详细设计规约文档

植保精灵(PGUARD)

宋宇然 陈晓坤 许经宝

**修订历史记录：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编写日期 | SEPG | 版本 | 说明 | 作者 |
| 2025/1/3 |  | v1.0 | 完成详细设计规约 | 宋宇然、许经宝 |
|  |  |  |  |  |

1. **引言**

**1.1编写目的**

本项目旨在通过移动端平台，通过先进的YOLO图像识别技术，全面辅助农民、农业研究人员及种植爱好者，高效完成农田作物与园艺作物的病虫害精准检测及地块智能监管。我们致力于打造一个低成本、易操作且功能强大的系统，为不同技术水平的用户提供即时、准确的诊断与建议。通过实时跟踪分析与可视化地块监管，帮助用户实现精准农业管理，优化种植效果，同时促进种植知识的交流与分享，共同推动农业领域的智能化、可持续发展。

**1.2背景与依据**

**1.2.1项目背景**

在当今时代，传统农业正逐步向智能化、精准化转型，以适应资源高效利用与可持续发展的需求。长期以来，农业生产多依赖于经验积累与人力投入，面对复杂多变的病虫害挑战，往往缺乏有效的即时诊断手段。与此同时，随着现代生活条件的不断提升，越来越多的人开始拥有自己的小块土地，用于满足个人的种植爱好和绿色生活的追求。然而，对于这部分种植爱好者而言，缺乏专业的农业知识和经验，常常在面对农作物病虫害等问题时感到束手无策。

在这个背景下，图像识别、传感器等前沿科技为农业种植领域带来了新的曙光。这些技术不仅能够帮助农民实现精准农业，提高生产效率，还能够为种植爱好者提供强有力的支持。通过运用图像识别算法和大模型技术，可以实现对农作物病虫害等问题的快速、准确识别，为种植者提供及时的解决方案。这类农作物识别软件，能够服务于农民、种植爱好者、科研人员和政府机构等多个群体，帮助他们快速识别农作物及其潜在问题，并提出科学合理的解决方案，从而推动农业领域的智能化、精准化发展。

**1.2.2竞品调研依据**

经过深入的竞品调研，我们发现现有同类型产品在满足用户需求方面均存在不同程度的缺陷。

首先，以PestNet(美国)为代表的开放社区软件，虽然提供了农业病虫害互助的平台，但诊断结果往往受限于社区用户的经验水平，存在较大的偏差风险。同时，由于问题的响应速度依赖于其他用户的在线时间和活跃度，因此无法保证即时性。此外，社区中的信息质量参差不齐，重复度高，使得有效信息难以快速查找。

还有如加拿大的FieldClimate应用，这类基于实时数据的分析软件，虽然结合了气象站、传感器网络和数据分析功能，为用户提供了更为专业的建议，但大部分功能依赖于特定厂家的传感器等硬件设备，导致初期投入成本高昂。此外，系统涉及多种传感器，要求农户具备一定的技术知识以完成设备的维护和校准，增加了使用难度。

最后就是移动端图像识别软件，德国公司PEAT GmbH的产品 Plantix作为一款成熟的图像识别软件，在检测植物病害方面表现出色，但遗憾的是目前不支持中文，存在阅读障碍。而与之对标的国内软件，如慧植农当家、识农、耘眼等，虽然能够利用图像识别技术检测植物病虫害，但每次使用都需要用户先选择作物再拍照识别，缺乏对于同一用户地块作物的跟踪记录功能，用户体验有待提升。

针对以上问题，我们的系统将基于YOLO图像识别技术，提供在线的即时识别功能。用户仅需用手机摄像即可获取准确的建议，操作简便易用。同时，系统还将可视化跟踪记录用户地块，更好地贴合用户需求。

**1.3参考资料**

1. 《软件工程: 体系结构、设计、编码》
2. 《Design Patterns: Elements of Reusable Object-Oriented Software》
3. 需求规约文档——植保精灵(PGUARD)
4. 需求分析规约文档——植保精灵(PGUARD)
5. 详细设计规约文档——植保精灵(PGURARD)
6. **系统软件体系结构**

