曲阜師範大學

本科生毕业论文（设计）

题　　目	基于 React 和 Django 的 现代化仓库管理系统设计与实现
姓　　名	林德松　　学号 2020414327
院　　系	网络空间安全学院
专　　业	软件工程
指导教师	李霞　　职称　讲师

2024 年 5 月 15 日

曲阜师范大学教务处制

目　录

基于 React 和 Django 的现代化仓库管理系统设计与实现

软件工程专业学生　林德松
指导教师　李霞

摘要：随着商业竞争的激烈和数字化转型的推进，现代企业对于高效的仓库管理系统需求增长。本研究和设计基于 Django 和 React 技术，旨在设计并实现一套现代化仓库管理系统。通过结合强大的后端逻辑和现代的前端用户界面，系统旨在提高企业运营效率、降低成本，并促进数字化转型。这套系统的设计旨在整合强大的后端逻辑和现代的前端用户界面，以满足企业在运营管理方面的多样化需求。通过精心设计的后端逻辑，系统能够实现仓库货物的快速、准确地入库、出库和移库，有效管理库存数量和状态，以及实现对供应链的可视化跟踪和管理。

关键词：仓库管理系统　　Django　React

The Design and Implementation of a Modern Warehouse Management System based on React and Django

Student majoring in Software Engineer　　Lin Desong
Tutor　Li Xia

Abstract: With the intensification of commercial competition and the advancement of digital transformation, modern enterprises are experiencing a growing demand for efficient warehouse management systems. This research and design project is based on Django and React technologies, aiming to design and implement a modernized warehouse management system. By integrating powerful backend logic with a modern frontend user interface, the system is designed to enhance operational efficiency, reduce costs, and facilitate digital transformation for businesses. The design of this system aims to integrate robust backend logic with a contemporary frontend user interface to meet the diverse needs of businesses in operational management. Through meticulously designed backend logic, the system is capable of rapidly and accurately handling the warehousing, outbound shipping, and transferring of goods. It effectively manages inventory quantities and status and implements visual tracking and management of the supply chain. This integration not only ensures the smooth and efficient operation of warehouse activities but also provides businesses with real-time data analysis and decision-making tools. The system's design emphasizes scalability and security, ensuring that it can adapt to growing business needs and protect sensitive data against potential threats. Overall, this warehouse management system stands as a testament to how technology can be leveraged to significantly improve the operational dynamics of modern enterprises. By embracing digital tools and methodologies, businesses can position themselves at the forefront of efficiency and innovation in the digital age.

Key words: Warehouse Management System　　Django　　React

1 引言

1.1 研究背景与意义

仓库作为企业运营的重要环节，直接影响到供应链的顺畅、成本的控制以及客户满意度的提升。随着商业竞争的日益激烈和数字化转型，现代企业对于高效的仓库管理系统的需求日益增长。在过去的几十年里，随着信息技术的发展，传统的仓库管理方式已经不能满足现代企业的需求。传统的手工记录和人工管理已经逐渐被自动化、数字化的管理方式所取代。尤其是在大规模生产和物流运输的背景下，仓库管理系统不仅需要高效地追踪货物的流动，更需要及时地提供数据分析和决策支持。在众多的 Web 开发框架中，Django/Python——一款开源的 Web 开发框架以其优秀的性能和快捷的开发正越来越引起人们的重视[1]。

基于上述背景，本研究旨在设计并实现一套基于 Django 和 React 的现代化仓库管理系统，加入安全高效的 Rust 语言来优化性能，通过结合 Django 强大的后端逻辑和 React 现代的前端用户界面，打造一套高度可扩展、灵活且用户友好的系统。该系统将注重用户体验和系统的性能优化，以满足现代企业对于仓库管理的多样化需求。

1.2 国内外研究现状

现如今在我国国内，出现了许多快速崛起的企业和公司，它们的规模和类型各异，但大多属于创新型技术的企业，并且在不断地进行研发和推出新的产品。企业生产的这些产品，进入市场后进一步扩大了企业的规模，增加了产品种类，形成了一个循环往复的过程。随着产品种类的增加和信息量的增长，对仓库和仓库管理系统的要求也相应提高[2]。仓库管理的首要目标是提升企业的盈利能力。为实现这一目标，确保销售信息及时、准确、高效地传达给管理人员至关重要。大型仓库管理软件的强大功能是其优势之一，但这也意味着对程序的可靠性要求极高。因为在大型仓库环境中，即使是小的系统故障也可能导致严重的操作中断和损失。因此，这类软件的开发和维护需要严格符合编程规范，确保系统的稳定性和可靠性。先进的仓库管理系统象征着企业的高度先进化。这类系统采用世界领先的 IT 技术和先进的管理手段，实现对仓库运营的精准控制和高效管理。借助专业化的物流经验，这些系统逐步将企业的商业运营和客户服务推向国际化水平。

如今，计算机技术已经十分成熟，软硬件的发展也日新月异，用户需求也日益多样化，因此需要企业不断优化仓库管理系统。然而目前的情况并不容乐观[3]。在实际情况中，关于仓库管理系统的内容非常繁多，其中包括对仓库自身状态的显示以及查看、对用户员工的管理以及产品库存的增删改查操作等。尽管企业管理发展总体上呈现积极向好的趋势，但要彻底淘汰一些落后的管理方式仍需要时间。现代化企业仓库管理系统的开发和应用，可以使得管理员能够通过现代化的互联网计算机技术，方便全面地管理仓库系统以及员工管理，并且可以确保产品数据的实时性和正确性，避免了过去繁琐的人工方式所带来的误差，从而提高了工作效率。仓库操作人员可以通过用户名和密码登录系统，并根据其权限进行划分。用户界面友好、操作简便，无需花费过多时间学习软件的使用方法。

2 开发技术相关概述

2.1 Django 框架简介

Django 框架作为 Python 的三大主流 Web 框架之一[4]，几乎囊括了 Web 应用的各个方面，并提供了许多常用于网站后台开发的模块。Django 作为一个高效、功能强大的 Web 框架，拥有许多优点。首先，它提供了完整的开发环境，包括 ORM（对象关系映射）、模板引擎、表单处理、认证系统等，使得开发者能够快速构建功能丰富的 Web 应用。其次，Django 具有强大的安全性，内置了防止常见攻击（如跨站点请求伪造、SQL 注入等）的机制，为应用程序提供了可靠的保护。此外，Django 的文档详尽且易于理解，社区庞大活跃，提供了丰富的插件和第三方应用程序，可以大大加速开发过程。

