第一章 概述

1.1 课程设计目的

通过本课程设计的准备与总结，复习、领会、巩固和运用课堂上所学的软件开发方法和知识，为学生综合应用本专业所学习的多门课程知识创造实践机会，使每个学生了解软件工具与环境对于项目开发的重要性，并且重点深入掌握好几种较新或较流行的软件工具或计算机应用技术，提高学生今后参与开发稍大规模实际软件项目和探索未知领域的能力和自信心。

本次课程设计通过开发一个完整的在线电脑DIY系统，让学生深入掌握现代Web开发技术栈，包括前端框架、后端开发、数据库设计、API设计等核心技能。

1.2 课程设计任务

本课程设计的主要任务是开发一个功能完整、用户友好的在线电脑DIY系统。

1.2.1 基础功能要求

系统需要实现以下基础功能：

1. **首页展示功能**：企业介绍和电脑配件（CPU、内存、硬盘、主板、显卡、机箱）的详情展示，无需登录即可浏览
2. **多品牌配件管理**：每种配件支持多个品牌，不同价格层次满足用户需求
3. **用户权限管理**：包含管理员和普通用户两种角色，需要登录后才能进行核心操作
4. **管理员功能模块**：

   配件类型管理、商品信息管理、用户信息浏览、订单查看和管理
5. **用户功能模块**：

   用户注册和登录、个人信息修改、配件选购和购物车管理、装机单配置（每种配件选择1个）、订单提交和结算（模拟付款）、个人订单查看
6. **数据统计功能**：

   1. 按消费金额倒排的前十用户列表
   2. 按销售量倒排序的前十配件列表
   3. 统计结果的图形化展示

1.2.2 创新功能实现

在满足基础要求的基础上，本系统还实现了以下创新功能：

1. **AI智能助手**：
    1. 集成Claude AI模型，提供专业的配件咨询服务
    2. 支持自然语言查询配件信息
    3. 智能推荐适合的配件组合
    4. 基于预算和需求的个性化建议
2. **Tool工具调用系统**：
    1. 实现了query-components工具，支持按类型、品牌、价格等条件智能查询
    2. 实现了show-components工具，以卡片形式展示配件详情
    3. 支持流式响应，提供实时的交互体验

1.3 使用技术及开发环境

1.3.1 技术栈概述

本项目采用了现代化的全栈开发技术栈，充分体现了当前软件开发的最佳实践和技术趋势。

前端技术栈

- **Next.js 15**：作为核心React框架，提供服务端渲染(SSR)、静态站点生成(SSG)、文件系统路由等现代化特性
- **React 19**：最新版本的React库，提供组件化开发和状态管理
- **TypeScript 5**：提供类型安全和更好的开发体验
- **Tailwind CSS 4：原子化CSS框架，快速构建现代化UI界面

后端技术栈

- **Next.js API Routes**：基于Next.js的API路由系统，实现RESTful API
- **Prisma 6.10.1**：现代化的数据库ORM工具，提供类型安全的数据库操作
- **PostgreSQL**：关系型数据库，存储用户、商品、订单等核心数据
- **JSON Web Token (JWT)**：用户身份认证和授权机制
- **bcryptjs 3.0.2**：密码加密和验证

AI集成技术

- **Anthropic Claude API**：集成Claude AI模型，提供智能对话功能
- **@anthropic-ai/sdk 0.54.0**：Anthropic官方SDK，处理AI API调用

开发工具和环境

- **Bun**：高性能的JavaScript运行时和包管理器
- **Visual Studio Code**：主要开发IDE

第二章 需求分析

2.1 功能分析

2.1.1 系统概述

在线电脑DIY系统是一个面向电脑爱好者和DIY玩家的综合性电商平台。系统不仅提供传统的商品浏览、购买功能，更创新性地集成了AI智能助手，为用户提供专业的配件选择建议和装机指导。系统采用B2C模式，支持多角色用户管理，具备完整的商品管理、订单处理、数据统计等功能模块。

2.1.3 功能模块详细分析

2.1.3.1 用户管理模块

基础功能：

- **用户注册**：支持邮箱注册，验证用户名唯一性
- **用户登录**：基于JWT的安全认证机制
- **个人信息管理**：用户可修改个人资料、联系方式
- **密码管理**：安全的密码修改功能

技术实现特点：

- 采用bcrypt加密存储密码，确保数据安全
- JWT无状态认证，支持分布式部署
- 前端表单验证与后端数据校验双重保障

2.1.3.2 商品管理模块

配件分类管理：

- **配件类型**：CPU、内存、硬盘、主板、显卡、机箱六大类
- **品牌管理**：每类配件支持多品牌，如Intel、AMD、NVIDIA等
- **规格参数**：详细的技术规格和兼容性信息

库存管理：

- **实时库存**：支持库存数量的实时更新

2.1.3.3 购物车与订单模块

购物车功能：

- **商品添加**：支持快速添加配件到购物车
- **数量调整**：灵活的数量增减操作
- **实时计算**：动态显示总价和优惠信息
- **持久化存储**：登录用户购物车数据持久保存

订单处理流程：

- **订单创建**：从购物车生成订单，分配唯一订单号
- **库存扣减**：下单时自动扣减库存，防止超卖
- **订单状态管理**：待确认 → 已确认 → 处理中 → 已发货 → 已送达
- **订单取消**：支持用户主动取消和管理员取消

装机单功能：

- **配件选择约束**：确保每种配件类型只能选择一个
- **兼容性检查**：AI助手提供兼容性建议
- **价格统计**：实时显示装机单总价
- **一键下单**：将完整装机单转换为订单

2.1.3.4 AI智能助手模块

核心对话引擎：

- **自然语言理解**：理解用户的配件需求和预算
- **多轮对话**：支持上下文保持的连续对话
- **专业知识库**：内置丰富的硬件知识和兼容性信息

工具调用系统：

- **query-components工具**：支持按类型、品牌、价格范围查询配件
- **show-components工具**：以卡片形式展示配件详细信息

对话管理：

- **对话历史**：完整记录用户与AI的对话内容
- **对话分类**：按时间和主题自动分类
- **对话导出**：支持对话记录的导出和分享

2.1.3.5 管理后台模块

数据统计分析：

- **用户排行榜**：按消费金额排序的前十用户
- **商品排行榜**：按销售量排序的前十配件
- **销售趋势**：时间维度的销售数据分析
- **图表展示**：使用柱状图、饼图等可视化数据

订单管理：

- **订单列表**：分页显示所有订单信息
- **订单详情**：查看订单的详细配件和用户信息
- **状态更新**：批量或单个订单状态修改
- **订单搜索**：按订单号、用户、状态等条件筛选

商品管理：

- **商品CRUD**：完整的商品增删改查功能
- **批量导入**：支持Excel/CSV格式的批量数据导入
- **图片管理**：商品图片的上传和管理
- **分类管理**：配件分类的维护和调整

2.1.4 功能结构图

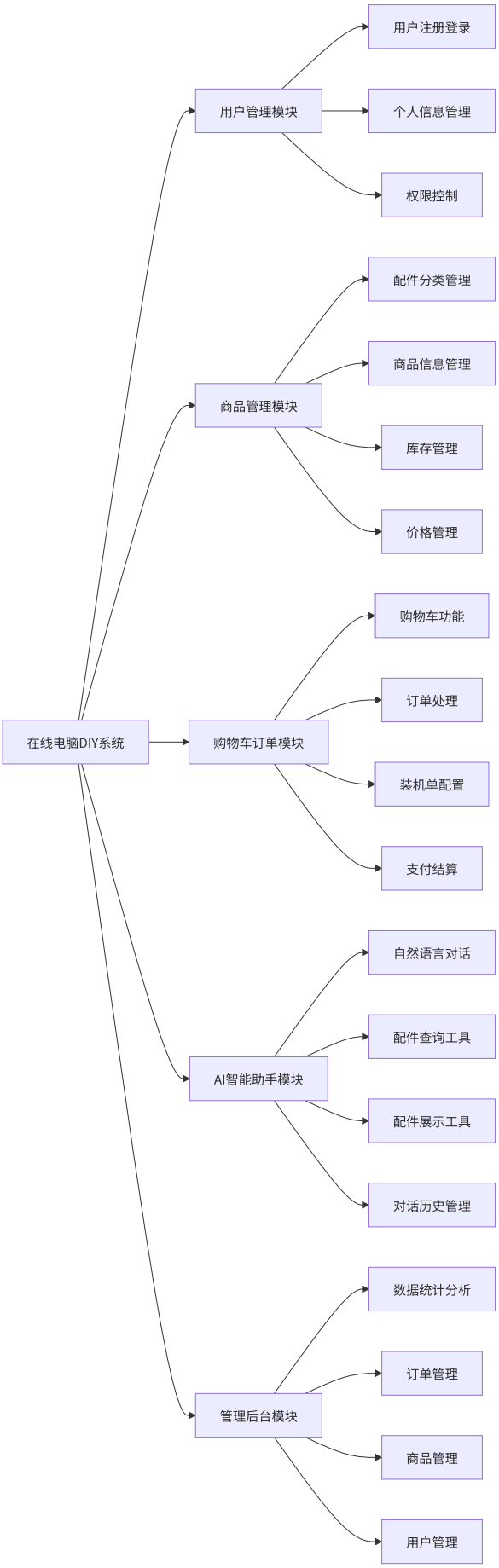
在线电脑DIY系统
用户管理模块
用户注册登录
个人信息管理
权限控制
商品管理模块
配件分类管理
商品信息管理
库存管理
价格管理
购物车订单模块
购物车功能
订单处理
装机单配置
支付结算
AI智能助手模块
自然语言对话
配件查询工具
配件展示工具
对话历史管理
管理后台模块
数据统计分析
订单管理
商品管理
用户管理

图2-1 系统功能结构图

2.2 概念模型分析

2.2.1 实体识别

通过对系统需求的深入分析，识别出以下主要实体：

2.2.1.1 核心业务实体

用户实体（User）

- **属性**：用户ID、用户名、邮箱、密码、姓名、电话、地址、是否管理员、创建时间、更新时间
- **业务意义**：系统的核心参与者，包括普通用户和管理员

配件类型实体（ComponentType）

- **属性**：类型ID、类型名称、描述
- **业务意义**：配件的分类标准，如CPU、内存、硬盘等

配件实体（Component）

- **属性**：配件ID、名称、品牌、型号、价格、描述、图片URL、库存数量、规格参数、创建时间、更新时间
- **业务意义**：系统中的商品实体，用户购买的对象

订单实体（Order）

- **属性**：订单ID、订单号、总金额、状态、创建时间、更新时间
- **业务意义**：用户购买行为的记录

订单项实体（OrderItem）

- **属性**：项目ID、数量、价格
- **业务意义**：订单中具体的商品项

购物车项实体（CartItem）

- **属性**：项目ID、数量、创建时间、更新时间
- **业务意义**：用户临时存放待购买商品的容器

对话实体（Conversation）

- **属性**：对话ID、标题、消息内容（JSON）、创建时间、更新时间
- **业务意义**：用户与AI助手的对话记录

2.2.2 实体关系分析

2.2.2.1 用户相关关系

1. **用户-订单关系（1:N）**
    1. 一个用户可以有多个订单
    2. 一个订单只属于一个用户
    3. 关系属性：下单时间
2. **用户-购物车项关系（1:N）**
    1. 一个用户可以有多个购物车项
    2. 一个购物车项只属于一个用户
    3. 约束：同一用户对同一商品只能有一个购物车记录
3. **用户-对话关系（1:N）**
    1. 一个用户可以有多个AI对话记录
    2. 一个对话记录只属于一个用户

2.2.2.2 商品相关关系

1. **配件类型-配件关系（1:N）**
    1. 一个配件类型下可以有多个配件
    2. 一个配件只属于一个配件类型
2. **配件-订单项关系（1:N）**
    1. 一个配件可以出现在多个订单项中
    2. 一个订单项只对应一个配件
3. **配件-购物车项关系（1:N）**
    1. 一个配件可以被多个用户加入购物车
    2. 一个购物车项只对应一个配件

2.2.2.3 订单相关关系

1. **订单-订单项关系（1:N）**
    1. 一个订单可以包含多个订单项
    2. 一个订单项只属于一个订单
    3. 级联删除：删除订单时同时删除所有订单项