**2.1软件功能子系统设计**

系统按照功能被划分为以下五个子系统，这是五个子系统的功能说，主要类和调用关系：

**2.1.1用户管理子系统**

**功能描述：**

提供用户信息管理功能，包括用户注册、用户登入以及套餐充值服务等功能。

**核心类：**

User：用户类，存储了用户信息，包括用户名、用户位置和总结使用次数等。

Package：套餐包类，存储了用户可以购买的套餐包信息，包括套餐包的价格、提供的总结次数和套餐包名称。

UserController：控制器类，接收请求，根据用户身份认证提供当前登入用户信息。

**调用关系图：**



**2.1.2地块管理子系统**

**功能描述：**

提供地块信息管理功能，包括地块的创建、更改和删除以及地块上种植作物等功能

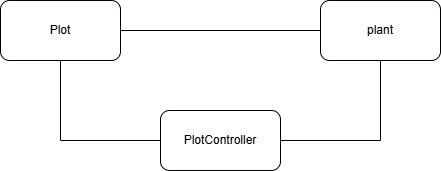
**核心类：**

Plot：地块类，存储地块的基本属性，包括地块名称、地块中的植物Id等。

Plant：植物类，存储植物的基本信息包括植物Id、植物名称和植物特点等。

PlotController：控制器类，处理地块管理业务逻辑以及获取地块信息和地块中植物信息。

**调用关系图：**



**2.1.3天气预告子系统**

**功能描述：**

提供基于地理位置获取对应地区的天气数据功能

**核心类：**

City：城市类，存储了城市名和该城市在外部天气系统中的编码号

WeatherController：控制器类，调用外部天气系统API并处理结果

**调用关系图：**



**2.1.4智能检测子系统**

**功能描述：**

提供病虫害检测功能，包括根据对应病虫害给出相应的防治信息建议功能。

**核心类：**

Disease：病虫害类，存储了病虫害的信息，包括疾病名称、对应防治建议等

DiseaseController：控制器类，进行病虫害检测以及获取病虫害对应的防治措施

**调用关系图：**



**2.1.5统计分析子系统**

**功能描述：**

提供记录用户对病虫害进行检测功能以及根据相应记录生成总计报告功能

**核心类：**

Log：日志类，存储检测日志信息，包括

LogController：控制器类，处理日志查询和根据日志获取总结报告

**调用关系图：**



2.2子系统关系调用综合图

**交互关系**

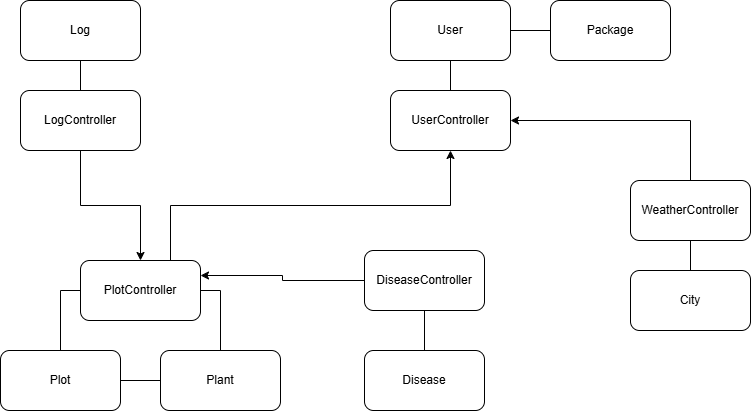
**1.用户管理子系统**

用户管理子系统和地块管理子系统以及天气预告子系统直接交互，用户可以通过这直接进行管理自己的地块信息，包括新增修改和删除地块等操作。同时可以直接查看该地区的天气情况。

**2.地块管理子系统**

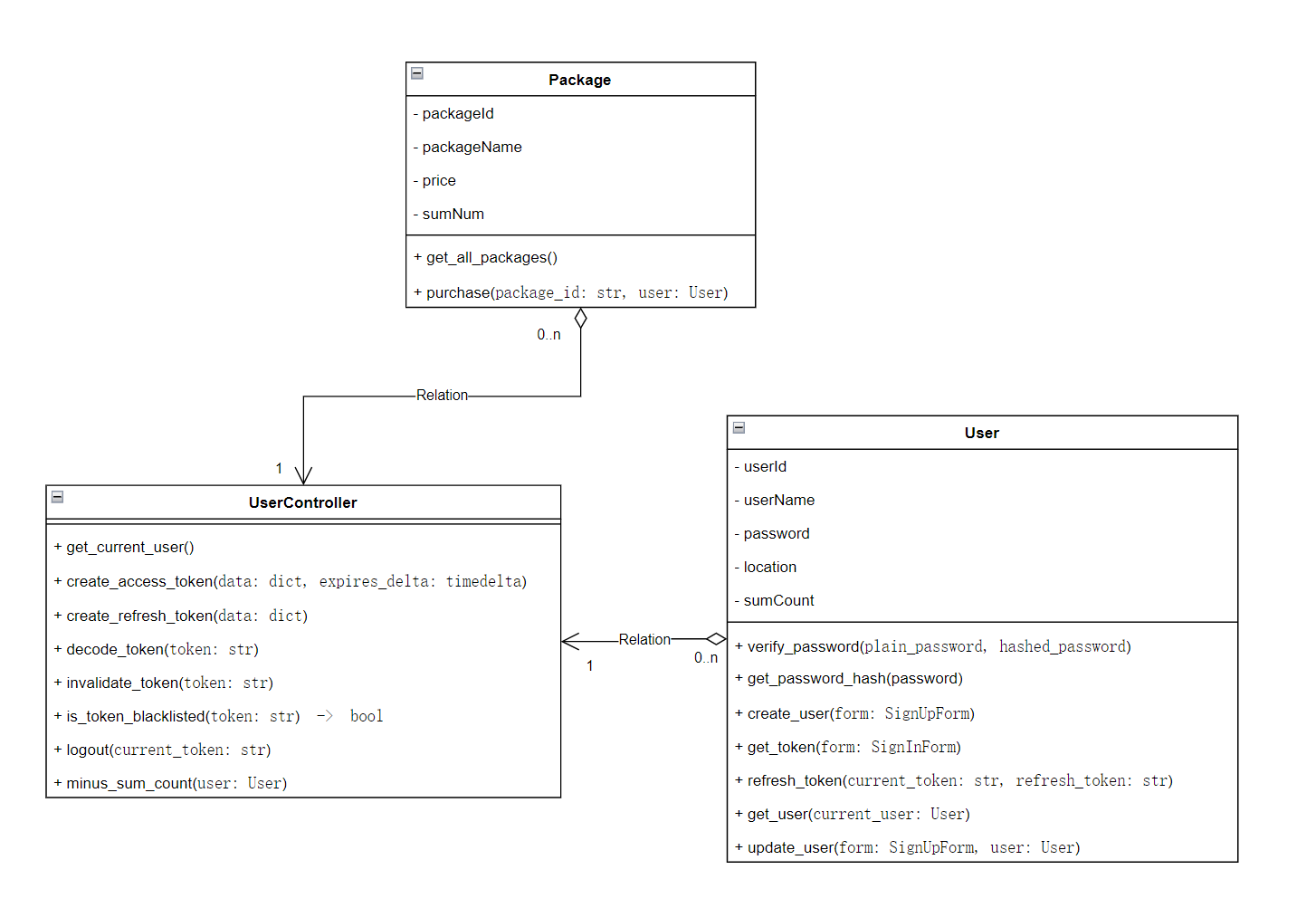
地块管理子系统与智能检测子系统和总结分析子系统直接交互，用户只能通过地块子系统与智能检测和统计分析进行交互，只有在进入地块后才能使用智能检测和获取统计分析的功能。

