需求分析

目标

健康是生命之基。互联网的普及，使得人们可以随时随地通过网络获取信息。因此为了保持健康，了解药品是必不可少的。与传统的方式相比，通过网络查询药品信息更加方便快捷，无需前往药店或医疗机构，节省了时间和精力。

因此，我们项目的目标是通过搜集药品百科网站上的信息，并对数据进行整理和归纳，创建一个新的网页平台，为用户提供方便、准确和易于搜索的药品相关信息。通过该平台，用户可以快速查询药品的适用症状、使用说明、适用人群等内容，以帮助他们做出更加明智的药品使用决策，从而帮助他们正确地购买药品。

功能需求

为了准确、全面、直观地向用户展示药品的信息，该项目有如下的功能需求：

1. 网页爬取：实现自动爬取药品百科网站的页面，并提取所需的药品信息，包括药品名称、适用症状、剂量、成分、禁忌、注意事项等。爬取的数据应该准确、完整，并能够及时更新。
2. 数据整合与存储：对爬取的药品信息进行整理和归纳，确保数据的准确性和一致性。将整理后的数据存储到数据库中，包括药品的基本信息、分类信息等。
3. 搜索功能：实现根据药品名称、用途、成分等关键词进行快速搜索的功能。用户可以输入关键词，系统应该能够根据关键词进行匹配，并返回相关的药品信息。
4. 药品详情展示：实现展示单个药品的详细信息页面。其中，详细信息包含药品的名称、适用症状、生产企业、注意事项等。用户可以通过点击药品列表或搜索结果中的药品链接来查看详细信息。
5. 分类和标签：对药品进行分类和标记，以便用户可以根据特定的药品分类或标签进行浏览和筛选。系统应该能够提供分类列表和标签云等功能，帮助用户快速定位和筛选药品。
6. 界面设计和用户体验：设计直观、美观的用户界面，提供良好的用户体验。界面应该简洁明了，操作流程应该简单易懂，以便用户快速找到所需的信息和功能。
7. 安全性和隐私保护：确保系统的安全性和用户隐私的保护。采取必要的安全措施，防止恶意访问和数据泄露。例如，使用加密技术保护敏感数据的存储，在数据传输过程中采用安全的通信协议（如HTTPS）等。
8. 反馈和报告：为用户提供反馈和报告的渠道，使用户能够报告网站上的问题、提供建议或投诉。系统应该能够及时收集和处理用户反馈，并提供适当的回复和解决方案。
9. 系统管理和监控：提供系统管理和监控功能，以便管理员能够监视系统的运行状态、处理异常情况和进行必要的维护操作。
10. 移动端适配：针对移动设备进行响应式设计，使得系统在不同尺寸的屏幕上能够提供良好的显示效果和用户体验。用户可以通过移动设备（如手机或平板电脑）访问系统，并获得与桌面端类似的功能和内容。

性能需求

对于Web端的应用，我们应该注重与用户交互的实时性，因此，性能需求成了必不可少的一项。本项目的性能需求列举如下：

1. 响应时间：系统应该能够在合理的时间范围内响应用户请求，对于一般的页面访问或搜索请求，响应时间应该在几秒钟内完成。

2. 并发处理：系统应该能够处理多个并发请求，同时保持较高的性能水平，以确保在高峰期间仍然能够提供稳定的服务。

3. 数据爬取速度：系统应该能够高效地从药品百科网站上爬取数据。爬取速度应该足够快，以确保能够及时获取最新的药品信息，并在用户请求时提供更新的数据。

4. 数据存储和检索：系统应该能够高效地存储和检索大量的药品信息。数据库设计和查询性能应该能够满足用户的需求，并且在数据量增加时仍然能够保持良好的性能。

5. 扩展性：系统应该具备良好的扩展性，以便在需要时能够轻松地添加更多的爬取任务、增加服务器资源或进行性能优化，以满足未来的需求。

可维护性需求

软件工程不仅需要完成基本功能需求，还要有较强的可维护性，以便长久运营和发展。本项目的可维护性需求列举如下：

1. 模块化设计：系统应该采用模块化的设计和开发方法，以便容易理解、维护和修改。各个模块应该具有清晰的职责和接口，并且模块之间的依赖应该尽可能减少，以降低修改一个模块对其他模块的影响。

2. 可读性和文档化：系统的代码应该具有良好的可读性，使用清晰的命名、注释和文档，以便其他开发人员能够理解代码的意图和功能。此外，系统的架构和设计决策也应该进行适当的文档化，以便于未来的维护和改进。

3. 错误处理和日志记录：系统应该具备良好的错误处理机制，能够捕获和处理潜在的错误情况，并提供适当的错误提示或回退机制。同时，系统应该能够生成详细的日志记录，记录关键操作和错误信息，以便于故障排查和维护。

4. 测试和自动化：系统应该进行充分的单元测试、集成测试和系统测试，以确保各个模块的正确性和稳定性。同时，建立自动化测试框架和持续集成环境，可以帮助快速发现和修复潜在的问题，并提高开发效率和可维护性。

5. 版本控制和代码管理：系统的代码应该使用版本控制系统（如Git）进行管理，以便跟踪和管理代码的变更。合适的分支管理策略和代码合并流程可以帮助团队成员更好地协作和维护代码库。

1. 定期维护和更新：系统应该进行定期的维护和更新，包括修复漏洞、升级依赖库和优化性能等。定期的维护可以保持系统的稳定性和安全性，并及时适应新的技术发展和用户需求变化。
2. 可扩展性和可重用性：系统的设计和代码应该具备良好的可扩展性和可重用性，以便在将来的开发过程中能够方便地添加新功能或模块。重复利用已有的代码和组件可以减少开发工作量和降低维护成本。
3. 文档和知识共享：系统的设计和实现应该进行适当的文档记录，包括架构设计、接口说明和技术文档等。同时，建立知识共享和团队交流的机制，可以促进团队成员之间的知识传递和技能提升，提高系统的可维护性和稳定性。