Django 框架采用了 MTV 的软件设计模式，即模型(Model)、视图(View)和模板(Template)，这是借鉴了 MVC 的设计模式[5]。

2.2 React 框架简介

React 是一个专注于视图层的框架[6]。React 作为一个流行的前端框架，具有诸多优点。首先，React 采用了组件化的开发思想，使得代码结构清晰，可维护性高。通过将页面拆分成多个组件，开发者可以更加专注于每个组件的功能和逻辑，提高了代码的复用性和可扩展性。其次，React 采用了虚拟 DOM 技术，通过比对前后两次渲染的虚拟 DOM 树，只对需要更新的部分进行重新渲染，从而提高了页面渲染的性能和效率。此外，React 还具有较为灵活的生态系统，丰富的第三方库和组件，以及强大的社区支持，为开发者提供了丰富的资源和工具，加速了项目的开发进程。最重要的是，React 采用了 JSX 语法，将 HTML 和 JavaScript 混合编写，使得代码更加直观清晰，提高了开发效率。

3 系统设计

3.1 数据库中表的设计

良好的数据库设计能够节省数据的存储空间、保证数据的完整性以及能够方便地进行数据库应用系统的开发[7]。在本仓库管理系统中，需要设计多个数据库表来存储各种不同类型的数据，以满足系统的功能需求。其中，关键的数据库表包括用户信息表、产品信息表、分类信息表和供应商信息表等。

首先，用户信息表存储了系统中的用户账号、密码、权限等信息，用于用户的身份验证和权限管理。这个表可以记录用户的基本信息，包括姓名、联系方式等，以便系统管理员进行管理。其次，产品信息表用于存储仓库中各种产品的相关信息，如产品名称、描述、价格、库存量等。通过这个表，可以方便地对产品进行管理和查询，确保库存信息的准确性和及时性。分类信息表用于管理产品的分类信息，如产品分类名称、描述等。通过这个表，可以将产品进行分类管理，方便用户对产品进行组织和检索。最后，供应商信息表存储了与供应商相关的信息，如供应商名称、联系方式、地址等。这个表可以帮助系统管理员跟踪和管理与供应商的合作关系，确保供应链的畅通和稳定。