2.2.3 ER图设计

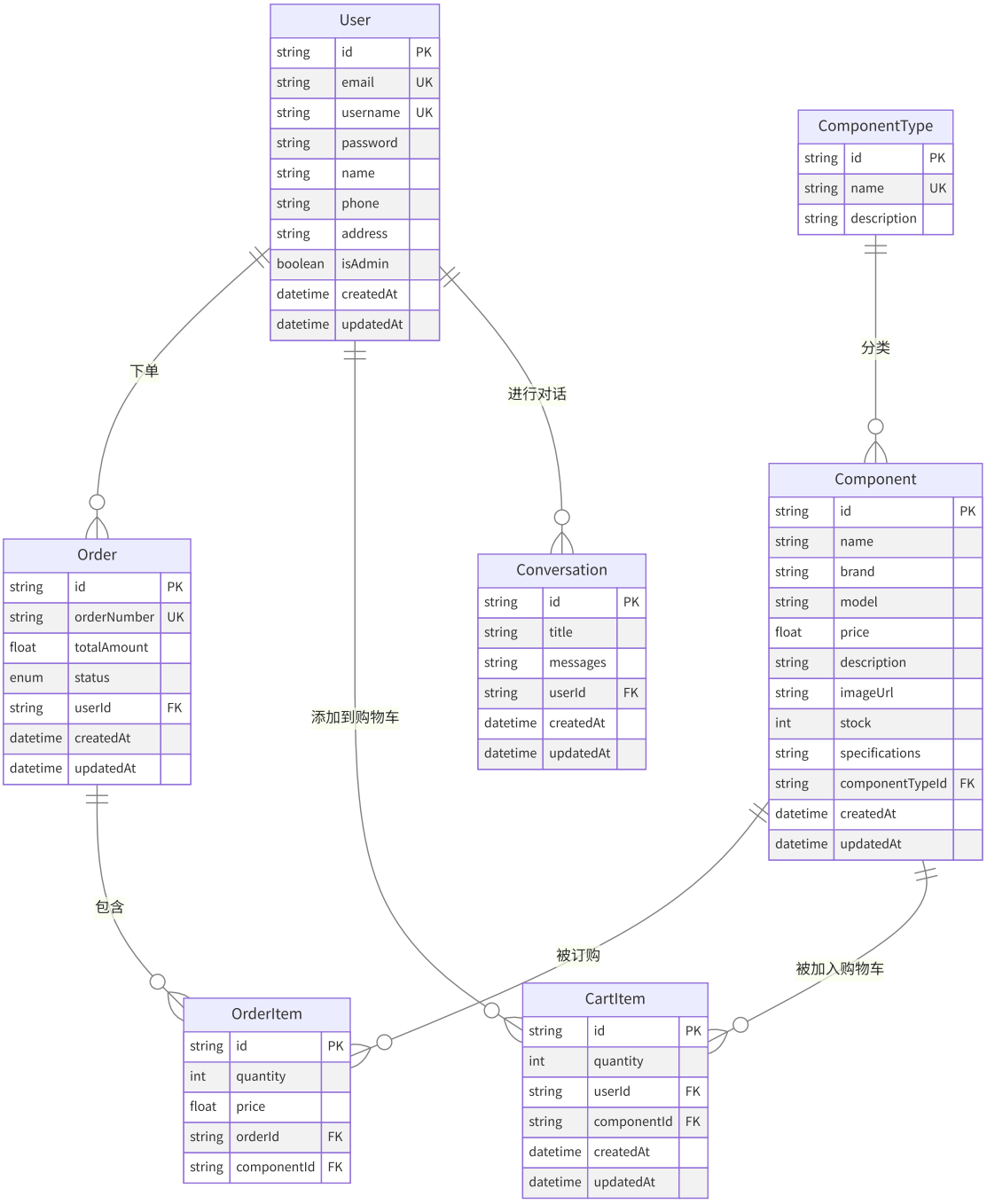

图2-2 系统 ER 图

第三章 系统设计

3.1 数据库设计

3.1.1 数据库设计概述

本系统采用PostgreSQL作为主数据库，通过Prisma ORM进行数据访问。数据库设计遵循第三范式（3NF），确保数据的一致性和完整性，同时兼顾查询性能和扩展性。数据库设计充分考虑了AI助手功能的特殊需求，包括对话历史的存储和快速检索。

3.1.2 数据表详细设计

3.1.2.1 用户管理相关表

用户表（users）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	CUID主键，全局唯一标识符
email	VARCHAR(255)	UNIQUE NOT NULL	用户邮箱，用于登录和通知
username	VARCHAR(50)	UNIQUE NOT NULL	用户名，用于显示和登录
password	VARCHAR(255)	NOT NULL	加密后的密码，使用bcrypt加密
name	VARCHAR(100)	NULL	真实姓名，可选填写
phone	VARCHAR(20)	NULL	联系电话，用于订单配送
address	TEXT	NULL	收货地址，支持多行地址
isAdmin	BOOLEAN	DEFAULT FALSE	是否为管理员，控制系统权限
createdAt	TIMESTAMP	DEFAULT NOW()	注册时间，自动记录
updatedAt	TIMESTAMP	DEFAULT NOW()	最后更新时间，自动维护

索引设计：

```sql
-- 邮箱唯一索引（用于登录查询）
CREATE UNIQUE INDEX idx_users_email ON users(email);

-- 用户名唯一索引（用于登录查询）
CREATE UNIQUE INDEX idx_users_username ON users(username);

-- 管理员查询索引
CREATE INDEX idx_users_isAdmin ON users(isAdmin);

-- 注册时间索引（用于用户增长分析）
CREATE INDEX idx_users_createdAt ON users(createdAt);
```

3.1.2.2 商品管理相关表

配件类型表（component_types）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	类型唯一标识符
name	VARCHAR(50)	UNIQUE NOT NULL	类型名称（CPU、内存、硬盘等）
description	TEXT	NULL	类型描述信息

配件表（components）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	配件唯一标识符
name	VARCHAR(255)	NOT NULL	配件名称
brand	VARCHAR(100)	NOT NULL	品牌名称
model	VARCHAR(100)	NOT NULL	型号信息
price	DECIMAL(10,2)	NOT NULL CHECK(price > 0)	价格，精确到分
description	TEXT	NULL	详细描述
imageUrl	VARCHAR(500)	NULL	产品图片URL
stock	INTEGER	DEFAULT 0 CHECK(stock >= 0)	库存数量

字段名	数据类型	约束条件	说明
specifications	TEXT	NULL	规格参数（JSON格式）
componentTypeId	VARCHAR(25)	NOT NULL	外键，关联配件类型
createdAt	TIMESTAMP	DEFAULT NOW()	创建时间
updatedAt	TIMESTAMP	DEFAULT NOW()	更新时间

外键约束：

```
ALTER TABLE components
ADD CONSTRAINT fk_components_componentType
FOREIGN KEY (componentTypeId) REFERENCES component_types(id);
```

索引设计：

```
-- 配件类型查询索引
CREATE INDEX idx_components_typeId ON components(componentTypeId);

-- 品牌查询索引
CREATE INDEX idx_components_brand ON components(brand);

-- 价格范围查询索引
CREATE INDEX idx_components_price ON components(price);

-- 库存查询索引
CREATE INDEX idx_components_stock ON components(stock);

-- 全文搜索索引
CREATE INDEX idx_components_search ON components
USING gin(to_tsvector('simple', name || ' ' || brand || ' ' ||
model));

-- 复合索引（类型+价格，常用查询组合）
CREATE INDEX idx_components_type_price ON components(componentTypeId,
price);
```

3.1.2.3 订单管理相关表

订单表（orders）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	订单唯一标识符
orderNumber	VARCHAR(50)	UNIQUE NOT NULL	订单号，用户可见
totalAmount	DECIMAL(10,2)	NOT NULL CHECK(totalAmount > 0)	订单总金额
status	order_status_enum	DEFAULT 'PENDING'	订单状态枚举
userId	VARCHAR(25)	NOT NULL	外键，关联用户
createdAt	TIMESTAMP	DEFAULT NOW()	下单时间
updatedAt	TIMESTAMP	DEFAULT NOW()	状态更新时间

订单状态枚举：

```
CREATE TYPE order_status_enum AS ENUM (
    'PENDING',    -- 待确认
    'CONFIRMED',  -- 已确认
    'PROCESSING', -- 处理中
    'SHIPPED',    -- 已发货
    'DELIVERED',  -- 已送达
    'CANCELLED'   -- 已取消
);
```

订单项表（order_items）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	订单项唯一标识符

字段名	数据类型	约束条件	说明
quantity	INTEGER	DEFAULT 1 CHECK(quantity > 0)	购买数量
price	DECIMAL(10,2)	NOT NULL CHECK(price > 0)	下单时的价格
orderId	VARCHAR(25)	NOT NULL	外键，关联订单
componentId	VARCHAR(25)	NOT NULL	外键，关联配件

外键约束：

```
ALTER TABLE orders
ADD CONSTRAINT fk_orders_user
FOREIGN KEY (userId) REFERENCES users(id);

ALTER TABLE order_items
ADD CONSTRAINT fk_order_items_order
FOREIGN KEY (orderId) REFERENCES orders(id) ON DELETE CASCADE;

ALTER TABLE order_items
ADD CONSTRAINT fk_order_items_component
FOREIGN KEY (componentId) REFERENCES components(id);
```

3.1.2.4 购物车相关表

购物车项表（cart_items）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	购物车项唯一标识符
quantity	INTEGER	DEFAULT 1 CHECK(quantity > 0)	商品数量
userId	VARCHAR(25)	NOT NULL	外键，关联用户
componentId	VARCHAR(25)	NOT NULL	外键，关联配件

字段名	数据类型	约束条件	说明
createdAt	TIMESTAMP	DEFAULT NOW()	添加时间
updatedAt	TIMESTAMP	DEFAULT NOW()	更新时间

唯一约束：

```
-- 确保同一用户对同一商品只有一个购物车记录
ALTER TABLE cart_items
ADD CONSTRAINT uk_cart_items_user_component
UNIQUE (userId, componentId);
```

3.1.2.5 AI对话相关表

对话表（conversations）

字段名	数据类型	约束条件	说明
id	VARCHAR(25)	PRIMARY KEY	对话唯一标识符
title	VARCHAR(255)	NULL	对话标题，可自动生成
messages	TEXT	NOT NULL	对话消息JSON数据
userId	VARCHAR(25)	NOT NULL	外键，关联用户
createdAt	TIMESTAMP	DEFAULT NOW()	对话创建时间
updatedAt	TIMESTAMP	DEFAULT NOW()	最后消息时间

对话消息JSON结构：

```
interface MessageParam {
  role: "user" | "assistant" | "system";
  content: string | ToolUseContent[];
}

interface ToolUseContent {
  type: "text" | "tool_use" | "tool_result";
  text?: string;
  id?: string;
  name?: string;
  input?: any;
  tool_use_id?: string;
}
```

3.2 模块设计

3.2.1 系统架构设计

3.2.1.1 总体架构

本系统采用现代化的三层架构设计，结合微服务理念，实现了高内聚、低耦合的模块化设计。系统架构如下：

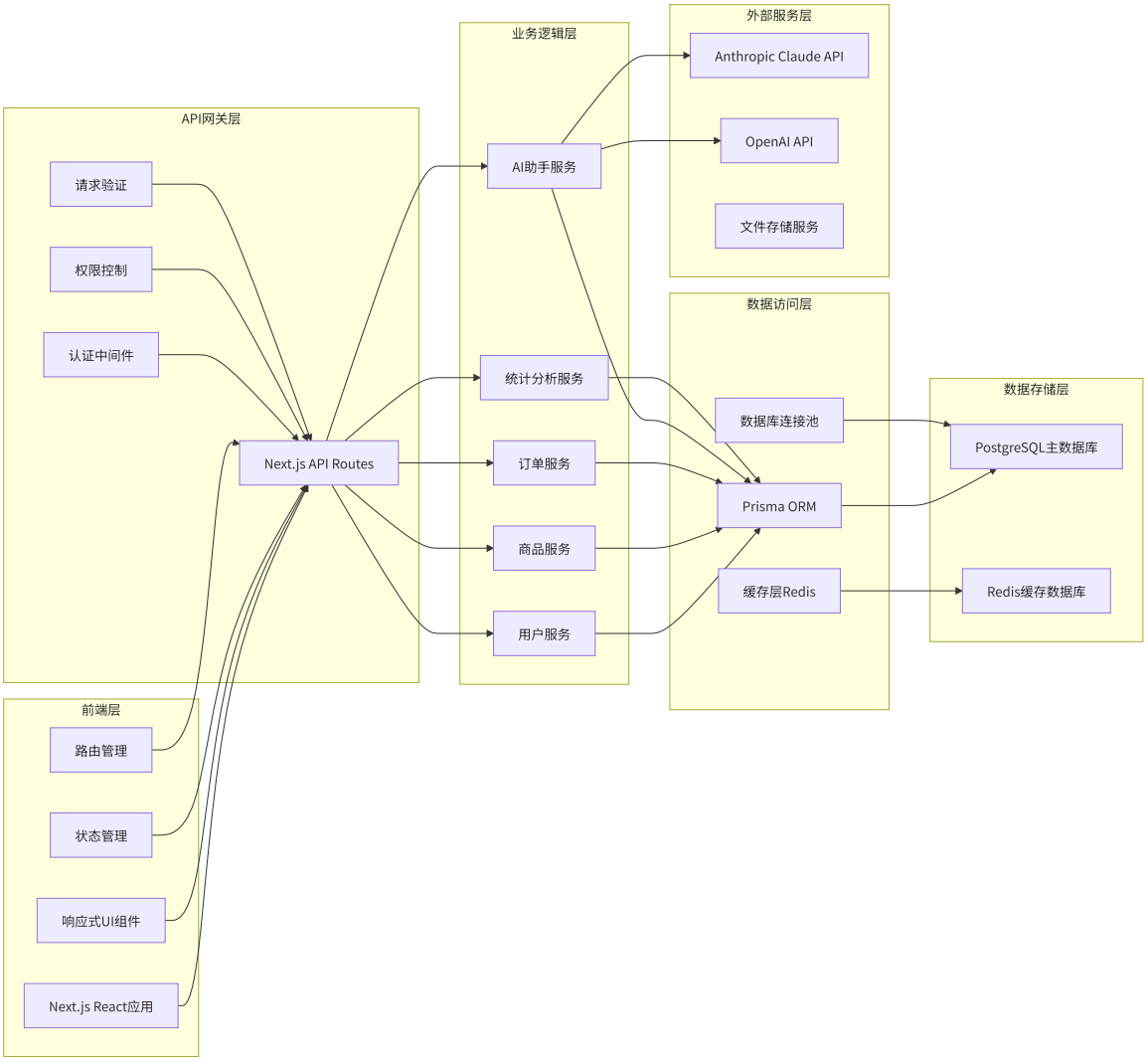

图3-3 系统总体架构图

3.2.1.2 技术架构特点

前端架构特点：

- **Server-Side Rendering (SSR)**：提升首屏加载速度和SEO效果
- **Client-Side Rendering (CSR)**：动态页面的流畅交互体验
- **Static Site Generation (SSG)**：静态页面的极致性能
- **Incremental Static Regeneration (ISR)**：静态页面的增量更新