**交互图**



1. **子系统设计**

**3.1用户管理子系统**

****

**3.1.1子系统描述**

本系统主要用于管理用户信息以及实现充值服务，由三个实体类组成。UserManage是整个子系统的核心，负责管理用户注册和用户登陆状态，通过get\_current\_user这一依赖注入记录当前登录的用户；User类负责存储和修改用户信息；Package类记录套餐信息以及实现购买。

**3.1.2功能**

用户注册、用户登录、查看用户信息、更改用户信息、退出登录、套餐充值

**3.1.3与其他子系统接口**

**1.依赖项获取用户的接口get\_current\_user**

**功能描述:** 获取当前用户身份。

** 输入参数:**

 token: str: access令牌。

** 返回值:**

 user: User：用户实例。

**2.与Redis微服务的接口create\_access\_token**

 **功能描述:** 创建访问令牌。

** 输入参数:**

 data: dict: 令牌数据。

 expires\_delta: timedelta: 过期时间。

** 返回值:**

 str: 访问令牌。

**3. 与Redis微服务的接口create\_refresh\_token**

  **功能描述:** 创建刷新令牌。

** 输入参数:**

 data: dict: 令牌数据。

** 返回值:**

 str: 刷新令牌。

**4. 与Redis微服务的接口decode\_token**

 **功能描述:** 解码令牌。

** 输入参数:**

 token: str: 令牌。

** 返回值:**

 解码后的数据。

**5. 与Redis微服务的接口invalidate\_token**

 **功能描述:** 将令牌加入黑名单。

** 输入参数:**

 token: str: 令牌。

**6. 与Redis微服务的接口is\_token\_blacklisted**

 **功能描述:** 检查令牌是否在黑名单中。

** 输入参数:**

 token: str: 令牌。

** 返回值:**

 bool: 是否在黑名单中。

**7.与Redis微服务的接口logout**

 **功能描述:** 用户登出。

** 输入参数:**

 current\_token: str: 当前令牌。

** 返回值:**

 登出结果信息。

** 异常处理:**

 HTTPException: 如果令牌无效。

**8.与统计分析子系统的接口minus\_sum\_count**

 **功能描述:** 减少用户的使用次数。

** 输入参数:**

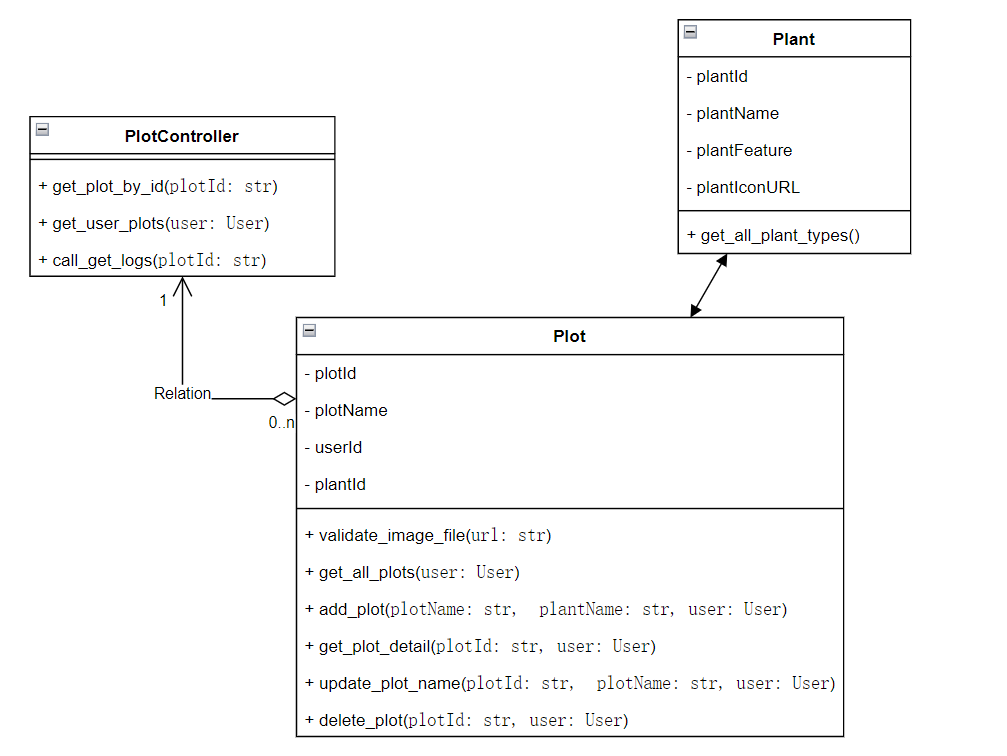
 user: User: 当前用户对象，通过依赖注入获取。

** 返回值:**

 bool: 是否成功减少次数。

 **异常处理:** 无。

**3.2地块管理子系统**

****

**3.2.1子系统描述**

Plot类存储了地块的一些基本信息，Plant类储存了系统提供可供检测的植物种类的信息。提供地块信息管理功能，包括地块的创建、更改和删除以及地块上种植作物等功能。