以下图 1 是本系统建立的部分数据库表。

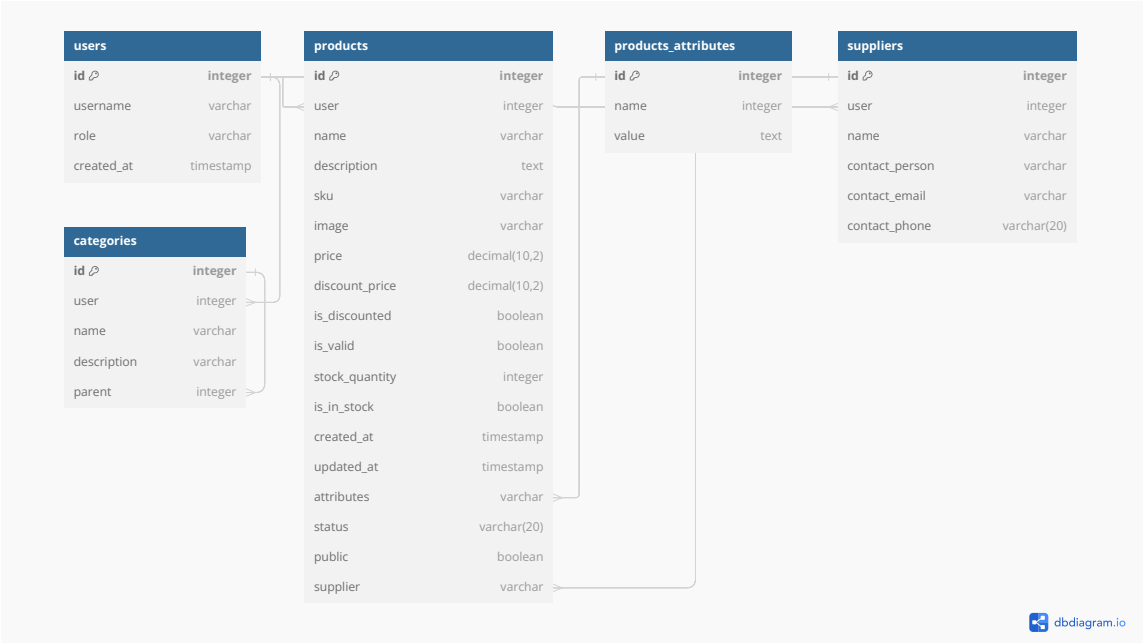

图 1 数据库表设计结构图

3.2 后端功能模块设计

如图 2 和图 3，产品和供应商管理模块允许管理员执行以下操作：

添加新产品/供应商，包括产名称、描述等信息；

删除不再需要的对象；

编辑现有的信息；

查看对象列表，按照不同的排序和筛选条件进行过滤。

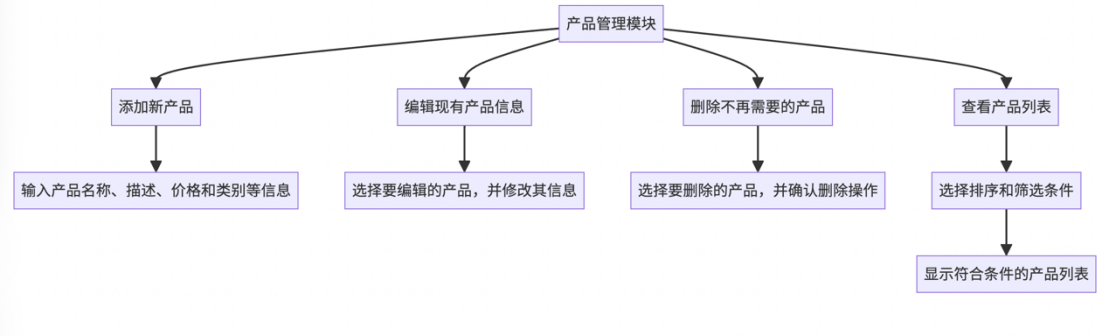

图 2 产品管理模块

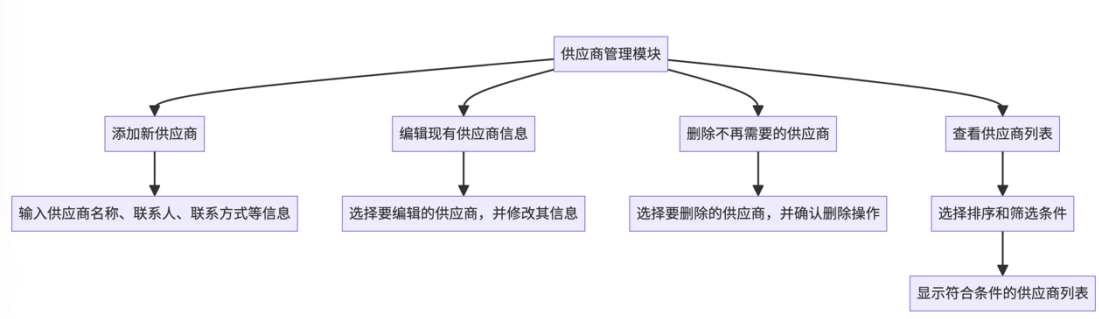

图 3 供应商管理模块

搜索功能模块允许用户在产品和供应商中执行关键字搜索，以便快速找到所需的信息。用户可以在搜索框中输入关键字，系统将返回与关键字匹配的产品和供应商，搜索结果可以根据相关性进行排序，如图 4。

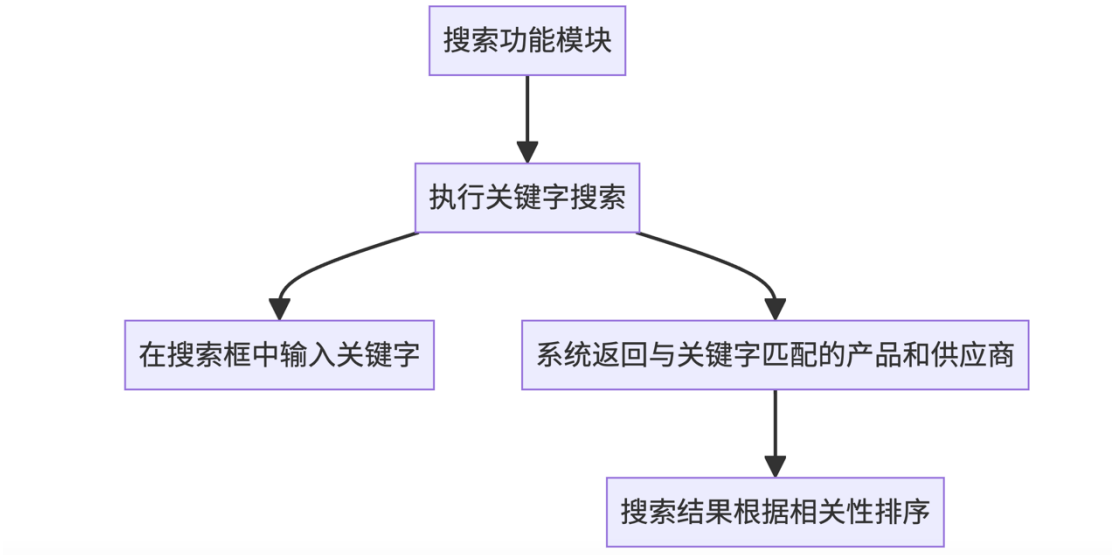

图 4 搜索功能模块

图 5 是用户管理模块允许管理员执行以下操作：
新增新的系统管理用户，包括用户名、密码和操作权限等相关信息；
移除不再需要的管理员用户；
修改现有/先前添加的管理员用户的信息；
查看用户列表，按照不同的排序和筛选条件进行过滤。

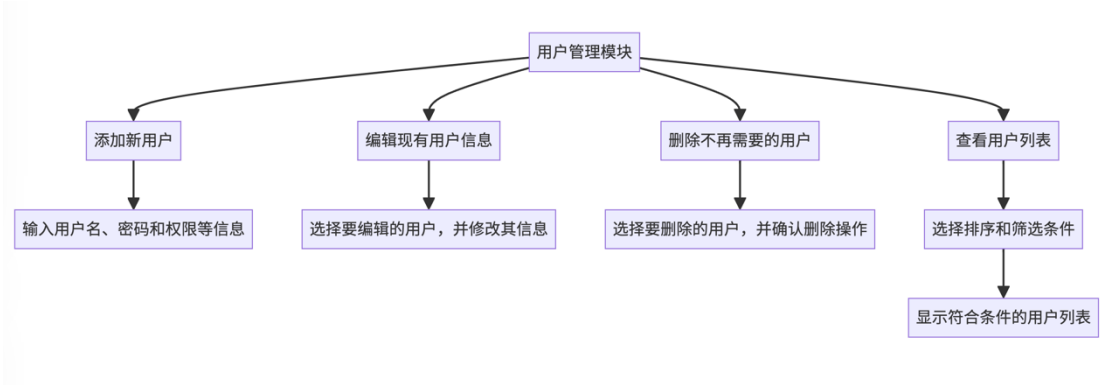

图 5 用户管理模块

类别管理模块，如图 6，允许管理员执行以下操作：
新增新类别，包括产品的类别名称和类型描述等信息；
移除不再需要的产品类别选项；
修改现有的产品类别的相关信息；
查看产品类别列表，并可以按照不同的排序和筛选条件进行过滤。