后端架构特点：

- **API-First设计**：前后端分离，支持多端复用
- **无状态设计**：基于JWT的无状态认证，支持水平扩展
- **微服务化**：按业务领域划分服务，便于维护和扩展
- **函数式编程**：利用Next.js API Routes的函数式特性

3.2.2 核心模块设计

3.2.2.1 用户认证与权限模块

模块职责：

- 用户注册、登录、注销
- JWT令牌生成和验证
- 权限检查和访问控制
- 密码安全管理

核心流程设计：

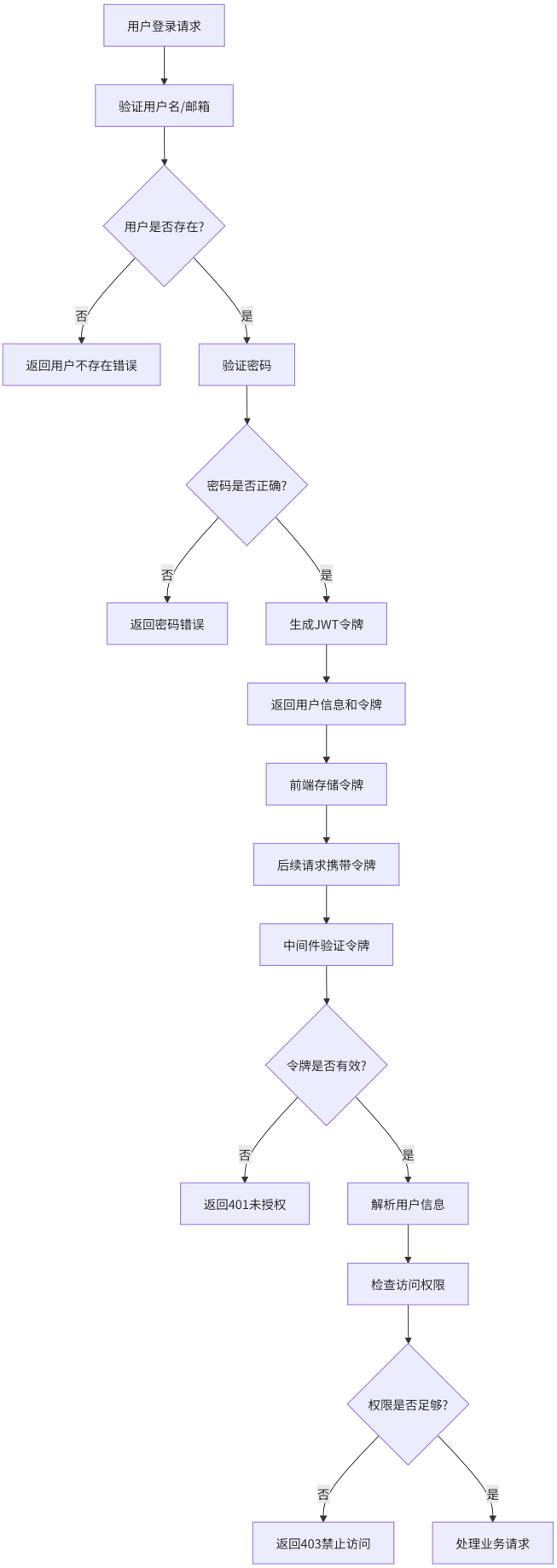
用户登录请求
验证用户名/邮箱
用户是否存在?
否
是
返回用户不存在错误
验证密码
密码是否正确?
否
是
返回密码错误
生成JWT令牌
返回用户信息和令牌
前端存储令牌
后续请求携带令牌
中间件验证令牌
令牌是否有效?
否
是
返回401未授权
解析用户信息
检查访问权限
权限是否足够?
否
是
返回403禁止访问
处理业务请求

图3-4 用户认证流程图

技术实现细节：

```
// JWT令牌生成
export function generateToken(user: User): string {
  const payload = {
    userId: user.id,
    email: user.email,
    isAdmin: user.isAdmin,
    iat: Math.floor(Date.now() / 1000),
    exp: Math.floor(Date.now() / 1000) + (7 * 24 * 60 * 60) // 7天过期
  }

  return jwt.sign(payload, process.env.JWT_SECRET!)
}

// 权限验证中间件
export function requireAuth(request: NextRequest): TokenPayload {
  const authHeader = request.headers.get('authorization')
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    throw new AuthError('请先登录')
  }

  const token = authHeader.substring(7)
  const decoded = verifyToken(token)
  if (!decoded) {
    throw new AuthError('登录已过期')
  }

  return decoded
}

// 管理员权限检查
export async function requireAdmin(request: NextRequest):
Promise<void> {
  const decoded = requireAuth(request)
  if (!decoded.isAdmin) {
    throw new AuthError('权限不足')
  }
}
```

3.2.2.2 商品管理模块

模块职责：

- 配件分类管理
- 商品信息CRUD操作
- 库存管理和控制

- 批量数据导入导出
- 商品搜索和筛选

商品查询流程设计：

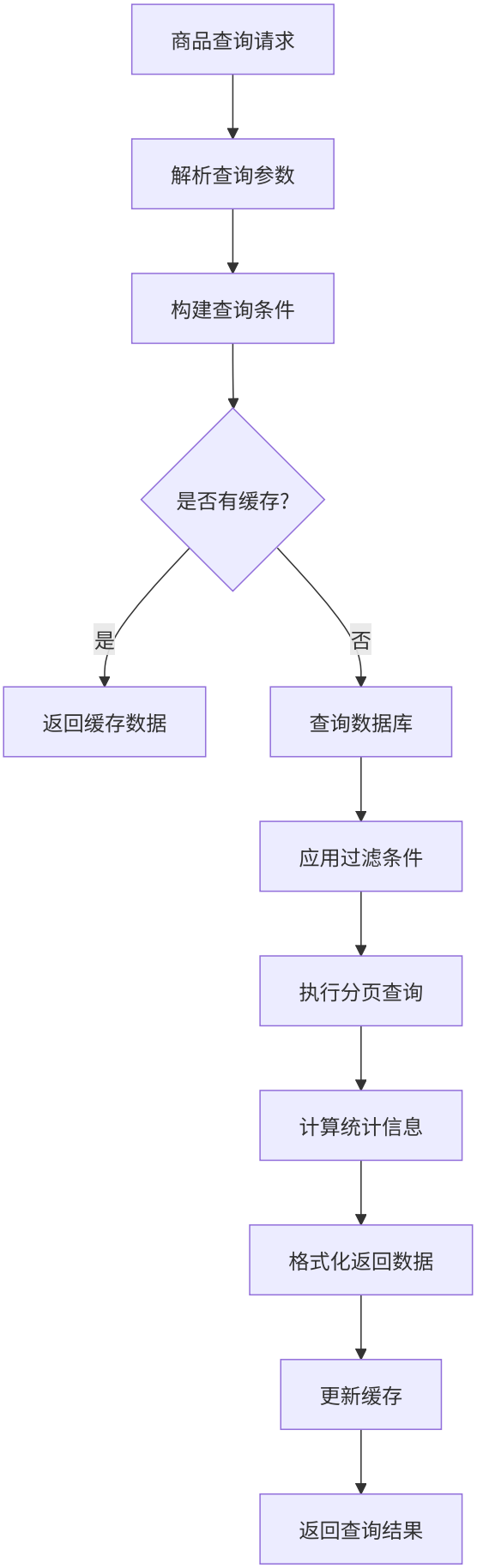
商品查询请求
解析查询参数
构建查询条件
是否有缓存?
是
否
返回缓存数据
查询数据库
应用过滤条件
执行分页查询
计算统计信息
格式化返回数据
更新缓存
返回查询结果

图3-5 商品查询流程图

核心服务实现：

```
export class ComponentService {
  // 智能商品查询
  static async queryComponents(params: ComponentQueryParams):
Promise<ComponentQueryResult> {
    const {
      type, typeId, brand, minPrice, maxPrice, search, page = 1, limit
= 12
    } = params

    // 构建动态查询条件
    const where: any = {}

    // 配件类型筛选
    if (typeId) {
      where.componentTypeId = typeId
    } else if (type) {
      const componentType = await prisma.componentType.findFirst({
        where: { name: { contains: type, mode: 'insensitive' } }
      })
      if (componentType) {
        where.componentTypeId = componentType.id
      }
    }

    // 品牌筛选
    if (brand) {
      where.brand = { contains: brand, mode: 'insensitive' }
    }

    // 价格范围筛选
    if (minPrice || maxPrice) {
      where.price = {}
      if (minPrice) where.price.gte = minPrice
      if (maxPrice) where.price.lte = maxPrice
    }

    // 全文搜索
    if (search) {
      where.OR = [
        { name: { contains: search, mode: 'insensitive' } },
        { brand: { contains: search, mode: 'insensitive' } },
        { model: { contains: search, mode: 'insensitive' } },
        { description: { contains: search, mode: 'insensitive' } }
      ]
    }

    // 并行查询数据和总数
    const [components, total] = await Promise.all([
```

```
      prisma.component.findMany({
        where,
        include: { componentType: true },
        orderBy: { createdAt: 'desc' },
        skip: (page - 1) * limit,
        take: limit
      }),
      prisma.component.count({ where })
    ])

    return {
      components: components.map(formatComponent),
      pagination: {
        page, limit, total,
        totalPages: Math.ceil(total / limit),
        hasNext: page < Math.ceil(total / limit),
        hasPrev: page > 1
      }
    }
  }

  // 批量导入商品
  static async batchImport(components: ComponentImportData[]):
Promise<BatchImportResult> {
    const results = []
    let successful = 0
    let failed = 0
    let newTypesCreated = 0

    for (const item of components) {
      try {
        // 检查或创建配件类型
        let componentType = await prisma.componentType.findFirst({
          where: { name: item.typeName }
        })

        if (!componentType) {
          componentType = await prisma.componentType.create({
            data: { name: item.typeName, description:
`${item.typeName}配件` }
          })
          newTypesCreated++
        }

        // 创建配件
        const component = await prisma.component.create({
          data: {
            name: item.name,
            brand: item.brand,
            model: item.model,
            price: item.price,
            description: item.description,
            imageUrl: item.imageUrl,
            stock: item.stock,
```

```
            specifications: item.specifications,
            componentTypeId: componentType.id
          }
        })

        results.push({ success: true, item, component })
        successful++
      } catch (error) {
        results.push({ success: false, item, error: error.message })
        failed++
      }
    }

    return {
      summary: { successful, failed, total: components.length,
newTypesCreated },
      results
    }
  }
}
```