**3.2.2功能**

创建地块、地块可视化、查看地块信息、删除地块

**3.2.3与其他子系统接口**

**1.与智能检测的接口get\_plot\_by\_id**

 **功能描述**: 根据地块ID获取地块信息。

 **输入参数**:

 plotId: str: 地块的唯一标识符。

 **返回值**:

 plot: 地块对象。

 **异常处理**:

 HTTPException: 如果地块不存在。

**2.与用户管理的接口get\_user\_plots**

 **功能描述**: 获取用户的所有地块。

 **输入参数**:

 user: User: 当前用户对象，通过依赖注入获取。

 **返回值**:

 地块列表。

 **异常处理**:

 HTTPException: 如果获取地块失败。

**3.与地块管理的接口call\_get\_logs**

 **功能描述**: 获取地块的日志信息。

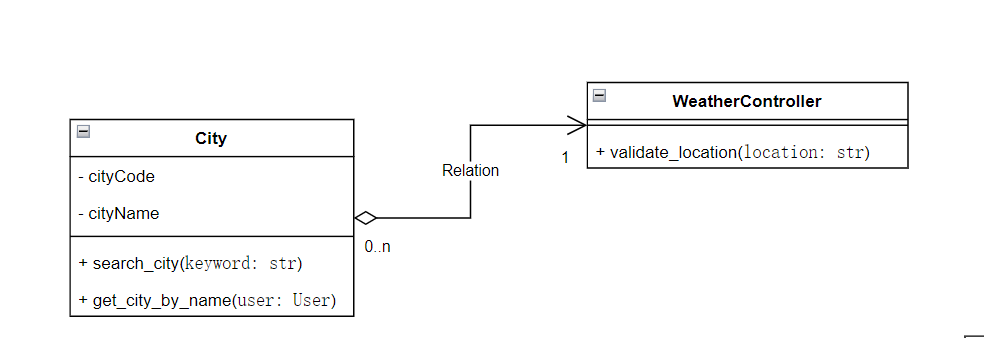
 **输入参数**:

 plotId: str: 地块的唯一标识符。

 **返回值**:

 日志列表。

**3.3 天气预告子系统**

****

**3.3.1子系统描述**

提供基于地理位置获取对应地区的天气数据功能

**3.3.2功能**

天气预报

**3.3.3与其他子系统接口**

**1.与用户管理的接口validate\_location**

 **功能描述**: 验证城市是否存在。

 **输入参数**:

 location: str: 城市名称。

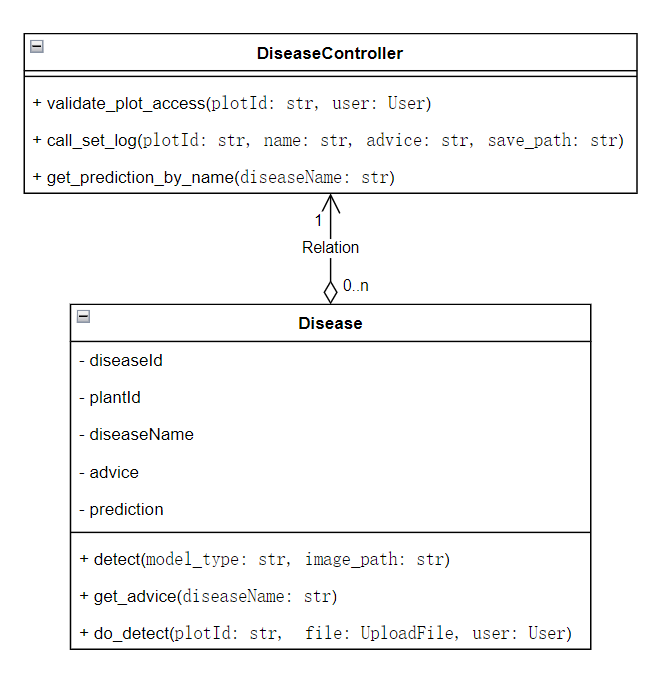
 **返回值**:

 bool: 是否存在。

 **异常处理**:

 ValueError: 如果城市不存在。

**3.4 智能检测子系统**

****

**3.4.1子系统描述**

本系统主要用于对地块上植物进行病虫害检测以及根据对应病虫害给出相应的建议，Disease类储存了病虫害的信息。DiseaseDetectControl类负责管理病虫害检测，包括从视频中截取图片、筛选有效截图、使用算法进行检测等。

