概要设计规约文档

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 |
| 2025/1/2 |  | v1.0 | 完成概要设计 | 许经宝、  陈晓坤、  宋宇然 |
|  |  |  |  |  |

1. **引言**

**1.1概要设计依据**

1. 杜庆峰老师上课所讲的概要设计基本原则；

2. 需求分析规约文档——植保精灵(PGUARD)

**1.2参考资料**

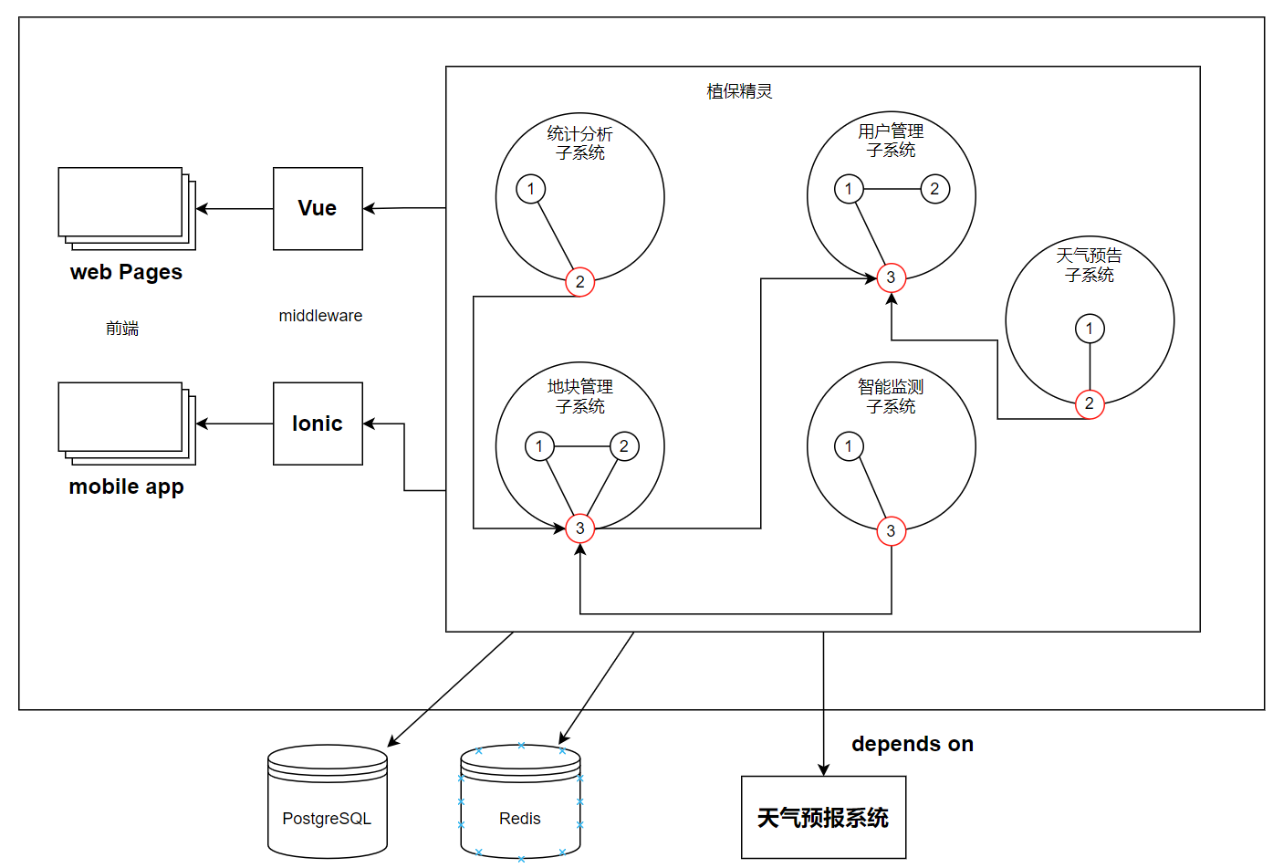
1. 《软件工程: 体系结构、设计、编码》
2. 《Design Patterns: Elements of Reusable Object-Oriented Software》
3. 需求规约文档——植保精灵(PGUARD)
4. 需求分析规约文档——植保精灵(PGUARD)