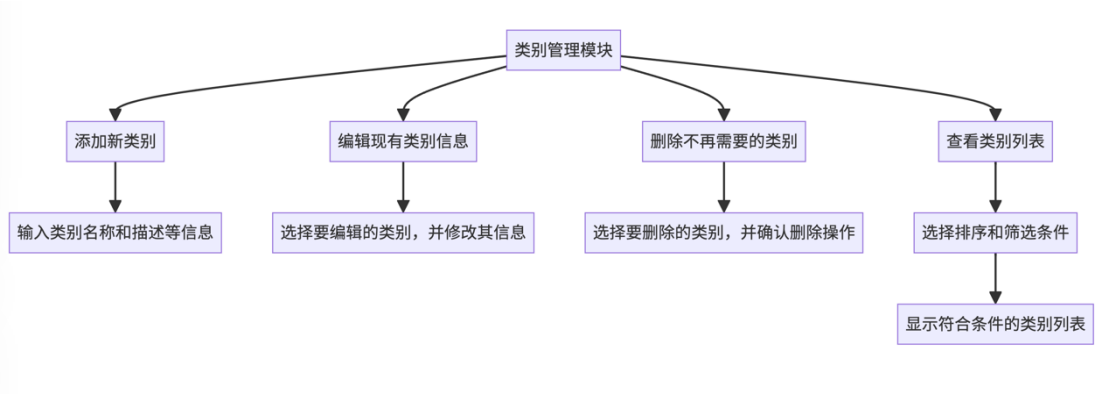

图 6 类别管理模块

3.3 后端代码实现

3.3.1 模型定义

根据前期数据库设计的模型字段和类型，在 Python 中进行 Product 类的定义。其中，categories、attributes 和 supplier 字段为 Django 的 ManyToMany 类型，即存在不同表中的外键与之相关联，这样可以做到在删除 Product 对象时，可以将与之相关的数据进行处理，从而实现数据库模型之间的互相关联，如图 7。

```
class Product(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True, null=True)
    sku = models.CharField(max_length=50, unique=True)
    image = models.CharField(max_length=255, blank=True, null=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    discount_price = models.DecimalField(max_digits=10)
    is_discounted = models.BooleanField(default=False)
    is_valid = models.BooleanField(default=False)
    stock_quantity = models.PositiveIntegerField(default=0)
    is_in_stock = models.BooleanField(default=True)
    categories = models.ManyToManyField("categories.Category")
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    attributes = models.ManyToManyField("ProductAttribute")
    supplier = models.ForeignKey("suppliers.Supplier")
    weight = models.DecimalField(max_digits=10, decimal_places=2)
    dimensions = models.CharField(max_length=50, blank=True, null=True)
    status = models.CharField(max_length=20, choices=ProductStatus.choices)
    public = models.BooleanField(default=True)
```

图 7 后端模型定义

3.3.2 View 视图定义

在 Django 中，通过定义 View 视图，实现对外开放的 API 接口，从而前端可以使用接口进行数据的增删改查，在具体实现时，可以通过 Python 的类的继承相关实现，减少系统的代码的编写量，从而简化开发者的开发工作。以下例子中，通过继承 generics 中的多个基本类，使得同一个 API 接口可以处理不同的增删改查请求，如图 8。

```
class ProductDetailAPIView(
    StaffEditorPermissionMixin,
    generics.RetrieveAPIView,
    generics.DestroyAPIView,
    generics.UpdateAPIView
):
    """
    Product Retrieve Detail API view.
    """
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    lookup_field = 'pk'
    parser_classes = (MultiPartParser, FormParser, JSONParser)

    def retrieve(self, request, *args, **kwargs):
        """
        Retrieve a model instance.
        """
        instance = self.get_object()
        serializer = self.get_serializer(instance)
        data = serializer.data
        result = Result(status='success', message='Product retrieved successfully', data=data)
        return JsonResponse(result.to_json(), status=200)
```

图 8　View 视图定义

3.3.3 序列化器的定义

后端在接收到前端发送来的原始 json 数据时，通过 Serializer 序列化器来对数据进行序列化。通过定义序列化器，将先前定义的数据模型各个字段进行解包，实现原始 json 数据与后端系统模型的字段间对应。通过继承 Django Rest Framework 的序列化器的类，可以增加自定义的字段处理逻辑，可以满足不同的数据处理要求，以下为 Rest Frame Work 中自带的序列化器的具体实现，指明序列化的寻找区域为 PK 字段，如图 9。

```
class ProductSerializer(serializers.ModelSerializer):
    url = serializers.HyperlinkedIdentityField(view_name="product_detail", lookup_field='pk')
    owner = UserPublicSerializer(source='user', read_only=True)
    # edit_url = serializers.SerializerMethodField(read_only=True)
    other_products = ProductInlineSerializer(
        source="user.product_set.all",
        read_only=True,
        many=True
    )
```

图 9 序列化器定义

3.4 前端代码实现

在实现前端代码之前，要完成不同部分之间系统接口的对接，前端通过后端提供的 API 接口，通过后端 API 进行的相关操作，获取到后端数据库的数据，并在前端页面显示出来，以上功能的实现，需要进行系统集成，确保高效的开发以及逻辑的正确。

在系统集成阶段，我们实现了 Django 后端与 React 前端的紧密融合，以确保数据的无缝传递和系统的高效运行。通过采用先进的集成策略，我们将构建一个协调一致的系