**3.4.2功能**

病虫害检测、智能建议

**3.4.3与其他子系统接口**

**1.与地块管理子系统的接口validate\_plot\_access**

 **功能描述:** 验证用户是否有权限访问指定的地块。

 **输入参数:**

 plotId: str: 地块的唯一标识符。

 user: User: 当前用户对象，通过依赖注入获取。

** 返回值:**

 plot: 地块对象。

** 异常处理:**

 HTTPException: 如果用户无权访问地块或地块不存在。

**2.与统计分析子系统的接口call\_set\_log**

 **功能描述:** 调用日志记录功能。

** 输入参数:**

 plotId: str: 地块的唯一标识符。

 name: str: 日志名称。

 advice: str: 建议内容。

 save\_path: str: 保存路径。

** 返回值:**

 日志记录结果。

**3.与统计分析子系统的接口get\_prediction\_by\_name**

 **功能描述:** 根据疾病名称获取预测信息。

 **输入参数:**

 diseaseName: str: 疾病名称。

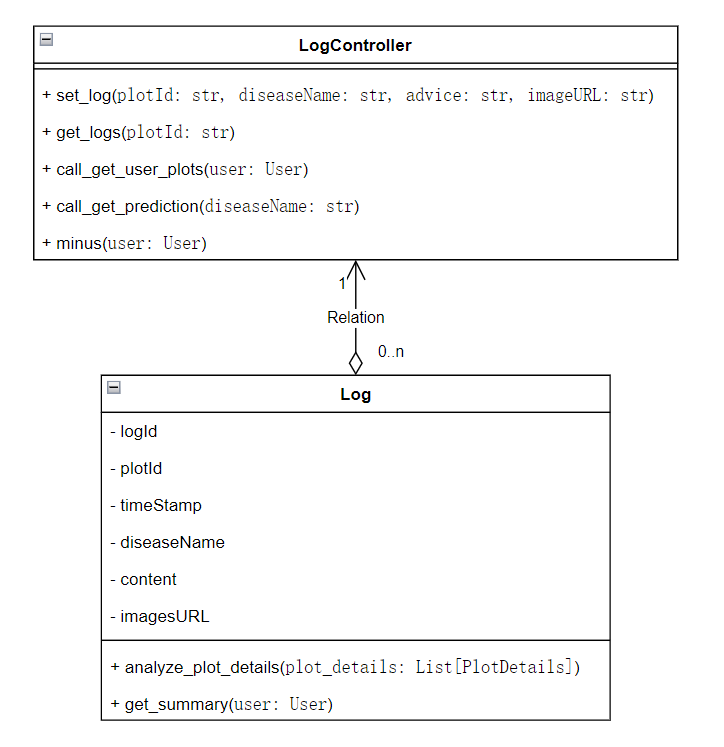
** 返回值:**

 prediction: 预测信息。

** 异常处理:**

 HTTPException: 如果疾病信息不存在。

**3.5 统计分析子系统**

****

**3.5.1子系统描述**

本系统主要用于根据以往的病虫害信息进行统计分析，给出相应的建议。Log类则是存储了以往所有的病虫害检测的日志内容，包括检测的时间、内容，检测图片等。LogControl类则是负责管理统计分析，其包含检测用户总结次数、扣除用户总结次数，返回总结内容等。