3.2.2.3 订单管理模块

模块职责：

- 购物车管理
- 订单创建和处理
- 库存扣减和恢复
- 订单状态管理
- 订单统计分析

订单处理流程设计：

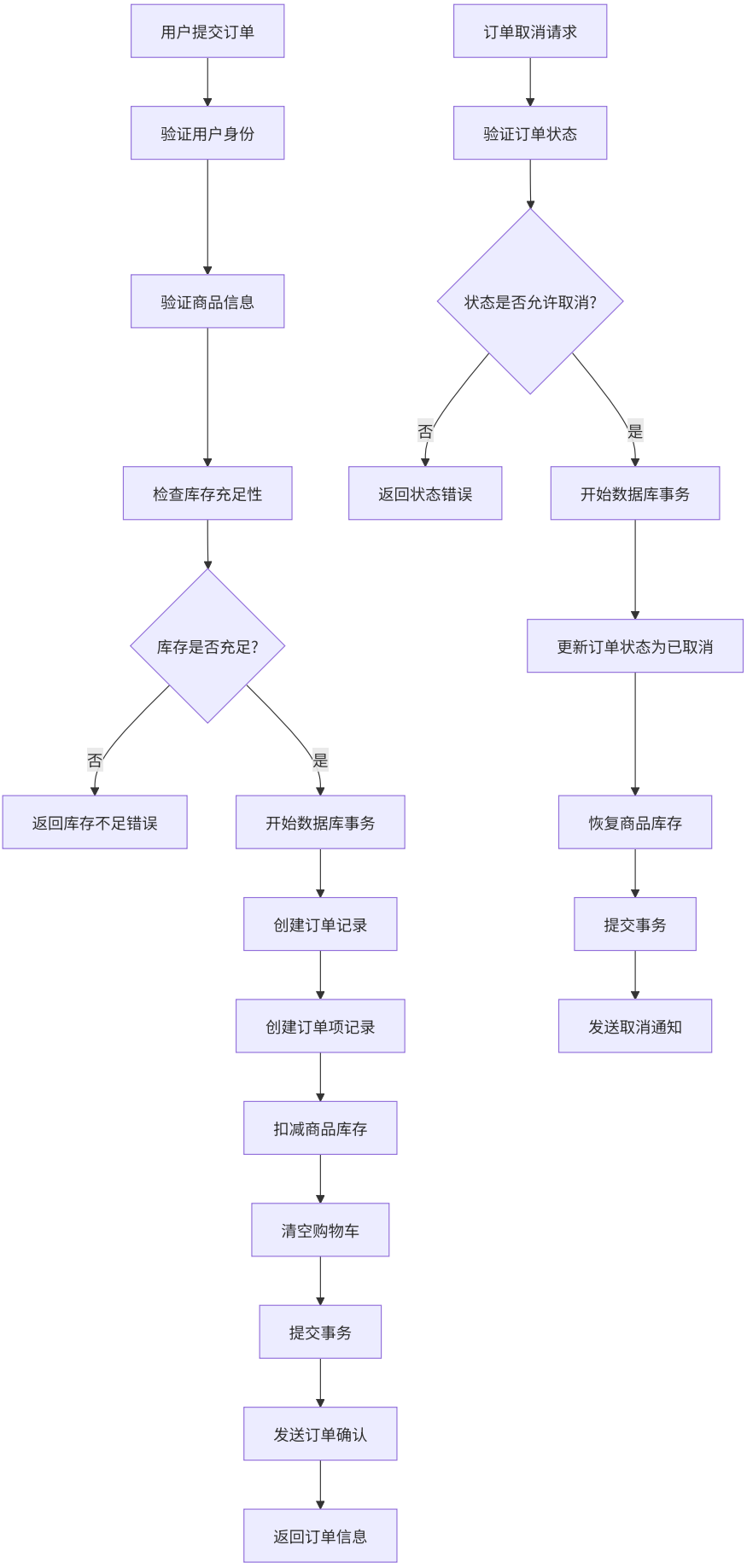
用户提交订单
验证用户身份
验证商品信息
检查库存充足性
库存是否充足?
否
返回库存不足错误
是
开始数据库事务
创建订单记录
创建订单项记录
扣减商品库存
清空购物车
提交事务
发送订单确认
返回订单信息
订单取消请求
验证订单状态
状态是否允许取消?
否
返回状态错误
是
开始数据库事务
更新订单状态为已取消
恢复商品库存
提交事务
发送取消通知

图3-6 订单处理流程图

3.2.2.4 AI助手模块

模块职责：

- AI对话管理
- 工具调用处理
- 流式响应处理
- 对话历史存储
- 上下文管理

AI交互流程设计：

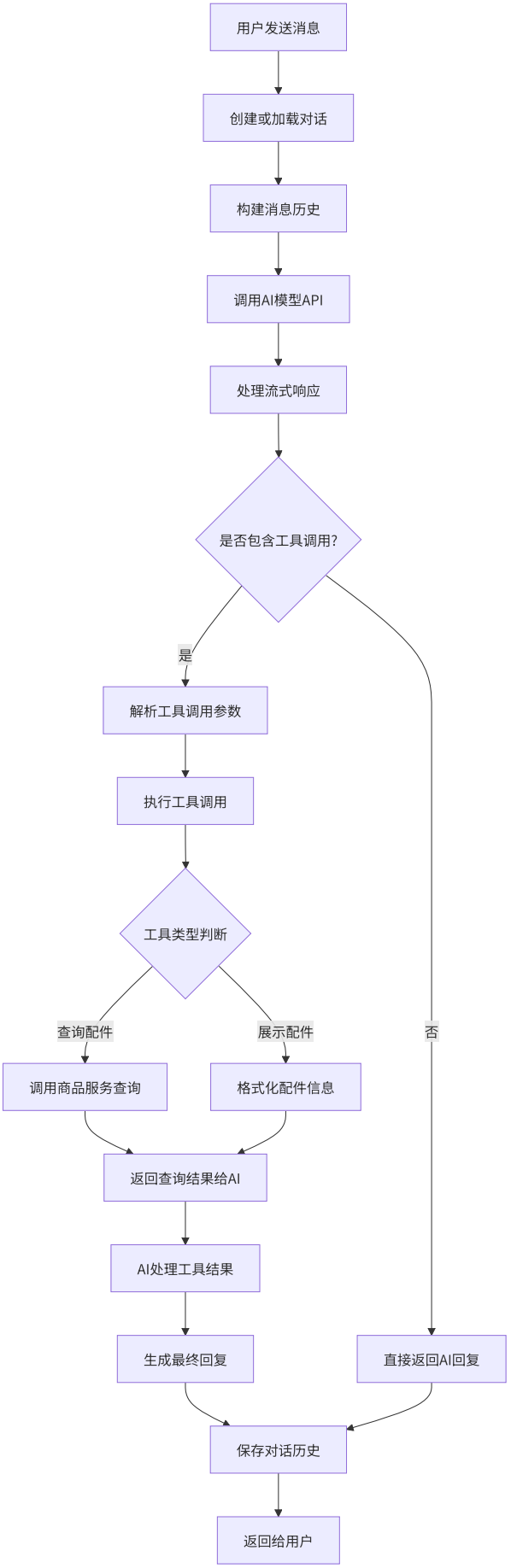
用户发送消息
创建或加载对话
构建消息历史
调用AI模型API
处理流式响应
是否包含工具调用?
是
否
解析工具调用参数
执行工具调用
工具类型判断
查询配件
展示配件
调用商品服务查询
格式化配件信息
返回查询结果给AI
AI处理工具结果
生成最终回复
直接返回AI回复
保存对话历史
返回给用户

图3-7 AI交互流程图

AI助手核心实现：

```
// 见5.1.2
```

3.2.2.5 数据统计分析模块

模块职责：

- 用户消费统计
- 商品销售统计
- 趋势分析
- 数据可视化
- 报表生成

统计分析流程设计：

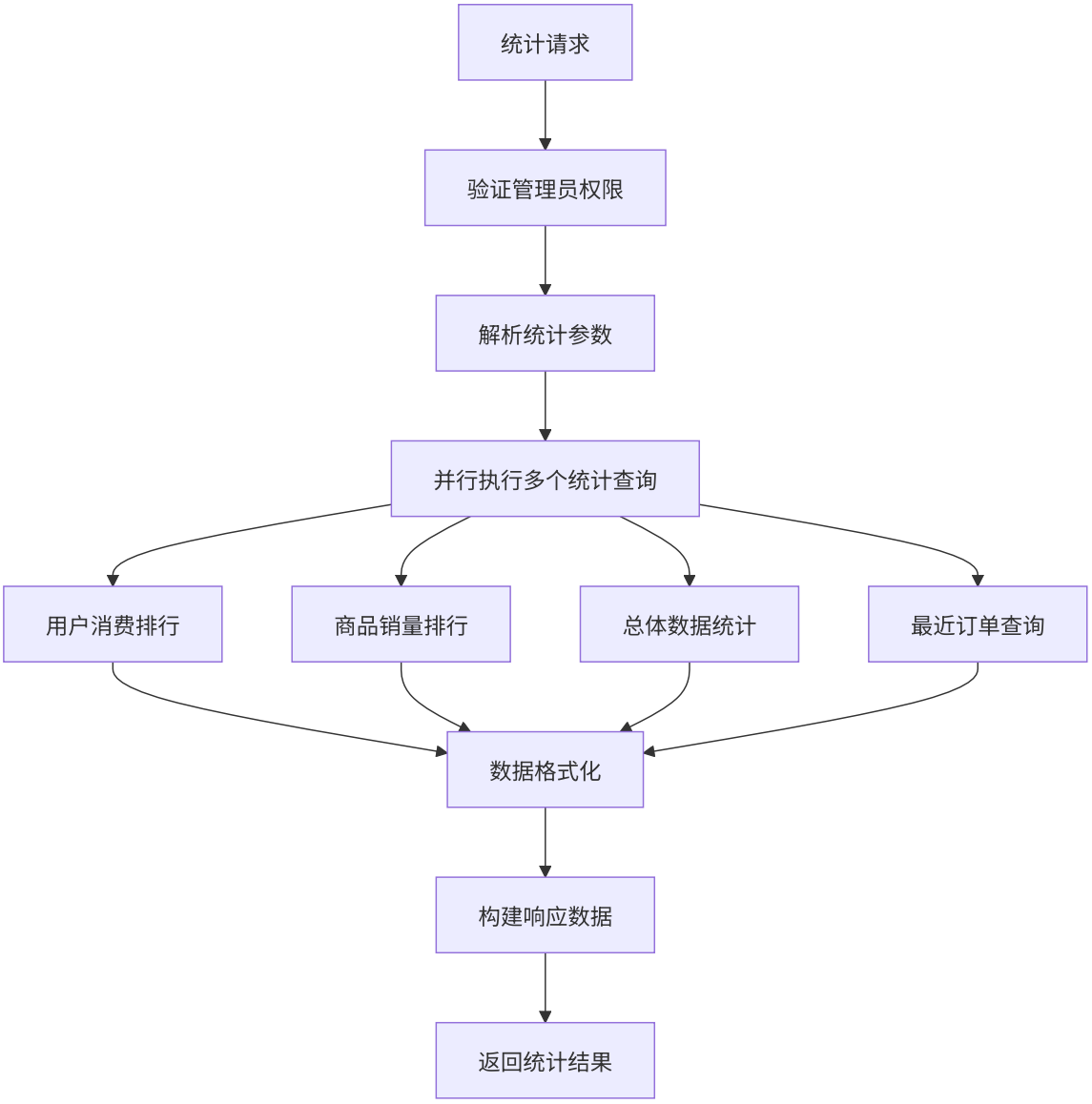

图3-8 数据统计流程图

统计服务实现：

```
export class StatsService {
  // 管理员数据统计
  static async getAdminStats(): Promise<AdminStatsData> {
    // 并行执行多个统计查询，提升性能
    const [
      topUsersBySpending,
      topComponentsBySales,
      overviewStats,
      recentOrders
    ] = await Promise.all([
      this.getTopUsersBySpending(),
      this.getTopComponentsBySales(),
      this.getOverviewStats(),
      this.getRecentOrders()
```

```
    ])

    return {
      overview: overviewStats,
      topUsers: topUsersBySpending,
      topComponents: topComponentsBySales,
      recentOrders
    }
  }

  // 用户消费排行榜
  private static async getTopUsersBySpending() {
    const users = await prisma.user.findMany({
      where: {
        isAdmin: false,
        orders: { some: {} }
      },
      select: {
        id: true,
        username: true,
        name: true,
        email: true,
        orders: {
          where: {
            status: { in: ['CONFIRMED', 'SHIPPED', 'DELIVERED'] }
          },
          select: { totalAmount: true }
        }
      },
      take: 50  // 先取更多数据，再排序截取
    })

    return users
      .map(user => ({
        id: user.id,
        username: user.username,
        name: user.name || user.username,
        email: user.email,
        totalSpending: user.orders.reduce((sum, order) => sum +
order.totalAmount, 0),
        orderCount: user.orders.length
      }))
      .sort((a, b) => b.totalSpending - a.totalSpending)
      .slice(0, 10)
  }

  // 商品销量排行榜
  private static async getTopComponentsBySales() {
    const components = await prisma.component.findMany({
      select: {
        id: true,
        name: true,
        brand: true,
        price: true,
```

```
        componentType: { select: { name: true } },
        orderItems: {
          where: {
            order: {
              status: { in: ['CONFIRMED', 'SHIPPED', 'DELIVERED'] }
            }
          },
          select: { quantity: true }
        }
      }
    })

    return components
      .map(component => ({
        id: component.id,
        name: component.name,
        brand: component.brand,
        price: component.price,
        typeName: component.componentType.name,
        totalSales: component.orderItems.reduce((sum, item) => sum +
item.quantity, 0),
        totalRevenue: component.orderItems.reduce((sum, item) => sum +
(item.quantity * component.price), 0)
      }))
      .sort((a, b) => b.totalSales - a.totalSales)
      .slice(0, 10)
  }

  // 总体统计数据
  private static async getOverviewStats() {
    const [totalUsers, totalOrders, totalComponents, totalRevenue] =
await Promise.all([
      prisma.user.count({ where: { isAdmin: false } }),
      prisma.order.count(),
      prisma.component.count(),
      prisma.order.aggregate({
        where: {
          status: { in: ['CONFIRMED', 'SHIPPED', 'DELIVERED'] }
        },
        _sum: { totalAmount: true }
      })
    ])

    return {
      totalUsers,
      totalOrders,
      totalComponents,
      totalRevenue: totalRevenue._sum.totalAmount || 0
    }
  }
}
```

