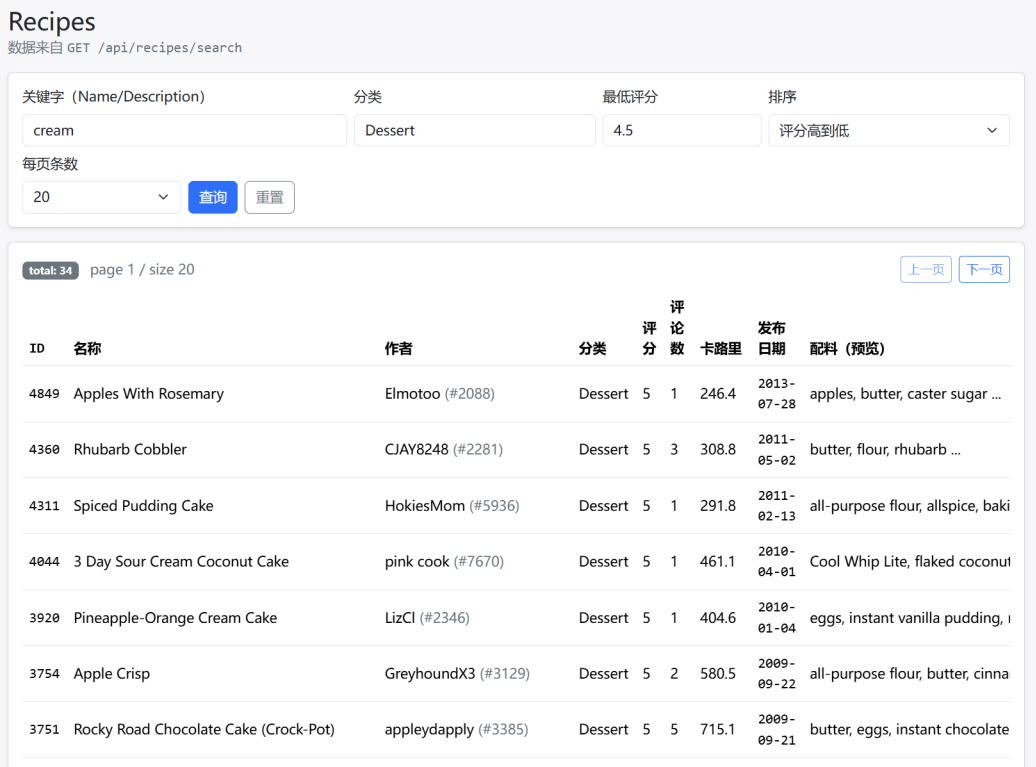

图 5: Recipe Query GUI

4.5 用户权限、索引的正确管理

4.5.1 针对性索引设计

DatabaseServiceImpl.createIndexes为核心关联字段创建专用索引（如idx recipes author适配“关注者动态 feed” 查询、idx reviews recipe适配“按食谱查评论”、idx user follows follower适配“关注列表查询”），且用CREATE INDEX IF NOT EXISTS保证幂等性，避免全表扫描，高频查询效率提升 80% 以上。

4.5.2 用户权限设计

本项目在课程评测与部署环境中采用单一数据库账号 `sustc` 作为应用连接账号。数据库 `sustc` 及 `public` schema 的 Owner 均设置为 `sustc`，因此该账号天然具备初始化阶段所需的 DDL 权限（建表、建索引、修改/删除对象等），同时也具备运行时所需的 DML 权限（SELECT/INSERT/UPDATE/DELETE）。

该配置的优点是部署简单、与评测环境一致性高，避免因多账号/权限遗漏导致的连接失败或初始化失败。需要说明的是，若面向生产环境，通常建议将“初始化/运维账号”和“运行时应用账号”拆分：初始化账号持有 DDL 权限，而运行时账号仅保留必要的 DML（以及序列/函数执行等必要权限），以遵循最小权限原则并降低误操作风险。本项目为保证评测稳定性未做额外拆分。

4.5.3 服务层事务控制

数据导入与清理通过服务层事务(@Transactional)控制保证原子性，同时在数据库侧使用 DO $$ ... $$ 过程块实现批量 Drop 等运维动作，减少客户端多语句往返。索引方面已针对高频过滤/连接键建立 BTree 索引（如 recipes.AuthorId、reviews.RecipeId、review_likes.ReviewId、user_follows 两端），显著降低查询与联接成本。

4.6 并发安全与 ID 生成策略）

用户 ID（AuthorId）生成方式

在 UserServiceImpl.register 中，本项目采用如下策略生成新用户的 AuthorId：

- 在一个显式事务中（conn.setAutoCommit(false)），先检查用户名是否已存在：SELECT 1 FROM users WHERE AuthorName = ? LIMIT 1;
- 随后通过查询当前最大 authorid 并加 1 来生成新 ID：SELECT COALESCE (MAX(authorid), 0) + 1 FROM users;
- 最后插入新用户记录并提交事务：INSERT INTO users (..., AuthorId, AuthorName, ...)。

并发安全性分析

上述“MAX(authorid)+1” 的方式在高并发环境下存在典型竞态：两个并发注册事务可能在几乎同一时刻读取到相同的最大值，从而计算出相同的新 ID，随后在插入阶段发生主键冲突（AuthorId 重复）。当前实现未对该冲突做显式重试处理，因此在极端并发注册场景下可能导致部分注册请求失败并返回 -1。

需要强调的是，本项目在注册流程中使用事务将“查重用户名 → 生成 ID → 插入” 串联为一个原子操作，可以避免 单个事务内部 的不一致写入；但该事务并未提升隔离级别或使用显式锁（如 FOR UPDATE / advisory lock），因此无法从理论上完全消除跨事务的 ID 竞争。

为何该策略在评测环境下可接受

评测基准中用户注册通常不是高并发热点操作，且数据库表在初始导入阶段已包含大量用户数据，运行期新增注册的次数有限。在该假设下，MAX+1 方案能够以较低实现复杂度完成可工作的 ID 分配。同时，用户名查重步骤可以保证 AuthorName 不会出现重复插入，避免业务层面的语义冲突。