统，使得后端与前端之间的通信畅通无阻，如图 10。

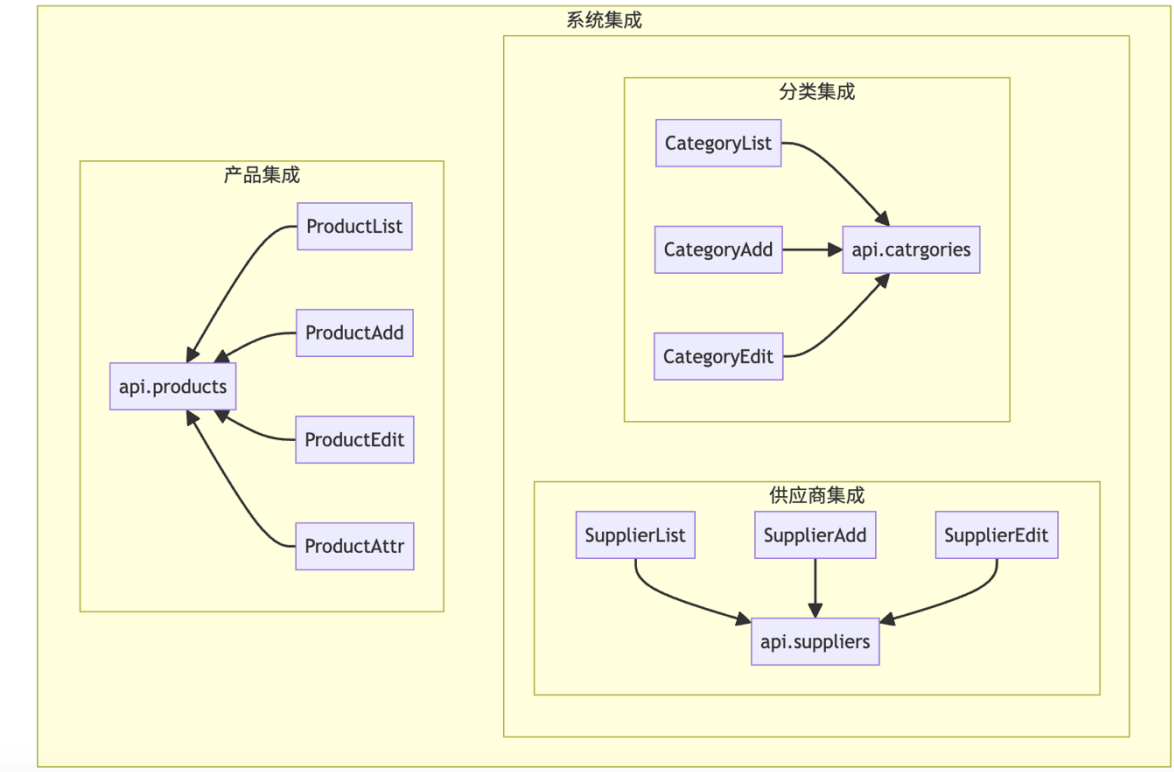

图 10 系统集成

与传统的服务器端路由不同，React Router 使用了基于浏览器地址栏的客户端路由。这意味着在用户导航时，页面的切换不会使整个页面刷新，而是通过 JS 动态地刷新页面内容，从而提供更加流畅的用户体验。

React Router 允许开发人员将应用程序分解成多个独立的组件，并根据 URL 路径来渲染这些组件。它提供了一系列的组件，如<Browser Router>、<Route>、<Switch>等，来帮助开发人员定义和管理路由规则。通过定义 React 路由，实现页面的懒加载，从而提升页面切换的速度，改善用户体验，见图 11。

```
function App() {
    return (
        <>
            <Router>
                <Routes>
                    <Route path="*" element={<Notfound/>}/>
                    <Route path="/workspace/*" element={<WorkSpace/>}/>
                    <Route path="/login" element={<Login/>}/>
                    <Route path="/signup/*" element={<SignUpRoutes/>}/>
                    <Route path="/welcome" element={<Welcome/>}/>
                    <Route path="/about" element={<About/>}/>
                    <Route path="/" element={<Welcome/>}/>
                </Routes>
            </Router>
        </>
    );
}
```

图 11 react route

通过 Ant Design 的上传图片组建，可以实现上传图片并在前端实时显示出来，与此同时，后端需要配合实现接口，达到存储文件到本地后返回文件的 URL 链接，前端页面通过使用返回的图片链接，在页面中显示图片预览，供用户查看使用。通过二者的配合，可以实现图片的上传以及显示，见图 12。

```
/**
 *  Url for displaying preview image.
 */
const [imagePreviewURL, setImagePreviewURL] = useState("");

/**
 * Configuration for upload element.
 */
const props: UploadProps = {
    name: "image",
    action: `${axios.defaults.baseURL}/api/upload/`,
    headers: {
        Authorization: `Bearer ${get_local_access_code()}`,
    },
    onChange(
        event: UploadChangeParam<
            UploadFile<{
                image_url: string;
            }>
        >
    ) {
        if (event.file && event.file.response && event.file.response.image_url) {
            setImageURL(event.file.response.image_url);
            setImagePreviewURL(event.file.response.image_url);
        } else {
            console.error("Unexpected response structure:", event);
        }
    },
};
```

图 12 图片上传前端实现

4. 开发成果

4.1 用户登录及注册

在用户登录页面，提供记住账号和密码的选项，可以有效提高登录的效率。这一功能允许用户在下次登录时自动填充账号和密码，省去重复输入的繁琐步骤，提升用户体验。同时，我们设计了注册按钮，使用户可以轻松地跳转到用户注册页面进行新用户注册。一旦完成注册流程，系统会自动跳转回登录页面，让用户立即登录，符合用户的逻辑需求。这种设计方案旨在简化用户操作流程，提高用户体验。记住账号和密码选项减少了用户每次登录时的输入次数，节省了时间和精力，提高了用户的满意度。而注册按钮的设置则使得用户能够方便地完成注册流程，随时享受系统提供的服务，增强了用户的黏性，见图 13。

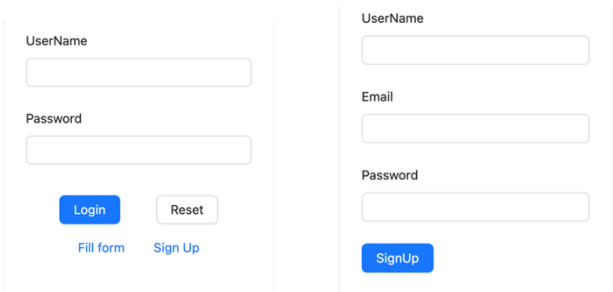

图 13 用户登录及注册页面

4.2 供应商操作

供应商管理界面是任何企业管理系统中至关重要的一部分。其功能不仅仅是简单地显示供应商列表，更要实现对供应商信息的全面管理，包括添加、删除、修改等操作。为了提升用户体验，我们采用了对齐布局和优美大方的样式设计，以确保用户在使用过程中感受到舒适和便捷。在供应商列表的显示方面，我们精心设计了布局，使得每个供应商信息都清晰可见，同时利用对齐布局使得界面整洁有序。通过这样的设计，用户可以轻松地浏览和查找所需的供应商信息，减少了操作的繁琐性和可能的误操作。在添加、删除和修改供应商信息的功能上，我们着重考虑了操作的便捷性和准确度。通过直观的界面设计和必填选项的设置，用户在添加供应商时必须提供其名称等基本信息，从而确保了数据的完整性和准确性。同时，我们优化了删除和修改功能的交互流程，使得用户能够快速且准确地完成操作，减少了出错的可能性，如图 14。