**1.3假定和约束**

本项目开发主要受软件工程课程的约束，因此本项目的假定和约束如下所示：

1. 项目开发期限为2个月，时间为2024年11月上旬~2025年1月上旬
2. 项目开发无经费，设备条件为3台个人电脑以及华为云服务器
3. 项目在开发前已对同领域软件耘眼的工作流程进行了充分的研究
4. 在开发过程中，每周周一、周四线下开组会，汇报工作进度，同时通过git和Github进行版本控制
5. **概要设计**

**2.1系统体系架构设计**



用户管理子系统：1：User类；2：Package类；3：UserController

地块管理子系统：1：Plot类；2：Plant类；3：PlotController

天气预告子系统：1：City类；2：WeatherController

智能检测子系统：1：Disease类；2：DiseaseController

统计分析子系统：1：Log类; 2：LogController

**2.2软件体系结构设计**

系统按照功能划分为以下五个子系统，以下是子系统的功能说明、主要类和调用关系：

**2.2.1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**



**2.2.2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

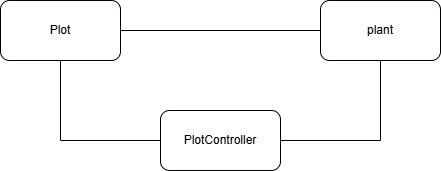
**核心类：**

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**



**2.2.3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**



**2.2.4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**



**2.2.5统计分析子系统**

**功能描述：**

记录用户病虫害检测日志，存储检测的时间和检测结果信息

**核心类及调用关系**

Log：日志类，存储检测日志

LogController：控制器类，处理日志查询和分析功能

**调用关系图**



**2.2.6类调用关系综合图**

**交互关系**

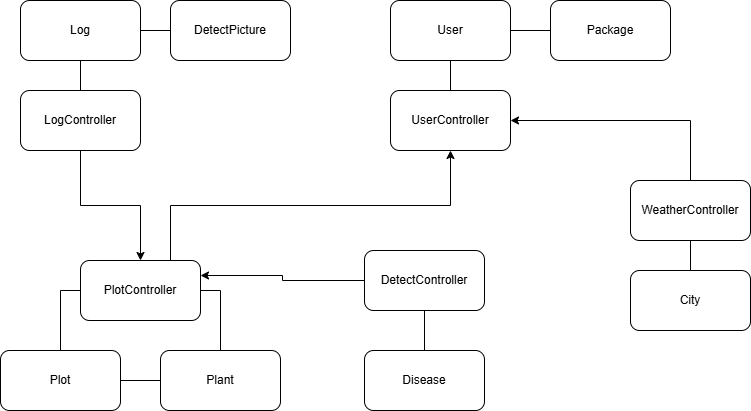
**1.用户管理子系统**

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**



**2.3接口设计**

**2.3.1 detectController**

**1.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.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.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。

**2.3.2 userController**

**1.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.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.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.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.invalidate\_token**

 **功能描述:** 将令牌加入黑名单。

** 输入参数:**

 token: str: 令牌。

**6.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.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.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。

**2.3.3 plotController**

**1.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.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.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。

**2.3.4 logController**

**1.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.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.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。