3.2.3 模块间通信设计

3.2.3.1 API设计规范

RESTful API设计原则：

- **资源命名**：使用名词表示资源，复数形式
- **HTTP方法**：GET（查询）、POST（创建）、PUT（更新）、DELETE（删除）
- **状态码**：200（成功）、201（创建成功）、400（请求错误）、401（未授权）、403（禁止访问）、404（未找到）、500（服务器错误）
- **统一响应格式**：成功和错误都有统一的数据结构

API路由设计：

```
/api/auth/
  POST /login           # 用户登录
  POST /register        # 用户注册

/api/components/
  GET /                 # 获取配件列表
  GET /:id              # 获取配件详情
  POST /batch           # 批量导入配件

/api/orders/
  GET /                 # 获取订单列表
  POST /                # 创建订单
  GET /:id              # 获取订单详情
  PUT /:id              # 更新订单状态

/api/cart/
  GET /                 # 获取购物车
  POST /                # 添加到购物车
  PUT /:id              # 更新购物车项
  DELETE /:id           # 删除购物车项

/api/ai/
  POST /                # AI对话
  GET /conversations # 获取对话列表
  GET /conversations/:id # 获取对话详情

/api/admin/
  GET /stats            # 管理员统计数据
  GET /orders           # 管理员订单管理
  GET /components       # 管理员商品管理
```

这种模块化设计确保了系统的高内聚、低耦合特性，每个模块都有明确的职责边界，模块间通过标准化的API进行通信，便于维护、测试和扩展。同时，完善的错误处理和数据验证机制保证了系统的稳定性和安全性。

第四章 系统实现

4.1 项目介绍

4.1.1 项目整体架构

本在线电脑DIY系统采用现代化的全栈Web开发架构，基于Next.js 15.3.4框架构建，实现了前后端一体化的开发模式。系统不仅满足了课程设计的基本要求，更在此基础上集成了AI智能助手功能，为用户提供专业的配件咨询和推荐服务。

系统采用三层架构设计：

- **表现层（Presentation Layer）**：基于React 19.0.0和Tailwind CSS 4.x构建的现代化用户界面
- **业务逻辑层（Business Logic Layer）**：Next.js API Routes实现的RESTful API服务
- **数据访问层（Data Access Layer）**：Prisma ORM + PostgreSQL数据库

4.1.2 项目目录结构详解

基于文件系统的分析，本项目采用Next.js 13+的App Router架构，具有清晰的模块化组织结构：

```
pc-diy-store/
├── app/                              # Next.js 13+ App Router核心目录
│   ├── globals.css                  # 全局样式文件
│   ├── layout.tsx                   # 根布局组件
│   ├── page.tsx                     # 首页组件
│   ├── favicon.ico                  # 网站图标
│   │
│   ├── api/                         # API路由目录
│   │   ├── auth/                    # 认证相关API
│   │   │   ├── login/route.ts       # 用户登录接口
│   │   │   └── register/route.ts    # 用户注册接口
│   │   ├── admin/                   # 管理员专用API
│   │   │   ├── components/route.ts  # 配件管理接口
│   │   │   ├── orders/route.ts      # 订单管理接口
│   │   │   └── stats/route.ts       # 统计数据接口
│   │   ├── ai/                      # AI助手API
│   │   │   ├── route.ts             # AI对话接口
│   │   │   └── conversations/       # 对话管理
│   │   │       ├── route.ts         # 对话列表接口
```

```
|   |   |       └── [id]/route.ts # 单个对话接口
|   |   ├── components/           # 配件相关API
|   |   |   ├── route.ts          # 配件查询/创建接口
|   |   |   ├── [id]/route.ts     # 单个配件操作接口
|   |   |   └── batch/route.ts    # 批量操作接口
|   |   ├── cart/                 # 购物车API
|   |   |   ├── route.ts          # 购物车操作接口
|   |   |   └── [id]/route.ts     # 购物车项操作接口
|   |   ├── orders/               # 订单API
|   |   |   ├── route.ts          # 订单操作接口
|   |   |   └── [id]/route.ts     # 单个订单接口
|   |   ├── user/                 # 用户API
|   |   |   ├── profile/route.ts  # 用户资料接口
|   |   |   ├── stats/route.ts    # 用户统计接口
|   |   |   └── change-password/route.ts # 密码修改接口
|   |   └── component-types/route.ts # 配件类型接口
|   |
|   ├── admin/                    # 管理后台页面
|   |   ├── page.tsx              # 管理后台首页
|   |   ├── components/page.tsx   # 配件管理页面
|   |   └── orders/page.tsx       # 订单管理页面
|   |
|   ├── ai-assistant/             # AI助手页面
|   |   └── page.tsx              # AI对话界面
|   |
|   ├── build/                    # 装机配置页面
|   |   └── page.tsx              # 装机配置界面
|   |
|   ├── cart/                     # 购物车页面
|   |   └── page.tsx              # 购物车界面
|   |
|   ├── components/               # 配件浏览页面
|   |   ├── page.tsx              # 配件列表页面
|   |   └── [id]/page.tsx         # 配件详情页面
|   |
|   ├── login/                    # 用户登录页面
|   |   └── page.tsx              # 登录界面
|   |
|   ├── register/                 # 用户注册页面
|   |   └── page.tsx              # 注册界面
|   |
|   ├── orders/                   # 订单管理页面
|   |   ├── page.tsx              # 订单列表页面
|   |   └── [id]/page.tsx         # 订单详情页面
|   |
|   └── profile/                  # 个人中心页面
|       └── page.tsx              # 用户资料界面
|
├── components/                   # 可复用React组件
|   ├── Navbar.tsx                # 导航栏组件
|   ├── ComponentCard.tsx         # 配件卡片组件
|   ├── AddToCartButton.tsx       # 添加购物车按钮组件
|   └── admin/                    # 管理员专用组件
|       └── AdminAuth.tsx         # 管理员认证组件
```

```
│
├── lib/                              # 工具库和服务
│   ├── prisma.ts                     # Prisma数据库客户端
│   ├── auth.ts                       # 认证工具函数
│   ├── admin-auth.ts                 # 管理员认证工具
│   ├── ai-assistant.ts               # AI助手核心逻辑（Anthropic Claude）
│   ├── ai-assistant-openai.ts        # AI助手备用实现（OpenAI）
│   ├── markdown-it.ts                # Markdown渲染工具
│   └── services/                     # 业务服务层
│       ├── component-service.ts      # 配件业务逻辑
│       └── conversation-service.ts # 对话管理业务逻辑
│
├── prisma/                           # 数据库相关
│   ├── schema.prisma                 # 数据库模式定义
│   └── migrations/                   # 数据库迁移文件
│       ├── migration_lock.toml       # 迁移锁定文件
│       ├── 20250621121352_init/      # 初始数据库结构
│       ├── 20250622010446_add_cart_items/ # 购物车功能
│       └── 20250623095809_add_ai_conversations_json/ # AI对话功能
│
├── public/                           # 静态资源
│   ├── next.svg                      # Next.js logo
│   ├── vercel.svg                    # Vercel logo
│   └── [其他静态文件]
│
├── package.json                      # 项目配置和依赖
├── next.config.ts                    # Next.js配置
├── tsconfig.json                     # TypeScript配置
├── postcss.config.mjs                # PostCSS配置
├── bun.lock                          # Bun包管理器锁定文件
└── seed.ts                           # 数据库种子数据
```

4.1.3 核心文件功能详解

4.1.3.1 应用入口文件

app/layout.tsx - 根布局组件

```
export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="zh-CN">
      <body className={`${geistSans.variable} ${geistMono.variable}
antialiased`}>
        <Navbar />
        <main style={{height: "calc(100vh - 64px)", overflowY:
"auto"}}>
          {children}
```

```
          </main>
        </body>
      </html>
    );
  }
```

该文件是整个应用的根布局，负责：

- 设置HTML文档的基本结构和语言
- 加载全局字体和样式
- 渲染全局导航栏组件
- 为页面内容提供滚动容器

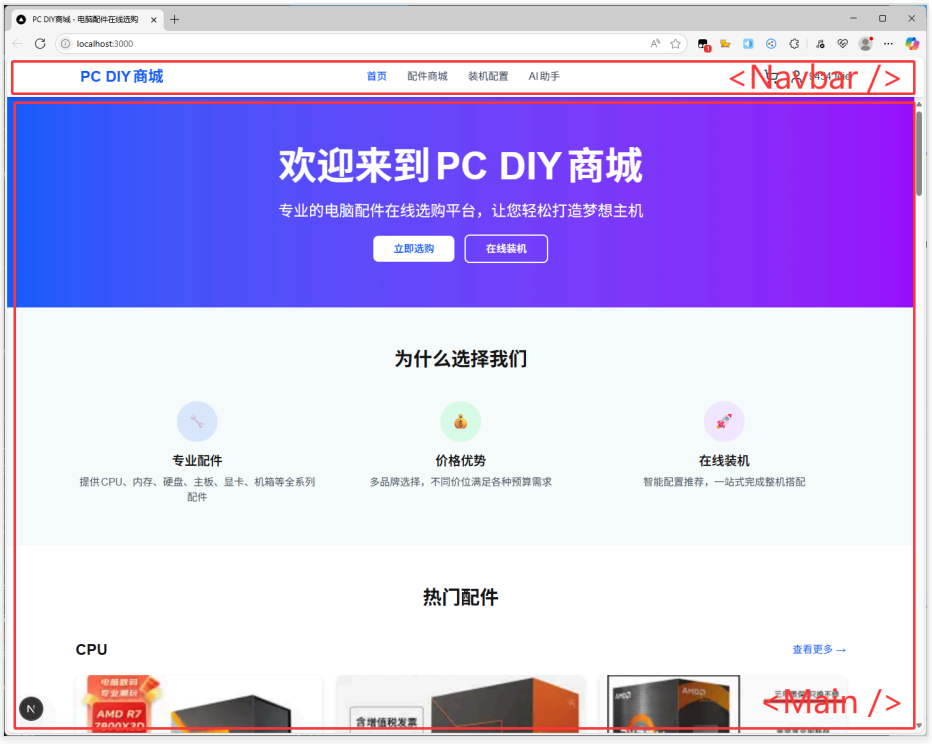

图4-9 Layout 说明

app/page.tsx - 系统首页

首页实现了项目要求的企业介绍和配件展示功能：

- 企业品牌宣传区域
- 各类配件的分类展示
- 响应式设计适配多种设备
- 无需登录即可浏览所有配件信息

4.1.3.2 API路由系统

认证系统 (app/api/auth/)

- `login/route.ts`: 实现用户登录验证，包括密码加密验证和JWT Token生成
- `register/route.ts`: 处理用户注册，包括数据验证、密码加密存储

配件管理系统 (app/api/components/)

- `route.ts`: 提供配件的CRUD操作，支持分页查询、条件筛选
- `[id]/route.ts`: 单个配件的详细操作
- `batch/route.ts`: 批量导入配件数据，支持CSV和JSON格式

AI助手系统 (app/api/ai/)

- `route.ts`: 核心AI对话接口，实现流式响应
- `conversations/`: 对话历史管理，支持创建、查询、删除对话

4.1.3.3 前端页面组件

配件浏览系统 (app/components/)

```
// 配件列表页面支持高级筛选功能
const [filters, setFilters] = useState({
  search: '',
  type: '',
  brand: '',
  minPrice: '',
  maxPrice: ''
})
```

实现功能：

- 多条件筛选（类型、品牌、价格区间、关键词）
- 分页展示
- 响应式网格布局
- 实时搜索

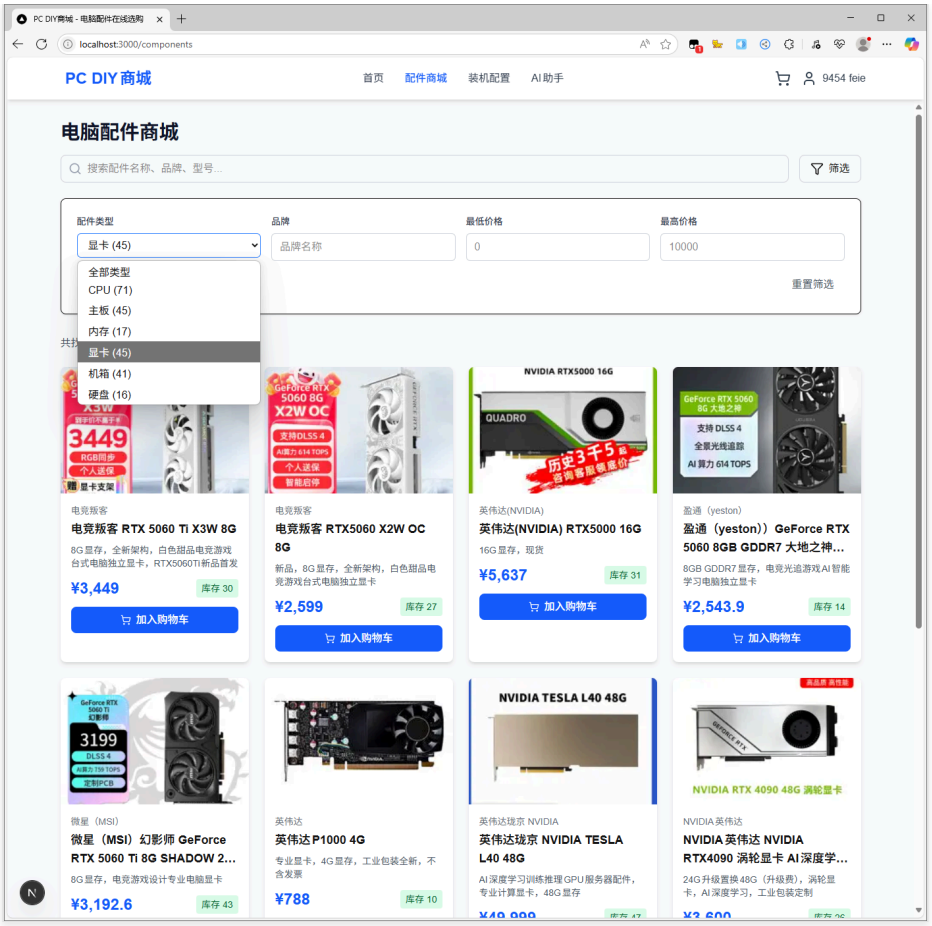

图4-10 配件商城 - 支持筛选与搜索

装机配置系统 (app/build/)