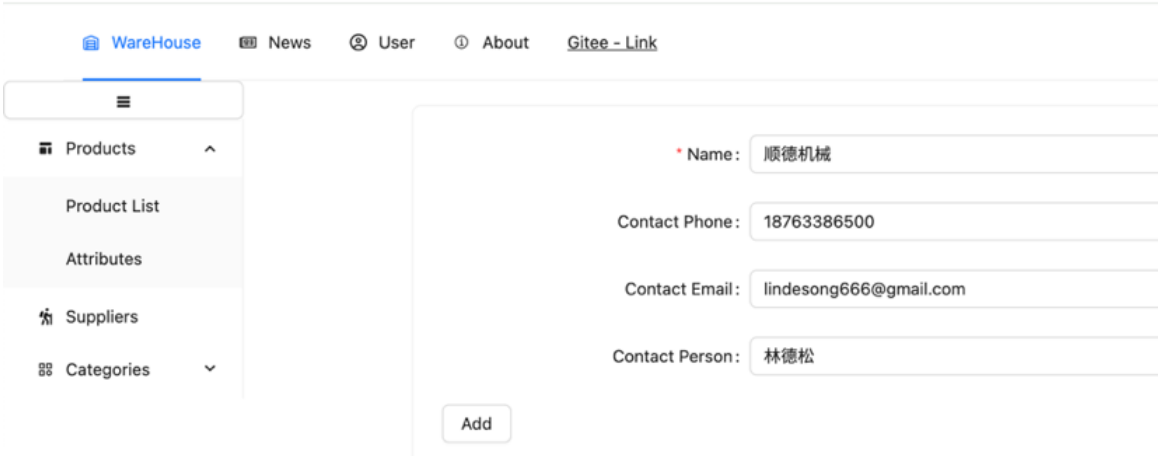

图 14 供应商添加

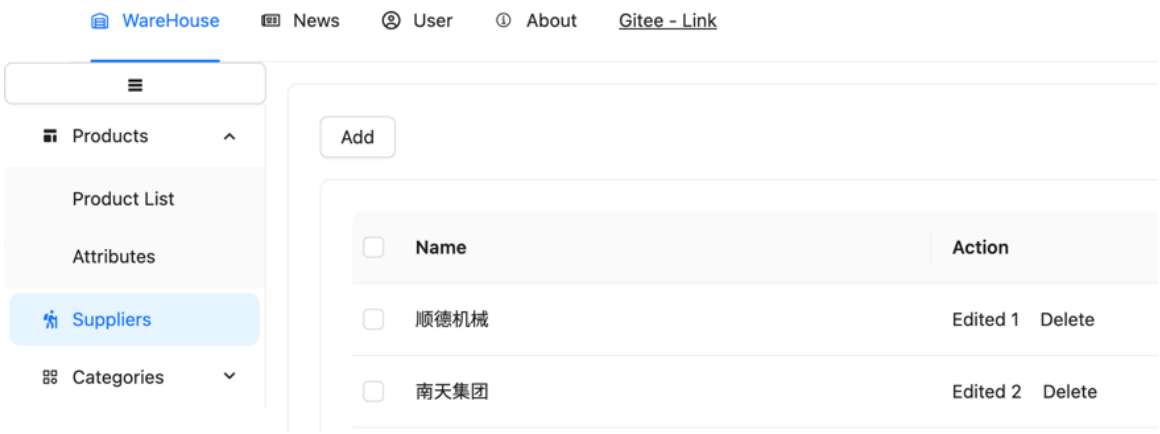

图 15 供应商列表

4.3 分类操作

在分类管理方面，我们引入了一级分类和二级分类的概念，以更清晰地组织产品分类，并提高分类管理的效率和整洁度。当用户添加新的分类时，系统允许用户选择将该分类设定为一级分类或者二级分类。这样的设计使得用户在创建分类时可以根据实际需要进行灵活选择，使得产品的分类更加清晰明了。在分类的列表页面，我们将一级分类和二级分类进行了分别显示，从而有效地区分不同层级的分类。通过这种方式，用户可以快速地定位到所需的分类，而无需在繁杂的分类列表中进行过多的搜索和筛选，提高了操作的效率和准确性。这种一级分类和二级分类的设计，不仅使得产品分类更加清晰，还使得分类管理界面更加整洁和易于操作。用户可以根据实际需要轻松地添加、删除和修改分类，从而更好地组织和管理产品信息，如图 16，图 17。

图 16 分类添加

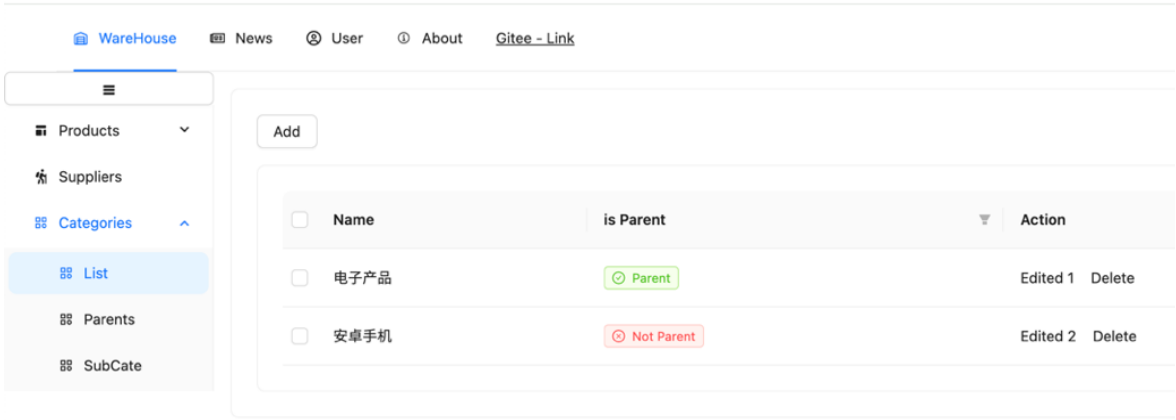

图 17 分类列表

4.4 产品操作

在产品添加页面的设计中，我们充分利用了 Ant Design 的 Tag 组件，以实现对产品属性和分类的设置，同时结合图片上传功能，提供实时预览，使得数据填写更加直观高效。我们通过 Ant Design 的 Tag 组件，为用户提供了便捷的产品属性和分类设置功能。用户可以通过简单的操作，选择或输入相关的产品属性，并将其以标签的形式呈现在页面上。这种标签式的设计不仅使得产品属性的设置更加直观和易于理解，还使得用户能