**2.3.5 weatherController**

**1.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。

**2.4界面设计**

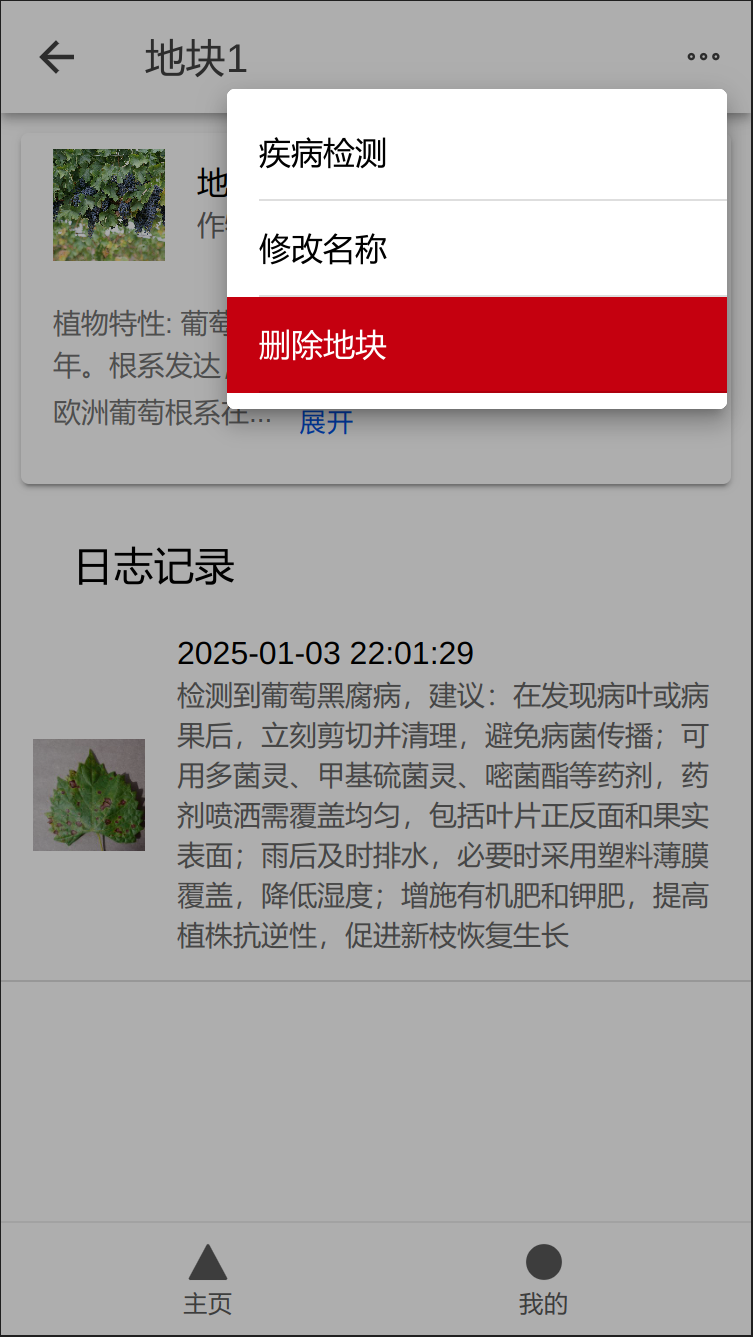
**2.4.1注册，登录界面设计**

**如图所示，用户进入登录界面后可以输入用户名和密码，进行登录。注册时需要用户输入用户密码与城市名称**

**2.4.2主页面设计**

**在用户登录以后，进入主页面。主页面最上方是搜索地块的搜索框，用地块名称和植物种类都可以进行搜索。随后是天气系统，这调用了高德天气第三方接口，会根据所在的城市获取当地的天气。右下方的浮动按钮可以新建地块**

**2.4.3 地块详情页**

**在该页面，我们可以看到所在地块种植植物的信息，已经历史检测记录。同时提供疾病检测、修改名称、删除地块的功能。**

**2.4.4个人信息页**

**在这个页面中，用户可以查看他的个人信息，同时也可以查看总结统计、进行套餐充值，更改个人信息，退出登录**

**2.5外部接口设计**

本项目使用了高德天体api

**2.5.1. 接口概述**

* **接口名称：**高德天气查询API
* **接口说明：**该接口提供实时天气信息查询，支持获取指定城市的实况天气或天气预报。
* **接口URL：**https://restapi.amap.com/v3/weather/weatherInfo
* **请求方式：**GET

**2.5.2. 请求参数**

| **参数名** | **含义** | **规则说明** | **是否必须** | **默认值** |
| --- | --- | --- | --- | --- |
| key | 请求服务权限标识 | 用户在高德地图官网申请的API Key | 是 | 无 |
| city | 城市编码 | 输入城市的adcode，详细参考城市编码表 | 是 | 无 |
| extensions | 气象类型 | 可选值：base（返回实况天气）、all（返回预报天气） | 否 | base |
| output | 返回格式 | 可选值：JSON、XML | 否 | JSON |

**2.5.3. 返回参数**

**2.5.3.1 状态信息**

| **参数名** | **含义** | **规则说明** |
| --- | --- | --- |
| status | 返回状态 | 1：成功；0：失败 |
| info | 返回状态信息 | 包含状态的描述信息 |
| infocode | 返回状态说明 | 10000 表示请求正确 |
| count | 返回结果总数 | 表示返回数据的数量 |