**3.5.2功能**

获取总结报告

**3.5.3与其他子系统接口**

**1.与智能检测的接口set\_log**

 **功能描述**: 创建新的日志记录。

 **输入参数**:

 plotId: str: 地块的唯一标识符。

 diseaseName: str: 疾病名称。

 advice: str: 建议内容。

 imageURL: str: 图片URL。

 **返回值**:

 创建日志的结果信息。

 **异常处理**:

 HTTPException: 如果日志创建失败。

**2.与地块管理的接口get\_logs**

 **功能描述**: 获取指定地块的所有日志。

 **输入参数**:

 plotId: str: 地块的唯一标识符。

 **返回值**:

 日志列表。

**3.与用户管理的接口minus**

 **功能描述**: 减少用户的使用次数。

 **输入参数**:

 user: User: 当前用户对象，通过依赖注入获取。

 **返回值**:

 bool: 是否成功减少次数。

1. **微服务设计**

**1.1微服务功能**

**1.Redis 微服务**

 **功能**:

 Redis在系统中主要用于缓存和会话管理。它可以存储用户会话信息、令牌黑名单等短期数据，以提高系统的响应速度和性能。

 在用户认证和令牌管理中，Redis用于存储和验证访问令牌的有效性。

**2.YOLOv8 模型微服务**

 **功能**:

 YOLOv8模型用于图像识别和疾病检测。通过分析上传的植物图片，模型可以识别出植物的病害类型，并返回相应的置信度。

 该模型是通过自主训练的方式，针对特定植物和病害进行优化，以提高识别准确性。

**1.2微服务交互接口**

**Redis 微服务交互接口**

 **接口1**: 获取登录用户信息

 **输入**: 访问令牌

 **输出**: 用户实例

 **接口2**: 令牌颁发get\_token

 **输入**: 用户登录表单

 **输出**: 访问令牌、刷新令牌

**接口3**: 令牌刷新refresh\_token

 **输入**: 访问令牌，刷新令牌

 **输出**: 返回新的访问令牌

**接口4**: 令牌黑名单管理is\_token\_blacklisted

 **输入**: 访问令牌

 **输出**: 令牌是否在黑名单里的bool值

**YOLOv8 模型微服务接口**

 **接口**: detect(model\_type: str, image\_path: str)

 **输入**:

 model\_type: 模型类型（如“Grape”或“Potato”）。

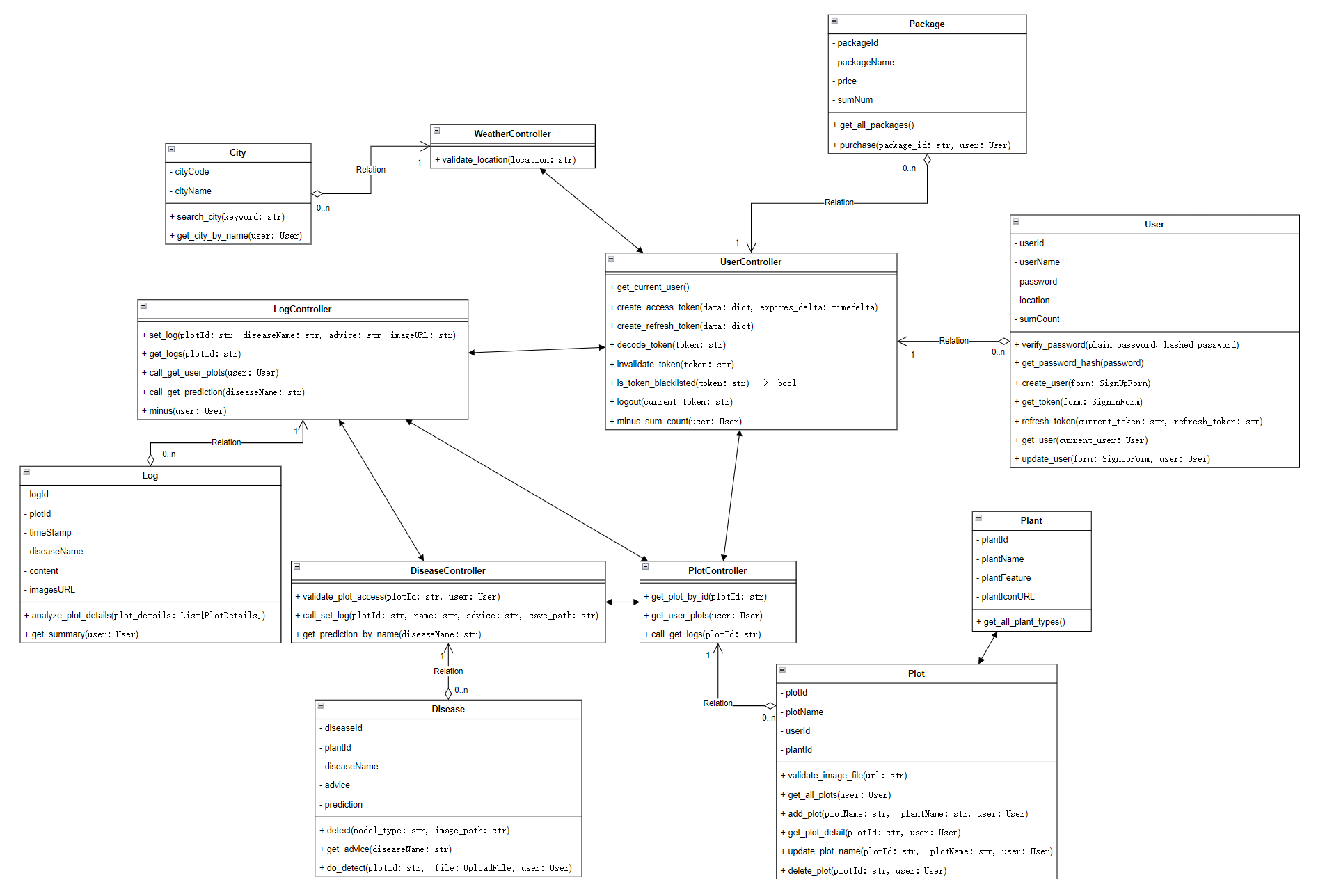
 image\_path: 图片文件路径。

 **输出**:

 detection\_result: 包含疾病名称和置信度的检测结果。

1. **类设计**

**1.1 设计类图**

****

**1.2 User类**

**1. 功能**

**User类的功能主要包括用户认证和管理相关的操作，如密码验证、令牌生成与管理、用户创建、用户信息获取与更新、用户登出等。**

**2. 输入输出接口**

** verify\_password(plain\_password, hashed\_password)**

** 输入:**

** plain\_password: 明文密码。**

** hashed\_password: 哈希密码。**

** 输出:**

** bool: 验证结果。**

** get\_password\_hash(password)**

** 输入:**

** password: 明文密码。**

** 输出:**

** str: 哈希后的密码。**

** create\_access\_token(data: dict, expires\_delta: timedelta = None)**

** 输入:**

** data: 令牌数据。**

** expires\_delta: 过期时间。**

** 输出:**

** str: 访问令牌。**

** create\_refresh\_token(data: dict)**

** 输入:**

** data: 令牌数据。**

** 输出:**

** str: 刷新令牌。**

** decode\_token(token: str)**

** 输入:**

** token: 令牌。**

** 输出:**

** dict: 解码后的数据。**

** invalidate\_token(token: str)**

** 输入:**

** token: 令牌。**

** 输出: 无。**

** is\_token\_blacklisted(token: str) -> bool**

** 输入:**

** token: 令牌。**

** 输出:**

** bool: 是否在黑名单中。**

** create\_user(form: SignUpForm)**

** 输入:**

** form: 注册表单。**

** 输出:**

** dict: 用户创建结果。**

** get\_token(form: SignInForm)**

** 输入:**

** form: 登录表单。**

** 输出:**

** dict: 包含访问令牌和刷新令牌。**

** refresh\_token(current\_token: str, refresh\_token: str)**

** 输入:**

** current\_token: 当前访问令牌。**

** refresh\_token: 刷新令牌。**

** 输出:**

** dict: 新的访问令牌。**

** get\_user(current\_user: User)**

** 输入:**

** current\_user: 当前用户对象。**

** 输出:**

** dict: 用户信息。**

** update\_user(form: SignUpForm, user: User)**

** 输入:**

** form: 更新表单。**

** user: 当前用户对象。**

** 输出:**

** dict: 更新后的用户信息。**

** logout(current\_token: str)**

** 输入:**

** current\_token: 当前访问令牌。**

** 输出:**

** dict: 登出结果。**

**3. 属性详细描述**

** SECRET\_KEY: 用于JWT加密的密钥。**

** ALGORITHM: JWT加密算法。**

** REFRESH\_TOKEN\_EXPIRE\_DAYS: 刷新令牌的过期天数。**

** oauth2\_scheme: OAuth2密码流的依赖项。**

** pwd\_context: 用于密码哈希和验证的上下文。**

**4. 方法详细描述及实现流程**

** verify\_password**

** 使用pwd\_context验证明文密码与哈希密码是否匹配。**

** get\_password\_hash**

** 使用pwd\_context对明文密码进行哈希处理。**

** create\_access\_token**

** 复制数据字典，设置过期时间，使用JWT加密生成访问令牌。**

** create\_refresh\_token**

** 复制数据字典，设置过期时间，使用JWT加密生成刷新令牌。**

** decode\_token**

** 使用JWT解码令牌，返回解码后的数据。**

** invalidate\_token**

** 解码令牌获取过期时间，计算剩余有效期，将令牌加入Redis黑名单。