够快速地进行多个属性的选择和编辑，提高了操作的效率和准确性。在图片上传方面，我们通过 Ant Design 提供的上传组件，实现了图片的上传功能，并在上传完成后立即显示上传的图片。用户可以在上传图片的同时，即时预览所选图片的效果，以确保上传的图片符合要求，并及时调整和修改。这种实时预览的设计不仅提升了用户体验，还帮助用户更加直观地了解最终效果，从而更加高效地进行数据填写和编辑，如图 18。

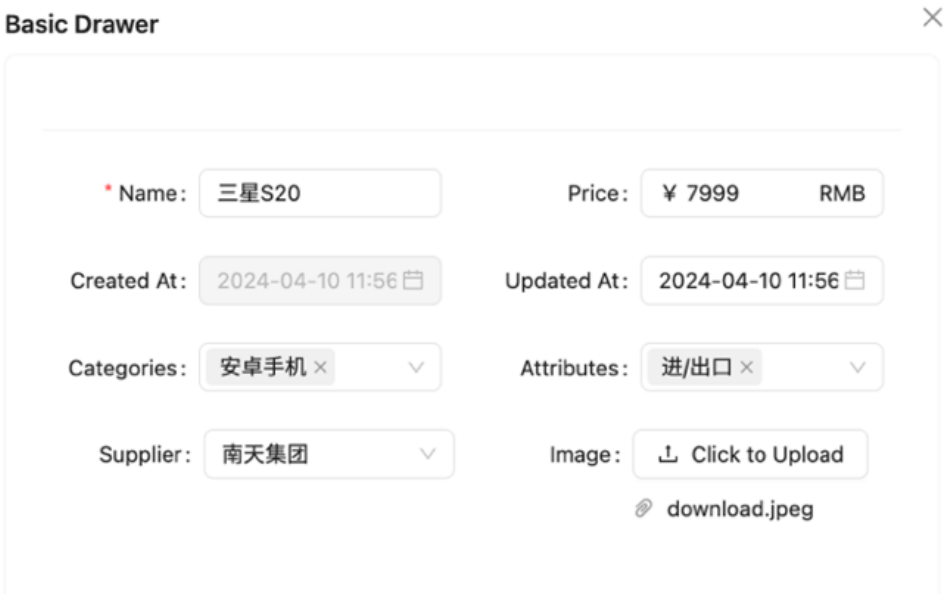

图 18 产品添加

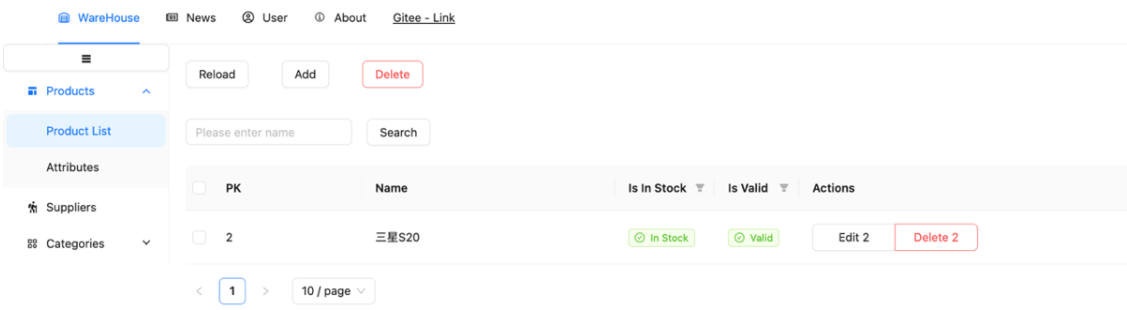

图 19 产品列表

4.5 仪表盘数据分析

仪表盘界面提供的功能为对产品以及供应商的数量进行实时的监控和显示。当用户在后台对产品进行增加或者删除操作后，对应仪表盘界面则会实时刷新图表的数据，这样做可以使得不同部门的用户之间互通库存数据。对供应商的图表显示，则会对不同供应商所对应的产品量进行实时的监控和显示，当添加对应供应商的产品时，图表会实时显示出产品的数量，以供用户查看，如图 19。