**2.5.3.2 预报天气**

| **参数名** | **含义** | **规则说明** |
| --- | --- | --- |
| province | 省份名称 | 返回的省份名称 |
| city | 城市名称 | 返回的城市名称 |
| adcode | 城市编码 | 返回的城市编码 |
| reporttime | 预报发布时间 | 天气预报的更新时间 |
| casts | 预报数据 | 包含当天、第二天和第三天的天气预报信息列表 |

**2.5.3.3 预报数据（casts）**

| **参数名** | **含义** | **规则说明** |
| --- | --- | --- |
| date | 日期 | 预报日期（例如：2025-01-03） |
| week | 星期几 | 预报的星期几（例如：星期五） |
| dayweather | 白天天气现象 | 白天的天气现象（例如：晴天） |
| nightweather | 晚上天气现象 | 晚上的天气现象（例如：多云） |
| daytemp | 白天温度 | 白天的温度（例如：25℃） |
| nighttemp | 晚上温度 | 晚上的温度（例如：18℃） |
| daywind | 白天风向 | 白天的风向（例如：东风） |
| nightwind | 晚上风向 | 晚上的风向（例如：西风） |
| daypower | 白天风力 | 白天的风力（例如：3级） |
| nightpower | 晚上风力 | 晚上的风力（例如：2级） |

**4. 接口示例**

**请求示例：**

https://restapi.amap.com/v3/weather/weatherInfo?city=110101&key=<your\_api\_key>&extensions=base&output=JSON

**返回示例（JSON格式）：**

{

"status": "1",

"info": "OK",

"infocode": "10000",

"lives": [

{

"province": "北京市",

"city": "北京市",

"adcode": "110101",

"weather": "晴",

"temperature": "25",

"winddirection": "东北",

"windpower": "3",

"humidity": "65",

"reporttime": "2025-01-03 10:00:00"

}

]

}

**5. 错误码说明**

| **错误码** | **含义** |
| --- | --- |
| 10000 | 请求成功 |
| 10001 | 请求失败 |
| 10002 | key无效或未授权 |
| 10003 | 请求过于频繁，请稍后再试 |

**2.6数据库设计**

**2.6.1数据库结构名称**

1. User (用户)
2. Package (套餐)
3. Plant (植物)
4. Plot (地块)
5. Log (地块日志)
6. Disease (病害)
7. City (城市)

**2.6.2标识符及数据项详情**

**User**

* userId: UUID, 主键, 默认值为uuid.uuid4()
* userName: 字符串, 最大长度40, 唯一
* password: 字符串, 最大长度100
* location: 字符串, 最大长度40
* sumCount: 小整数

**Package**

* packageId: UUID, 主键, 默认值为uuid.uuid4()
* packageName: 字符串, 最大长度40
* price: 浮点数
* sumNum: 小整数

**Plant**

* plantId: UUID, 主键, 默认值为uuid.uuid4()
* plantName: 字符串, 最大长度40, 唯一
* plantFeature: 文本字段
* plantIconURL: 字符串, 最大长度100

**Plot**

* plotId: UUID, 主键, 默认值为uuid.uuid4()
* plotName: 字符串, 最大长度40
* userId: 外键, 关联到User表的userId
* plantId: 外键, 关联到Plant表的plantId

**Log**

* logId: UUID, 主键, 默认值为uuid.uuid4()
* plotId: 外键, 关联到Plot表的plotId, 级联删除
* timeStamp: 日期时间, 自动添加当前时间
* diseaseName: 字符串, 最大长度40
* content: 文本字段
* imagesURL: 字符串, 最大长度200

**Disease**

* diseaseId: UUID, 主键, 默认值为uuid.uuid4()
* plantId: 外键, 关联到Plant表的plantId, 级联删除
* diseaseName: 字符串, 最大长度40
* advice: 文本字段
* prediction: 文本字段, 可为空