独特的装机配置功能，允许用户：

- 从每种配件类型中选择一个配件
- 实时计算总价
- 配置进度跟踪
- 一键添加到购物车

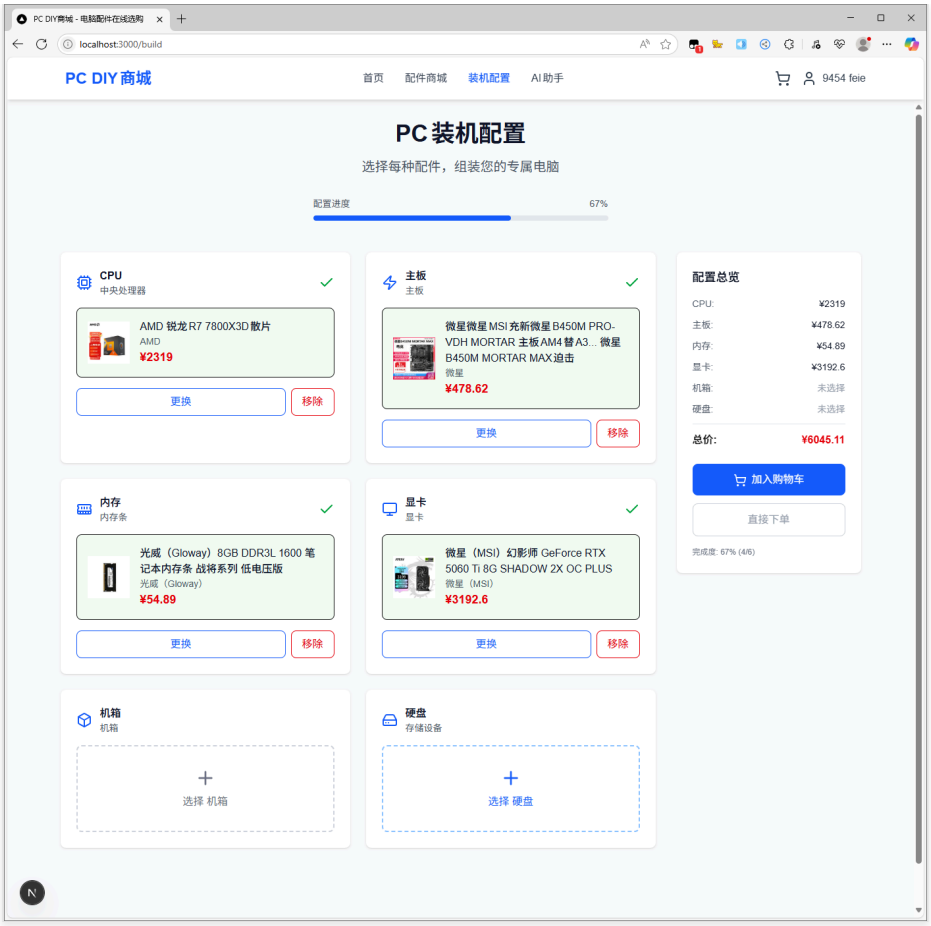

图4-11 整机配置界面

AI助手界面 (app/ai-assistant/)

创新的AI对话功能：

- 对话历史侧边栏
- 流式打字机效果
- 配件卡片展示
- 多轮对话支持

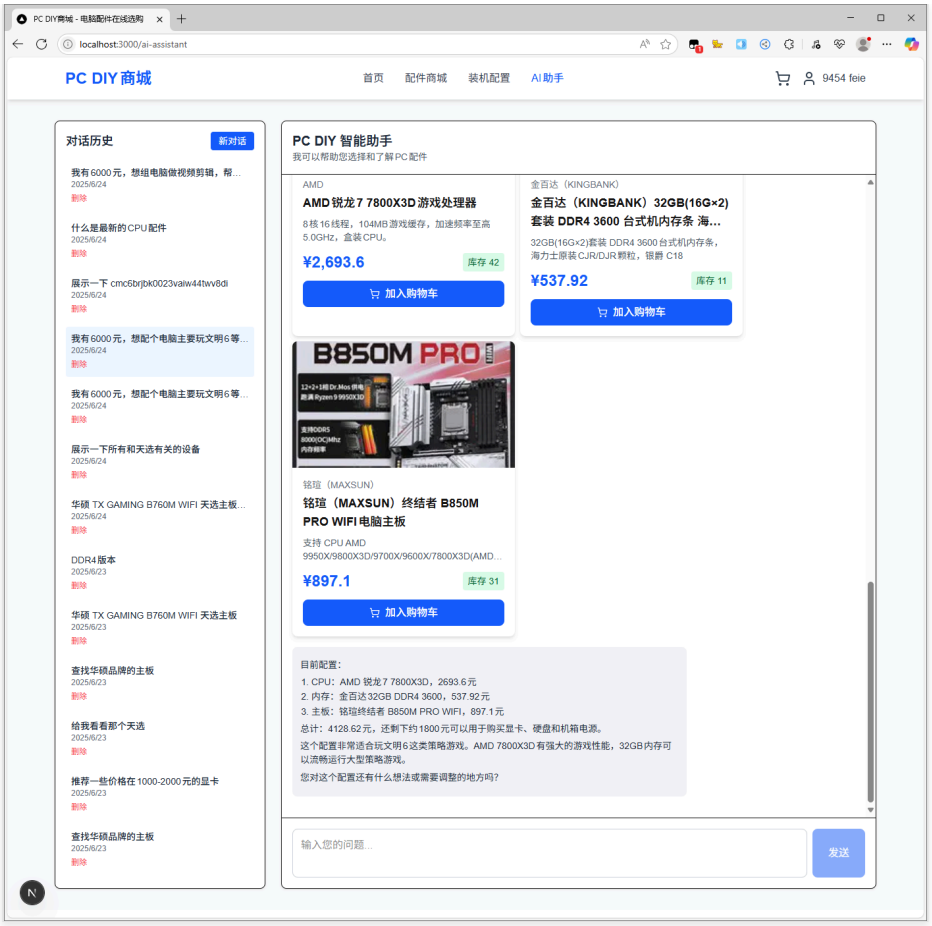

图4-12 AI 助手协助分析需求、搜索配件

4.1.3.4 业务逻辑层

lib/services/component-service.ts - 配件服务

核心业务逻辑封装：

```
export class ComponentService {
  static async queryComponents(params: ComponentQueryParams):
Promise<ComponentQueryResult> {
    // 复杂查询逻辑
    const where: any = {}

    // 类型筛选
    if (typeId) {
      where.componentTypeId = typeId
    } else if (type) {
      const componentType = await prisma.componentType.findFirst({
        where: { name: { contains: type, mode: 'insensitive' } }
      })
      if (componentType) {
        where.componentTypeId = componentType.id
      }
    }
```

```
      // 价格范围筛选
      if (minPrice || maxPrice) {
        where.price = {}
        if (minPrice) where.price.gte = minPrice
        if (maxPrice) where.price.lte = maxPrice
      }

      // 多字段模糊搜索
      if (search) {
        where.OR = [
          { name: { contains: search, mode: 'insensitive' } },
          { brand: { contains: search, mode: 'insensitive' } },
          { model: { contains: search, mode: 'insensitive' } },
          { description: { contains: search, mode: 'insensitive' } }
        ]
      }

      // 并行查询数据和总数
      const [components, total] = await Promise.all([
        prisma.component.findMany({
          where,
          include: { componentType: true },
          orderBy: { createdAt: 'desc' },
          skip: (page - 1) * limit,
          take: limit
        }),
        prisma.component.count({ where })
      ])

      return {
        components: formatComponents(components),
        pagination: calculatePagination(page, limit, total)
      }
    }
  }
```

lib/services/conversation-service.ts - 对话服务

AI对话管理的完整实现:

- 对话创建和初始化
- 消息历史的JSON存储和检索
- 对话标题自动生成
- 对话删除和清理

4.1.3.5 AI助手核心实现

lib/ai-assistant.ts - AI助手引擎

本系统的最大亮点是完整实现的AI助手功能：

```
export class AIClient {
  private anthropic: Anthropic;
  private tools: Tool[] = [
    {
      name: "query-components",
      description: "查询PC配件信息，支持按类型、品牌、价格范围、关键词等条
件搜索",
      input_schema: {
        type: "object",
        properties: {
          type: { type: "string", enum: ["CPU", "内存", "硬盘", "主
板", "显卡", "机箱"] },
          brand: { type: "string", description: "品牌名称" },
          minPrice: { type: "number", description: "最低价格" },
          maxPrice: { type: "number", description: "最高价格" },
          search: { type: "string", description: "搜索关键词" }
        }
      }
    },
    {
      name: "show-components",
      description: "向用户以卡片形式展示一个或多个具体型号的配件",
      input_schema: {
        type: "object",
        properties: {
          component_ids: {
            type: "array",
            items: { type: "string" },
            description: "要展示的配件ID数组"
          }
        }
      }
    }
  ];

  async *processQuery(query: string, userId: string, conversationId?:
string): AsyncGenerator<string, void, unknown> {
    // 流式AI响应处理
    // 工具调用处理
    // 对话历史管理
    // 具体分析见 5.1.2
  }
}
```

AI助手的核心特性：

1. **工具调用能力**：AI可以主动调用系统API查询实时数据
2. **流式响应**：实现打字机效果的实时对话体验
3. **多轮对话**：支持上下文保持的连续对话

4.1.3.6 数据库层

prisma/schema.prisma - 数据模型定义

完整的数据库模式设计，包括：

- 用户和权限管理
- 配件和分类管理
- 购物车和订单系统
- AI对话历史存储

4.2 系统功能实现

4.2.1 用户界面展示与功能实现

4.2.1.1 系统首页实现

功能概述：

系统首页完全满足课程设计要求，实现了企业介绍和配件展示功能，无需登录即可浏览所有配件信息。

核心实现代码：

```
// app/page.tsx - 首页实现
export default async function Home() {
  // 获取每种配件类型的前几个商品
  const componentTypes = await prisma.componentType.findMany({
    include: {
      components: {
        take: 6,
        orderBy: { createdAt: 'desc' }
      }
    }
  })

  return (
    <div className="min-h-screen bg-gray-50">
      {/* Hero Section - 企业介绍 */}
      <section className="bg-gradient-to-r from-blue-600 to-purple-600
text-white py-20">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 text-
center">
          <h1 className="text-4xl md:text-6xl font-bold mb-6">
```

```
            欢迎来到PC DIY商城
          </h1>
          <p className="text-xl md:text-2xl mb-8">
            专业的电脑配件在线选购平台，让您轻松打造梦想主机
          </p>
        </div>
      </section>

      {/* 配件分类展示 */}
      <section className="py-16 bg-white">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <h2 className="text-3xl font-bold text-center mb-12">热门配
件</h2>
          {componentTypes.map((type) => (
            <div key={type.id} className="mb-12">
              <div className="flex justify-between items-center mb-6">
                <h3 className="text-2xl font-semibold">{type.name}
</h3>
                <Link href={`/components?type=${type.id}`}>
                  查看更多 →
                </Link>
              </div>
              <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-
cols-3 gap-6">
                {type.components.map((component) => (
                  <ComponentCard key={component.id} component=
{component} />
                ))}
              </div>
            </div>
          ))}
        </div>
      </section>
    </div>
  )
}
```

页面功能特点：

1. **企业品牌展示**：渐变背景的Hero区域，突出品牌形象
2. **配件分类展示**：按照CPU、内存、硬盘、主板、显卡、机箱分类展示
3. **响应式设计**：适配桌面、平板、手机等多种设备
4. **快速导航**：每个分类都有"查看更多"链接，方便用户深入浏览