**

** is\_token\_blacklisted**

** 检查令牌是否在Redis黑名单中。**

** create\_user**

** 验证表单数据，创建新用户，返回创建结果。**

** get\_token**

** 验证用户信息，生成并返回访问令牌和刷新令牌。**

** refresh\_token**

** 验证刷新令牌和当前访问令牌，生成并返回新的访问令牌。**

** get\_user**

** 返回当前用户的信息。**

** update\_user**

** 验证表单数据，更新用户信息，返回更新结果。**

** logout**

** 将当前访问令牌加入黑名单，返回登出结果。**

**5. 限制条件及出错处理**

** 所有涉及数据库操作的函数都捕获异常并返回HTTP异常。**

** 令牌相关操作中，若令牌无效或过期，抛出HTTP异常。**

** 用户创建和更新时，若表单验证失败，抛出HTTP异常。**

**1.3 Package类**

**1. 功能**

**Package类的功能主要包括获取所有套餐信息和处理用户购买套餐的操作。**

**2. 输入输出接口**

** get\_all\_packages()**

** 输入: 无。**

** 输出:**

** List[dict]: 包含所有套餐信息的列表，每个字典包含套餐ID、名称、价格和次数。**

** purchase(package\_id: str, user: User)**

** 输入:**

** package\_id: 套餐的唯一标识符。**

** user: 当前用户对象。**

** 输出:**

** dict: 包含购买的套餐ID和用户更新后的次数。**

**3. 属性详细描述**

** Package: 数据库模型，表示套餐信息。**

** User: 数据库模型，表示用户信息。**

**4. 方法详细描述及实现流程**

** get\_all\_packages**

** 实现流程:**

**1. 异步获取所有Package对象。**

** 将每个Package对象转换为字典格式，包含packageId、packageName、price和sumNum。**

** 返回包含所有套餐信息的列表。**

** purchase**

** 实现流程:**

** 根据package\_id异步获取对应的Package对象。**

** 将Package对象的sumNum值加到用户的sumCount中。**

** 异步保存用户对象的更新。**

** 返回包含packageId和更新后的sumCount的字典。**

**5. 限制条件及出错处理**

** get\_all\_packages**

** 捕获所有异常，返回HTTP 500错误，附带错误信息。**

** purchase**

** 捕获所有异常，返回HTTP 500错误，附带错误信息。**

**1.4 Plot类**

**1. 功能**

**Plot类的功能主要包括地块的管理操作，如获取所有地块、添加新地块、获取地块详情、更新地块名称和删除地块。**

**2. 输入输出接口**

** validate\_image\_file(url: str)**

** 输入:**

** url: 图片的URL。**

** 输出:**

** str: 图片的资源路径或错误信息。**

** get\_all\_plots(user: User)**

** 输入:**

** user: 当前用户对象。**

** 输出:**

** List[dict]: 包含所有地块信息的列表。**

** add\_plot(plotName: str, plantName: str, user: User)**

** 输入:**

** plotName: 地块名称。**

** plantName: 植物名称。**

** user: 当前用户对象。**

** 输出:**

** dict: 新创建地块的信息。**

** get\_plot\_detail(plotId: str, user: User)**

** 输入:**

** plotId: 地块ID。**

** user: 当前用户对象。**

** 输出:**

** PlotDetails: 地块详情。**

** update\_plot\_name(plotId: str, plotName: str, user: User)**

** 输入:**

** plotId: 地块ID。**

** plotName: 新的地块名称。**

** user: 当前用户对象。**

** 输出:**

** dict: 更新后的地块信息。**

** delete\_plot(plotId: str, user: User)**

** 输入:**

** plotId: 地块ID。**

** user: 当前用户对象。**

** 输出:**

** dict: 删除成功信息。**

**3. 属性详细描述**

** RESOURCE\_PATH: 项目资源文件夹路径。**

** ALLOWED\_IMAGE\_TYPES: 允许的图片文件类型集合。**

**4. 方法详细描述及实现流程**

** validate\_image\_file**

** 实现流程:**

**1. 构建图片的完整路径。**

**2. 检查文件是否存在。**

** 检查文件扩展名是否合法。**

**4. 返回图片的资源路径或错误信息。**

** get\_all\_plots**

** 实现流程:**

**1. 获取用户的所有地块。**

**2. 验证每个地块的图片是否合法。**

**3. 构建并返回包含地块信息的列表。**

** add\_plot**

** 实现流程:**

** 验证植物是否存在。**

** 创建新地块。**

**3. 构建并返回新地块的信息。**

** get\_plot\_detail**

** 实现流程:**

** 验证地块ID格式。**

** 获取地块信息并预加载植物信息。**

** 验证图片是否合法。**

**4. 获取地块的所有日志。**

** 构建并返回地块详情。**

** update\_plot\_name**

** 实现流程:**

** 获取并验证地块。**

**2. 更新地块名称。**

** 保存更新并返回结果。**

** delete\_plot**

** 实现流程:**

** 获取并验证地块。**

** 删除地块。**

** 返回删除成功信息。**

**5. 限制条件及出错处理**

** validate\_image\_file**

** 返回错误信息字符串以指示图片不存在或不支持的扩展名。**

** get\_all\_plots**

** 捕获所有异常，返回HTTP 500错误，附带错误信息。**

** add\_plot**

** 捕获ValueError和DoesNotExist异常，返回HTTP 400或404错误。**

** 捕获其他异常，返回HTTP 500错误。**

** get\_plot\_detail**

** 捕获ValueError异常，返回HTTP 400错误。**

** 捕获其他异常，返回HTTP 500错误。**

** update\_plot\_name**

** 捕获ValueError异常，返回HTTP 400错误。**

** 捕获其他异常，返回HTTP 500错误。**

** delete\_plot**

** 捕获ValueError异常，返回HTTP 400错误。**

** 捕获其他异常，返回HTTP 500错误。**

**1.5 Plant类**

**1. 功能**

**Plant类的功能主要是获取所有植物类型的名称。**

**2. 输入输出接口**

** get\_all\_plant\_types()**

** 输入: 无。