**City**

* cityCode: 字符串, 主键, 最大长度10
* cityName: 字符串, 最大长度40, 唯一

**2.6.3层次或表格相互关系**

* User与Plot之间是一对多的关系，一个用户可以拥有多个地块。
* Plant与Plot之间是一对多的关系，一种植物可以出现在多个地块中。
* Plant与Disease之间是一对多的关系，一种植物可能会有多种疾病。
* Plot与Log之间是一对多的关系，一个地块可以有多个日志记录。
* City没有直接与其他表建立关系，是独立的。

**2.7系统出错处理设计**

**2.7.1 出错信息**

|  |  |  |
| --- | --- | --- |
| 可能的错误 | 出错信息 | 处理方法 |
| 图像或视频上传格式不正确或损坏 | 图像格式错误或损坏，请上传有效的图像文件。 | 抛出异常，图像格式不正确或图像损坏，要求上传有效图像 |
| 数据库连接超时。 | 无法连接到数据库，系统正在维护中，请稍后再试。 | 抛出异常，无法连接到数据库，可能处于维护中。 |
| 图像质量不佳，无法进行准确的病虫害检测 | 图像质量过低，无法进行有效分析，请上传更清晰的图像 | 抛出异常，图像质量过低，请上传清晰的图像 |
| 存储空间不足，导致无法存储日志或模型数据 | 存储空间不足，无法保存日志或结果，请联系管理员 | 抛出异常，存储空间不足，无法保存日志或结果 |
| 网络请求超时或错误响应 | 外部服务响应错误，请检查网络或稍后再试 | 抛出异常，服务相应错误，请检查网络设置 |

**2.7.2补救措施**

**2.7.2.1输入数据错误**

在用户提交数据时，进行输入验证（例如，检查图像文件是否符合要求）。

如果输入无效，提供明确的错误提示信息并要求用户重新输入或修复数据。

**2.7.2.2数据库连接错误**

重新尝试连接数据库，设置合理的重试次数和重试间隔。

提供备份数据库或离线操作功能，确保在数据库不可用时系统仍能提供部分服务。

**2.7.2.3模型推理错误**

在上传图像时，增加图像质量提示，并尽量提供支持的图像格式。

提供图像预处理功能（例如降噪、增强对比度），提高模型识别准确性。

**2.7.2.4API调用错误**

对外部API调用增加重试机制，避免由于短暂的网络问题导致系统无法提供服务。

如果API调用连续失败，可以向用户提供替代方案（例如，显示历史数据或进行本地推理）。

**2.7.2.5系统资源不足错误**

采用负载均衡和资源调度策略，确保系统能够平稳处理高并发请求。

定期清理不必要的临时数据和过期日志文件，确保系统存储空间充足。

**2.7.3系统维护设计**

**2.7.3.1输入数据错误**

数据验证：在前端进行基本的输入验证，确保数据符合预期格式。后端进行进一步验证，确保输入的地块ID、病虫害名称和图像数据的合法性。

日志记录：记录每次无效输入的具体信息，包括时间、用户ID、输入数据、错误类型等，便于后续分析和改进。

**2.7.3.2数据库连接错误**

数据库监控：定期检查数据库连接状态，确保数据库可用。如果数据库出现异常，自动切换到备用数据库或通知运维人员。

日志记录：记录数据库连接失败的日志，包含错误时间、原因、重试次数等信息，便于后续分析和优化。

**2.7.3.3模型推理错误**

图像处理优化：定期优化图像处理算法，确保能够处理不同类型的图像数据。利用深度学习模型进行自动图像修复或补充。

模型监控与更新：监控模型的推理效果，当发现推理失败率过高时，启动模型重训练或微调，并确保使用最新版本的病虫害检测模型。

故障分析与报告：记录每次图像处理失败的日志，包括图像ID、错误类型、失败原因等，便于后续调查和修复。

**2.7.3.4API调用错误**

API监控：监控外部API的可用性，记录每次调用的响应时间、状态码等信息，确保外部服务可用。

异常通知：当API调用失败时，及时通知管理员或相关人员，并根据预设的规则进行自动处理或修复。

**2.7.3.5系统资源不足错误**

资源监控：使用资源监控工具（如 Prometheus、Grafana）实时监控系统的CPU、内存、磁盘空间使用情况，及时发现并处理资源瓶颈。

定期清理：定期清理过期的日志文件和临时数据，防止磁盘空间占满。使用数据归档或压缩存储较旧的历史数据。