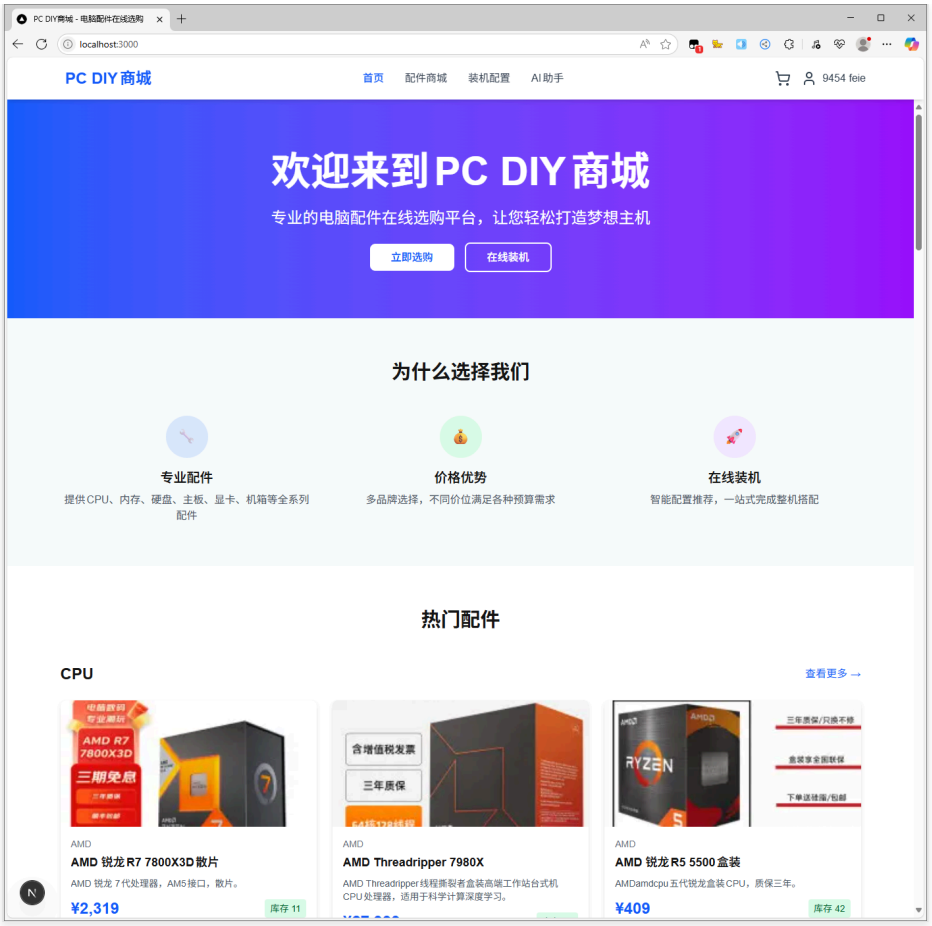
PC DIY商城 - 电脑配件在线选购
localhost:3000
PC DIY 商城
首页
配件商城
装机配置
AI 助手
9454 feie
欢迎来到PC DIY商城
专业的电脑配件在线选购平台，让您轻松打造梦想主机
立即选购
在线装机
为什么选择我们
专业配件
提供CPU、内存、硬盘、主板、显卡、机箱等全系列配件
价格优势
多品牌选择，不同价位满足各种预算需求
在线装机
智能配置推荐，一站式完成整机搭配
热门配件
CPU
查看更多 →
AMD R7 7800X3D
三期免息
含增值税发票
三年质保
AMD
AMD 锐龙R7 7800X3D散片
AMD 锐龙 7 代处理器，AM5接口，散片。
¥2,319
库存 11
AMD
AMD Threadripper 7980X
AMD Threadripper线程撕裂者盒装高端工作站台式机CPU处理器，适用于科学计算深度学习。
AMD
AMD 锐龙R5 5500盒装
AMDamdcpu五代锐龙盒装CPU，质保三年。
¥409
库存 42

图4-13 系统首页 - 上

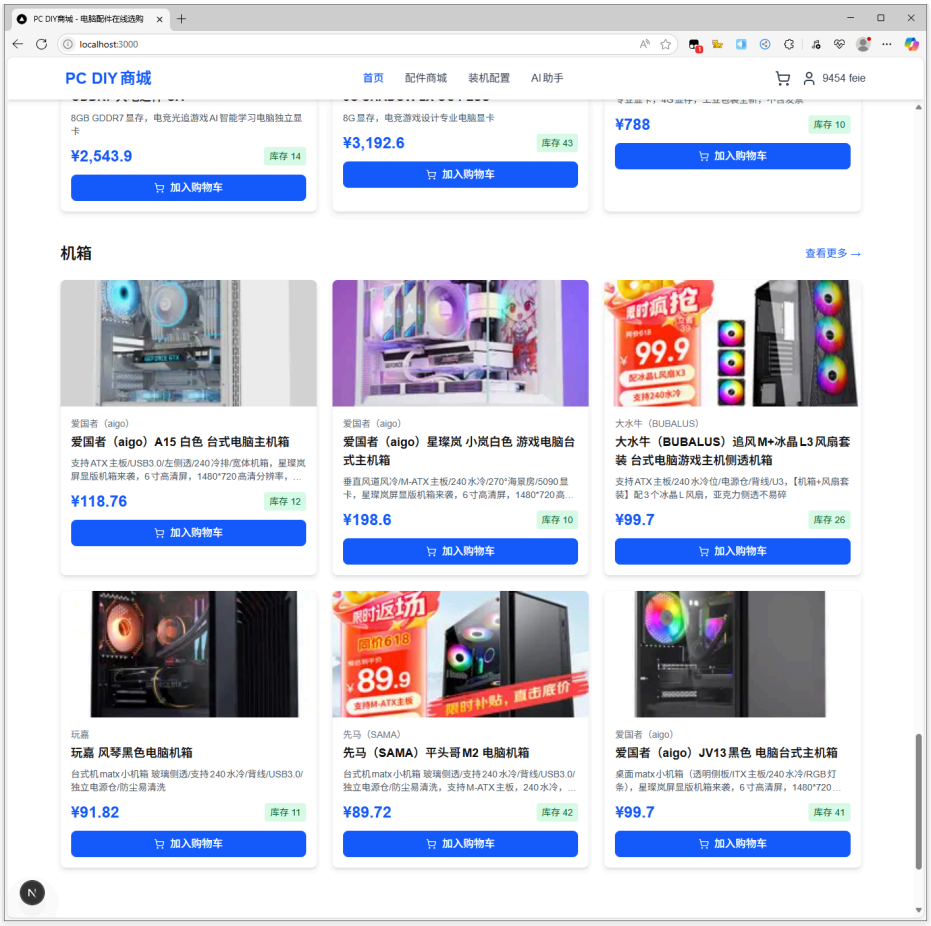

图4-14 系统首页 - 下

4.2.1.2 用户认证系统实现

用户注册功能：

```
// app/register/page.tsx - 注册页面核心逻辑
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault()

  if (formData.password !== formData.confirmPassword) {
    setError('两次密码输入不一致')
    return
  }

  try {
    const response = await fetch('/api/auth/register', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        email: formData.email,
        username: formData.username,
        password: formData.password,
        name: formData.name
      })
    })
```

```
      if (response.ok) {
        router.push('/login?message=注册成功，请登录')
      } else {
        const data = await response.json()
        setError(data.message || '注册失败')
      }
    } catch (error) {
      setError('网络错误，请稍后重试')
    }
  }
```

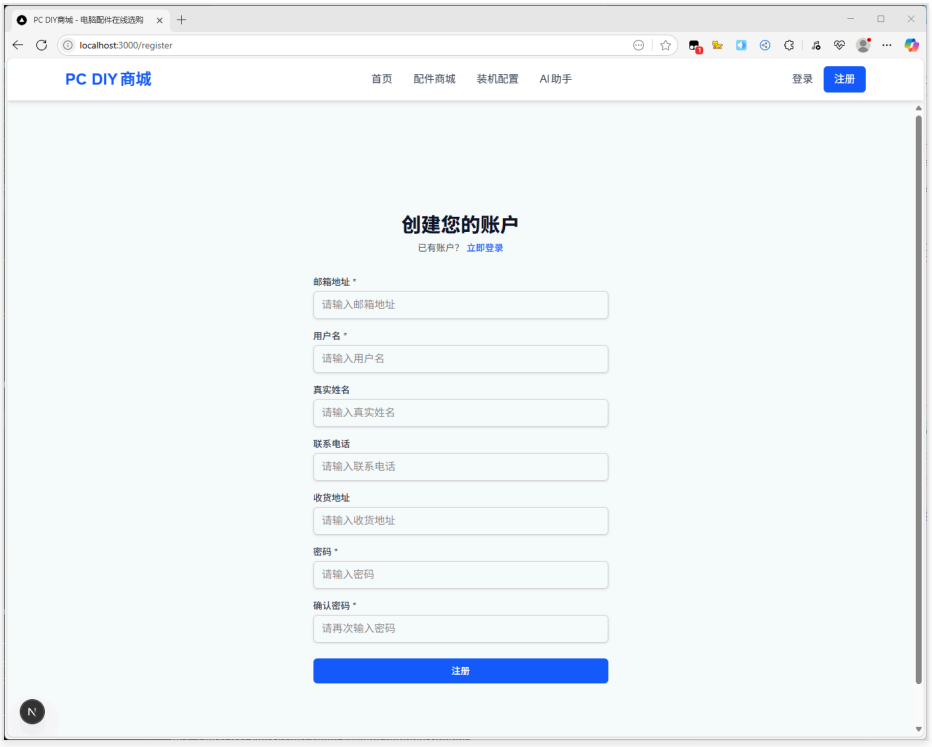

图4-15 用户注册

用户登录功能：

```
// app/api/auth/login/route.ts - 登录API实现
export async function POST(request: NextRequest) {
  try {
    const { identifier, password } = await request.json()

    // 支持邮箱或用户名登录
    const user = await prisma.user.findFirst({
      where: {
        OR: [
          { email: identifier },
          { username: identifier }
        ]
      }
```

```
    })

    if (!user) {
      return NextResponse.json(
        { message: '用户不存在' },
        { status: 401 }
      )
    }

    // 验证密码
    const isValidPassword = await verifyPassword(password,
user.password)
    if (!isValidPassword) {
      return NextResponse.json(
        { message: '密码错误' },
        { status: 401 }
      )
    }

    // 生成JWT Token
    const token = generateToken({
      userId: user.id,
      email: user.email,
      username: user.username,
      isAdmin: user.isAdmin
    })

    return NextResponse.json({
      message: '登录成功',
      token,
      user: {
        id: user.id,
        email: user.email,
        username: user.username,
        name: user.name,
        isAdmin: user.isAdmin
      }
    })
  } catch (error) {
    return NextResponse.json(
      { message: '登录失败' },
      { status: 500 }
    )
  }
}
```

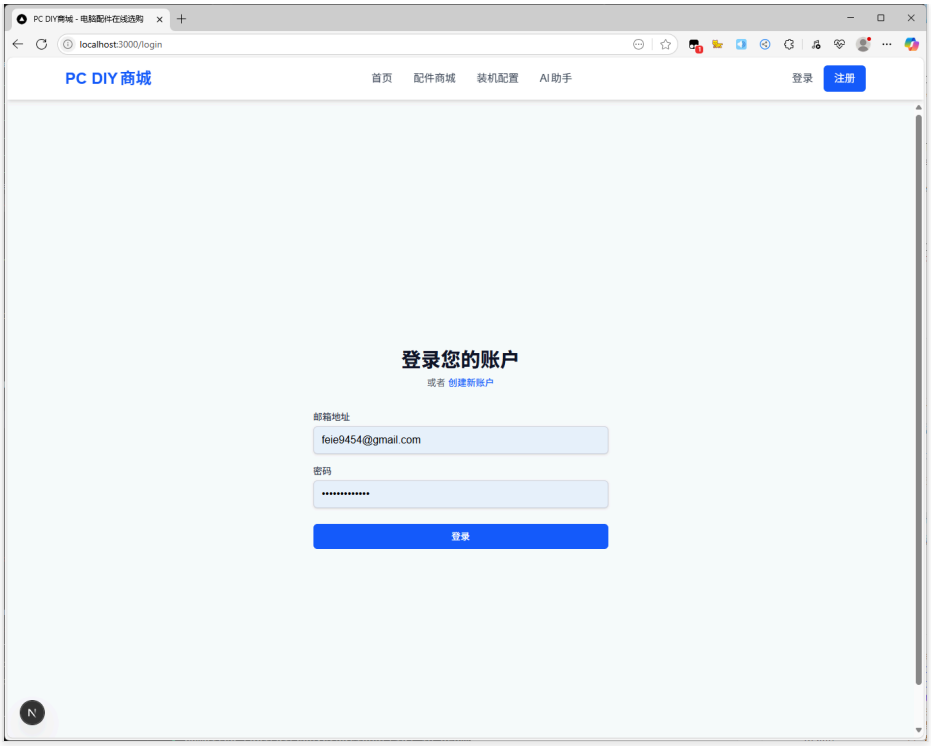

图4-16 用户登录

认证功能特点：

1. **双重登录支持**：支持邮箱或用户名登录
2. **密码安全**：使用bcrypt加密存储密码
3. **JWT认证**：无状态的Token认证机制
4. **表单验证**：前端实时验证和后端数据验证