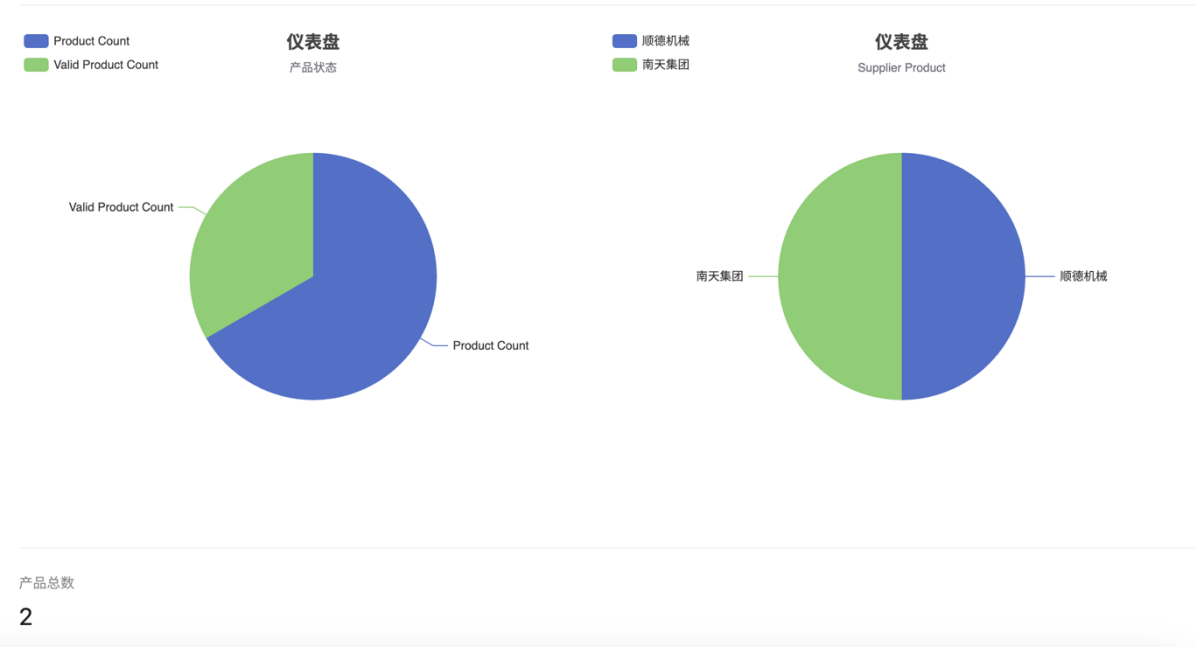

图 20 仪表盘数据分析

5. 系统测试

为保证系统的正常运行需要在系统设计实现完毕后进行测试，判断系统能否达到功能的要求，以减少后期的维护费用，提高用户体验感。

5.1 测试方法

此仓库管理系统主要使用功能测试来进行发布前的测试。功能测试是指测试人员站在用户的角度，根据产品设计人员设计的功能，在使用 Web 应用系统过程中，覆盖用户所有的操作[8]。我们采用黑盒测试方法，不必了解系统功能背后的实现逻辑与技术细节，只需关注产品的功能是否符合用户的预期，并且找到潜在的逻辑缺陷。

5.2 登录退出功能测试

测试系统的登录、退出、注册页面能否正常跳转，以及账号密码错误或不符合要求时，能否正常提示。测试结果如表 1。

序号	测试场景	输入内容	预期结果	测试结果
1	登陆测试	输入错误账号	登陆失败	达到预期
2		输入错误密码	登陆失败	达到预期
3	注册测试	输入存在账号密码	注册失败	达到预期
4		输入不存在账号密码	注册成功	达到预期
5	退出测试	点击主页退出按钮	进入首页	达到预期

表 1 登录功能测试

5.3 用户基础功能测试

测试用户能否正常的进行产品的增删改查、产品属性的设置、分类的增加和删除，并可以进行图片的上传。测试结果如表 2。

序号	测试场景	输入内容	预期结果	测试结果
1	修改测试	修改产品分类	修改成功	达到预期
2		修改产品信息	修改成功	达到预期
3		修改供应商信息	修改成功	达到预期
4	查询测试	产品按 id 查询	查询成功	达到预期
5		产品按名称查询	查询成功	达到预期
7	删除测试	删除单个产品	删除成功	达到预期
8		批量删除产品	删除成功	达到预期
9		删除附加属性	删除成功	达到预期
10	分析测试	点击统计分析	跳转仪表盘界面	达到预期

表 2 用户基础功能测试

5.4 测试结果评价

根据以上的一系列功能测试，并进行多次的测试场景变换测试，最后所得到的测试结果，表明系统功能的开发符合设计预期。在测试过程中所发现的系统逻辑缺陷，及时得到了修改，并不断优化现有的功能逻辑，使得系统的功能和整体性能得到很大程度的优化，表明测试功能符合软件测试的预期。

6. 总结与展望

本现代化仓库管理系统会给企业仓库管理方面带来很多优化，其中包括以下方面：

最大限度利用仓库，实现动态、随机库存：系统能够优化仓库空间利用，实现灵活、高效的库存管理，使得仓库内物品摆放更加合理，提高仓库存储效率[9]。

提供即时且准确的信息，全面记录仓储流程、库存状况、资源利用程度：系统能够实时监控仓储流程，提供详尽的库存信息，为不同部门的管理员提供精准实时的数据，帮助管理人员快速了解仓储状况，优化资源利用，提高管理水平。

减少库存损耗，降低人工输入数据误差：系统能够自动化数据录入和更新，减少人为操作引起的错误，提高数据准确性，同时通过及时的库存监控，降低库存损耗。

这些效益不仅能够为商家带来良好的收益，同时也无形中提高了仓库空间利用率，减少了库存损耗，并减轻了人工输入数据的误差。

相较于其他系统级编程语言，Python 在处理速度等方面仍不占优势，在未来，可以通过引入快速高效的系统级编程语言（如 Rust[10]），重构后端的部分代码，从而提高系统的整体反应速度以及处理数据的效率。

致谢

我要真诚地感谢我的导师李霞老师。从选题到调研，再到实现与测试，都得到了李霞老师的悉心指导。李霞老师不辞辛劳地为我们审阅论文，不厌其烦地指出论文中的结构不合理之处和内容的不足之处，甚至包括细节方面的问题。做本设计的过程中，遇到了很多难以解决的技术问题，但这次在李霞老师的精神指引下，我终于迎刃而解，积累了丰富的工程项目经验、极强的动手能力和积极乐观的人生态度，受益良多。

我还要感谢我的朋友苏阔阔，纪如意，他们使我最后的本科时光不那么枯燥乏味。我们一起努力，相互加油和鼓励，偶尔也劳逸结合进行娱乐项目。毫无疑问，我的一切

都与我的父母在背后默默的支持密不可分。对他们我无以为报，只能靠自己的努力来证明给自己，证明给所有帮助过我的人。

感谢这一路遇到的所有师长悉心指导。学生愚笨，这篇论文虽然简陋浅显，但也凝聚着指导教师李霞老师以及所有任课老师四年的淳淳教诲。

感谢曲阜师范大学，让我变得更加独立勇敢的去冒险。

参考文献

[1] 王冉阳. 基于 Django 和 Python 的 Web 开发[J]. 电脑编程技巧与维，2009.
[2] 李伟. 基于 Spring 的仓库管理系统的设计与实现[D]. 西安电子科技大学，2020.
[3] 张天一. 仓库管理系统的设计与实现[D]. 电子科技大学，2013.
[4] 邱红丽;张舒雅. 基于 Django 框架的 web 项目开发研究[J]. 科学技术创新，2021.
[5] 郭蕊;赵元苏. Web 前端框架技术综述[J]. 北京工业职业技术学院学报，2021.
[6] 贺怡龙. 基于 React 框架的校园点餐 APP 系统设计[J]. 产业与科技论坛，2021.
[7] 胡从寅. 基于 Django+Vue.js 的设计作品交易平台的实现[J]. 软件，2023.
[8] 陈春玲. 软件工程与数据库概论[M]. 西安电子科技大学出版社, 2002.
[9] 宁江. 自动化仓库管理系统的设计与实现[J]. 科技视界，2013.
[10] 顾锡华. Rust 语言在 Web 开发的应用研究[J]. 电脑知识与技术，2024.