**

** 输出:**

** List[str]: 包含所有植物名称的列表。**

**3. 属性详细描述**

** Plant: 数据库模型，表示植物信息。**

**4. 方法详细描述及实现流程**

** get\_all\_plant\_types**

** 实现流程:**

** 异步获取所有Plant对象。**

** 提取每个Plant对象的plantName属性。**

** 返回包含所有植物名称的列表。**

**5. 限制条件及出错处理**

** 捕获所有异常，返回HTTP 500错误，附带错误信息。**

**1.6 City类**

**1. 功能**

**City类的功能主要包括根据关键字搜索城市和获取用户所在城市的信息。**

**2. 输入输出接口**

** search\_city(keyword: str)**

** 输入:**

** keyword: 搜索关键字。**

** 输出:**

** List[dict]: 匹配城市的列表，每个字典包含城市名称和城市代码。**

** get\_city\_by\_name(user: User)**

** 输入:**

** user: 当前用户对象。**

** 输出:**

** dict: 包含城市名称和城市代码。**

**3. 属性详细描述**

** City: 数据库模型，表示城市信息。**

** User: 数据库模型，表示用户信息。**

**4. 方法详细描述及实现流程**

** search\_city**

** 实现流程:**

** 根据keyword异步过滤City对象。**

** 如果没有匹配的城市，返回提示信息。**

** 构建并返回包含城市名称和代码的列表。**

** get\_city\_by\_name**

** 实现流程:**

** 根据用户的location异步获取City对象。**

** 如果城市不存在，返回提示信息。**

**3. 返回包含城市名称和代码的字典。**

**5. 限制条件及出错处理**

** search\_city**

** 捕获所有异常，返回HTTP 500错误，附带错误信息。**

** get\_city\_by\_name**

** 捕获所有异常，返回HTTP 500错误，附带错误信息。**

**1.7 Disease类**

**1. 功能**

**Disease类的功能主要包括疾病检测、获取疾病建议和处理上传的图片文件。它通过调用外部脚本进行疾病检测，并根据检测结果提供相应的建议。**

**2. 输入输出接口**

** detect(model\_type: str, image\_path: str)**

** 输入:**

** model\_type: 模型类型（如“Grape”或“Potato”）。**

** image\_path: 图片文件路径。**

** 输出:**

** dict: 包含疾病名称和置信度的检测结果。**

** get\_advice(diseaseName: str)**

** 输入:**

** diseaseName: 疾病名称。**

** 输出:**

** str: 疾病建议。**

** do\_detect(plotId: str, file: UploadFile, user: User)**

** 输入:**

** plotId: 地块ID。**

** file: 上传的图片文件。**

** user: 当前用户对象。**

** 输出:**

** dict: 包含疾病名称、建议、置信度和图片URL的检测结果。**

**3. 属性详细描述**

** ULTRALYTICS\_PATH: 外部脚本的路径。**

** UPLOAD\_PATH: 上传文件的存储路径。**

** PLANT\_NAME\_MAP: 植物名称映射。**

** DISEASE\_NAME\_MAP: 疾病名称映射。**

**4. 方法详细描述及实现流程**

** detect**

** 实现流程:**

** 根据model\_type选择对应的Python脚本。**

**2. 构建命令行调用参数。**

** 使用subprocess执行外部脚本。**

** 解析脚本输出，提取疾病名称和置信度。**

** 返回检测结果。**

** get\_advice**

** 实现流程:**

**1. 异步获取Disease对象。**

** 返回疾病的建议信息。**

** do\_detect**

** 实现流程:**

** 验证地块访问权限。**

** 获取并验证植物类型。**

**3. 检查文件扩展名是否合法。**

**4. 保存上传的图片文件。**

** 调用detect函数进行疾病检测。**

** 获取检测结果的建议。**

**7. 保存检测日志。**

** 返回检测结果。**

**5. 限制条件及出错处理**

** detect**

** 如果model\_type不支持，抛出ValueError。**

** 捕获并处理子进程执行中的错误。**

** get\_advice**

** 捕获所有异常，返回HTTP 404错误，附带错误信息。**

** do\_detect**

** 捕获HTTPException并重新抛出。**

** 捕获其他异常，返回HTTP 500错误，附带错误信息。**

**1.8 Log类**

**1. 功能**

**Log类的功能主要包括分析地块的详细信息以生成统计数据，以及获取用户的地块统计摘要。它通过分析地块的日志信息，提供关于植物种类、疾病发生情况等的统计数据。**

**2. 输入输出接口**

** analyze\_plot\_details(plot\_details: List[PlotDetails])**

** 输入:**

** plot\_details: 包含多个地块详细信息的列表。**

** 输出:**

** dict: 包含地块数量、植物地块数量、每月病害数量、植物病害数量、病害数量和预测信息的统计数据。**

** get\_summary(user: User)**

** 输入:**

** user: 当前用户对象。**

** 输出:**

** dict: 包含用户地块统计信息的字典。**

**3. 属性详细描述**

** PlotDetails: 数据结构，包含地块的详细信息。**

** DISEASE\_NAME\_RMAP: 疾病名称映射，用于将疾病标识符转换为人类可读的名称。**

**4. 方法详细描述及实现流程**

** analyze\_plot\_details**

** 实现流程:**

** 获取当前年份。**

** 计算地块总数。**

**3. 统计每种植物占用的地块数量。**

**4. 初始化每月病害数量、植物病害数量和病害数量的计数器。**

** 遍历每个地块的日志，跳过“健康”记录。**

** 解析日志时间戳，更新相应的计数器。**

**7. 找出发生次数最多的病害。**

** 获取最常见病害的预测信息。**

**9. 返回统计数据。**

** get\_summary**

** 实现流程:**

** 检查用户余额是否足够。**

**2. 获取用户的所有地块。**

**3. 如果没有地块，返回空的统计信息。**

** 构建PlotDetails列表。**

** 调用analyze\_plot\_details分析统计信息。**

** 返回统计信息。**

**5. 限制条件及出错处理**

** analyze\_plot\_details**

** 捕获并打印日期解析错误，继续处理其他日志。**

** 捕获所有异常，返回HTTP 500错误。**

** get\_summary**

** 检查用户余额，不足时返回HTTP 400错误。**

** 捕获所有异常，返回HTTP 500错误。**