4.2.1.3 配件浏览和搜索系统

配件列表页面实现：

```
// app/components/page.tsx - 配件浏览核心功能
export default function ComponentsPage() {
  const [components, setComponents] = useState<Component[]>([])
  const [filters, setFilters] = useState({
    search: '',      // 关键词搜索
    type: '',        // 配件类型
    brand: '',       // 品牌筛选
    minPrice: '',    // 最低价格
    maxPrice: ''     // 最高价格
  })

  const fetchComponents = async () => {
    const searchParams = new URLSearchParams()
    if (filters.search) searchParams.set('search', filters.search)
    if (filters.type) searchParams.set('type', filters.type)
```

```
    if (filters.brand) searchParams.set('brand', filters.brand)
    if (filters.minPrice) searchParams.set('minPrice',
filters.minPrice)
    if (filters.maxPrice) searchParams.set('maxPrice',
filters.maxPrice)
    searchParams.set('page', pagination.page.toString())
    searchParams.set('limit', pagination.limit.toString())

    const response = await fetch(`/api/components?${searchParams}`)
    if (response.ok) {
      const data = await response.json()
      setComponents(data.components)
      setPagination(data.pagination)
    }
  }

  return (
    <div className="min-h-screen bg-gray-50">
      {/* 搜索和筛选区域 */}
      <div className="bg-white p-6 rounded-lg shadow-sm mb-6">
        <div className="grid grid-cols-1 md:grid-cols-5 gap-4">
          {/* 关键词搜索 */}
          <div className="relative">
            <Search className="absolute left-3 top-1/2 transform -
translate-y-1/2 text-gray-400 h-4 w-4" />
            <input
              type="text"
              placeholder="搜索配件..."
              value={filters.search}
              onChange={(e) => setFilters({...filters, search:
e.target.value})}
              className="w-full pl-10 pr-4 py-2 border border-gray-300
rounded-md"
            />
          </div>

          {/* 配件类型筛选 */}
          <select
            value={filters.type}
            onChange={(e) => setFilters({...filters, type:
e.target.value})}
            className="w-full px-3 py-2 border border-gray-300
rounded-md"
          >
            <option value="">所有类型</option>
            {componentTypes.map(type => (
              <option key={type.id} value={type.id}>{type.name}
</option>
            ))}
          </select>

          {/* 价格区间筛选 */}
          <input
            type="number"
```

```
            placeholder="最低价格"
            value={filters.minPrice}
            onChange={(e) => setFilters({...filters, minPrice:
e.target.value})}
            className="w-full px-3 py-2 border border-gray-300
rounded-md"
          />
          <input
            type="number"
            placeholder="最高价格"
            value={filters.maxPrice}
            onChange={(e) => setFilters({...filters, maxPrice:
e.target.value})}
            className="w-full px-3 py-2 border border-gray-300
rounded-md"
          />
        </div>
      </div>

      {/* 配件展示网格 */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3
xl:grid-cols-4 gap-6">
        {components.map((component) => (
          <ComponentCard key={component.id} component={component} />
        ))}
      </div>

      {/* 分页控件 */}
      <Pagination pagination={pagination} onPageChange=
{handlePageChange} />
    </div>
  )
}
```

配件卡片组件实现：

```
// components/ComponentCard.tsx - 配件展示卡片
export function ComponentCard({ component }: { component: Component })
{
  return (
    <div className="bg-white rounded-lg shadow-sm hover:shadow-md
transition-shadow">
      <Link href={`/components/${component.id}`}>
        <div className="aspect-square bg-gray-100 rounded-t-lg
overflow-hidden">
          {component.imageUrl ? (
            <img
              src={component.imageUrl}
              alt={component.name}
              className="w-full h-full object-cover"
            />
```

```
          ) : (
            <div className="flex items-center justify-center h-full">
              <Package className="h-16 w-16 text-gray-400" />
            </div>
          )}
        </div>

        <div className="p-4">
          <h3 className="font-semibold text-gray-900 mb-1 line-clamp-
2">
            {component.name}
          </h3>
          <p className="text-sm text-gray-600 mb-2">
            {component.brand} | {component.model}
          </p>
          <div className="flex justify-between items-center">
            <span className="text-lg font-bold text-red-600">
              ¥{component.price.toFixed(2)}
            </span>
            <span className={`text-sm ${component.stock > 0 ? 'text-
green-600' : 'text-red-600'}`}>
              {component.stock > 0 ? `库存${component.stock}` : '缺
货'}
            </span>
          </div>
        </div>
      </Link>

      <div className="px-4 pb-4">
        <AddToCartButton
          componentId={component.id}
          disabled={component.stock === 0}
          className="w-full"
        />
      </div>
    </div>
  )
}
```

搜索功能特点：

1. **多维度筛选**：支持类型、品牌、价格区间、关键词搜索
2. **实时搜索**：用户输入时实时更新结果
3. **分页展示**：大数据量的分页处理
4. **响应式布局**：自适应网格布局系统

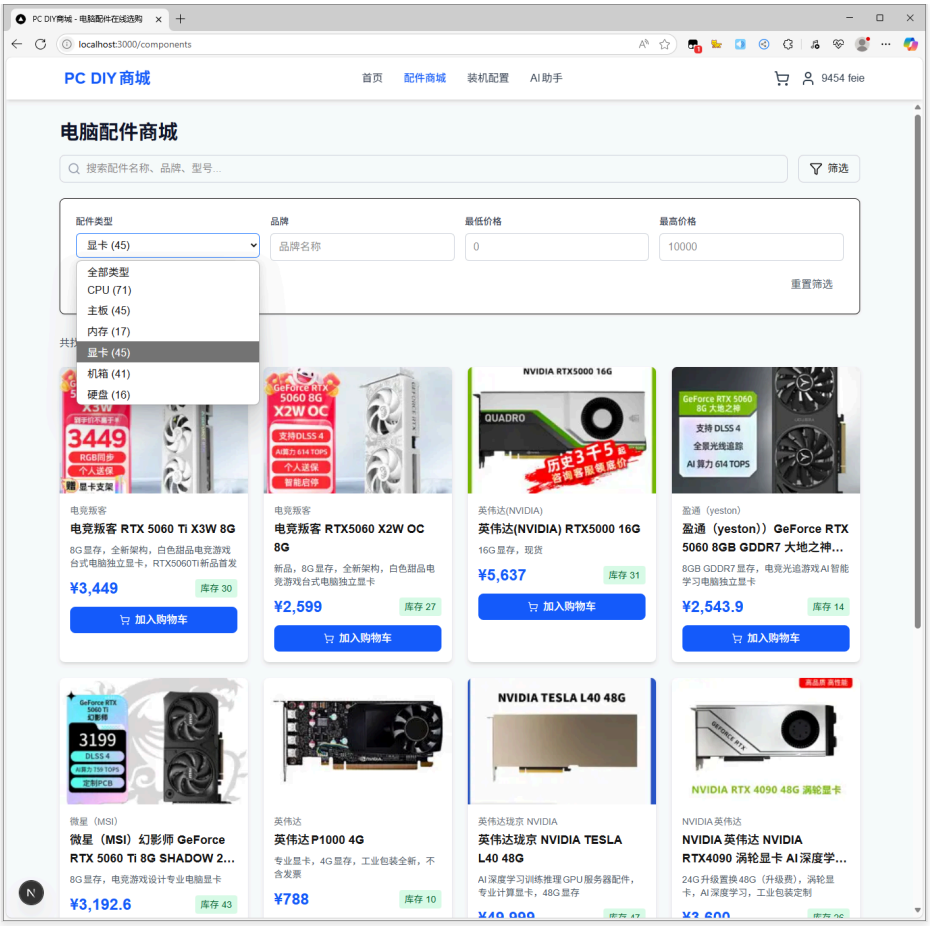

图4-17 配件商城 - 支持筛选与搜索

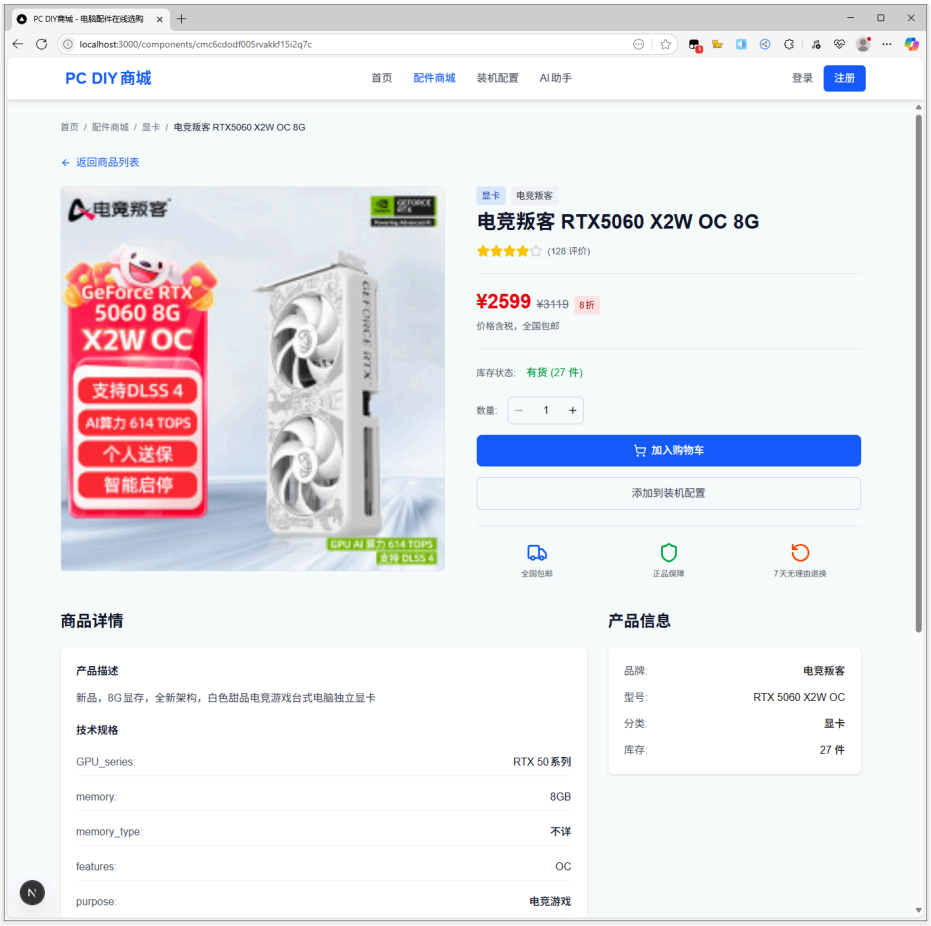

图4-18 配件详情页

4.2.1.4 购物车和订单系统

购物车功能实现：

```
// app/cart/page.tsx - 购物车页面
export default function CartPage() {
  const [cartItems, setCartItems] = useState<CartItem[]>([])

  const updateQuantity = async (itemId: string, newQuantity: number)
=> {
    const token = localStorage.getItem('token')

    if (newQuantity === 0) {
      // 删除商品
      await fetch(`/api/cart/${itemId}`, {
        method: 'DELETE',
        headers: { 'Authorization': `Bearer ${token}` }
      })
    } else {
      // 更新数量
      await fetch(`/api/cart/${itemId}`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
```

```
          'Authorization': `Bearer ${token}`
        },
        body: JSON.stringify({ quantity: newQuantity })
      })
    }

    loadCart() // 重新加载购物车
  }

  const checkout = async () => {
    if (cartItems.length === 0) {
      alert('购物车为空')
      return
    }

    try {
      const token = localStorage.getItem('token')
      const response = await fetch('/api/orders', {
        // ...
      })

      if (response.ok) {
        const order = await response.json()
        alert(`订单创建成功！订单号：${order.orderNumber}`)
        // 清空购物车
        await fetch('/api/cart', {
          method: 'DELETE',
          headers: { 'Authorization': `Bearer ${token}` }
        })
        router.push(`/orders/${order.id}`)
      }
    } catch (error) {
      alert('结算失败，请稍后重试')
    }
  }

  return (
    <div className="container mx-auto px-4 py-8">
      <h1 className="text-2xl font-bold mb-6">购物车</h1>

      {cartItems.length === 0 ? (
        <div className="text-center py-12">
          <ShoppingCart className="h-16 w-16 text-gray-400 mx-auto mb-
4" />
          <p className="text-gray-500">购物车是空的</p>
        </div>
      ) : (
        <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
          {/* 购物车商品列表 */}
          <div className="lg:col-span-2">
            {cartItems.map(item => (
              <CartItemCard
                key={item.id}
                item={item}
```

```
                onUpdateQuantity={updateQuantity}
              />
            ))}
          </div>

          {/* 订单摘要 */}
          <div className="bg-white p-6 rounded-lg shadow-sm h-fit">
            <h3 className="text-lg font-semibold mb-4">订单摘要</h3>
            <div className="space-y-2 mb-4">
              <div className="flex justify-between">
                <span>商品总价:</span>
                <span>¥{getTotalPrice().toFixed(2)}</span>
              </div>
              <div className="flex justify-between font-semibold text-
lg">
                <span>总计:</span>
                <span className="text-red-600">¥
{getTotalPrice().toFixed(2)}</span>
              </div>
            </div>
            <button
              onClick={checkout}
              className="w-full bg-blue-600 text-white py-3 px-4
rounded-lg hover:bg-blue-700"
            >
              立即结算
            </button>
          </div>
        </div>
      )}
    </div>
  )
}
```

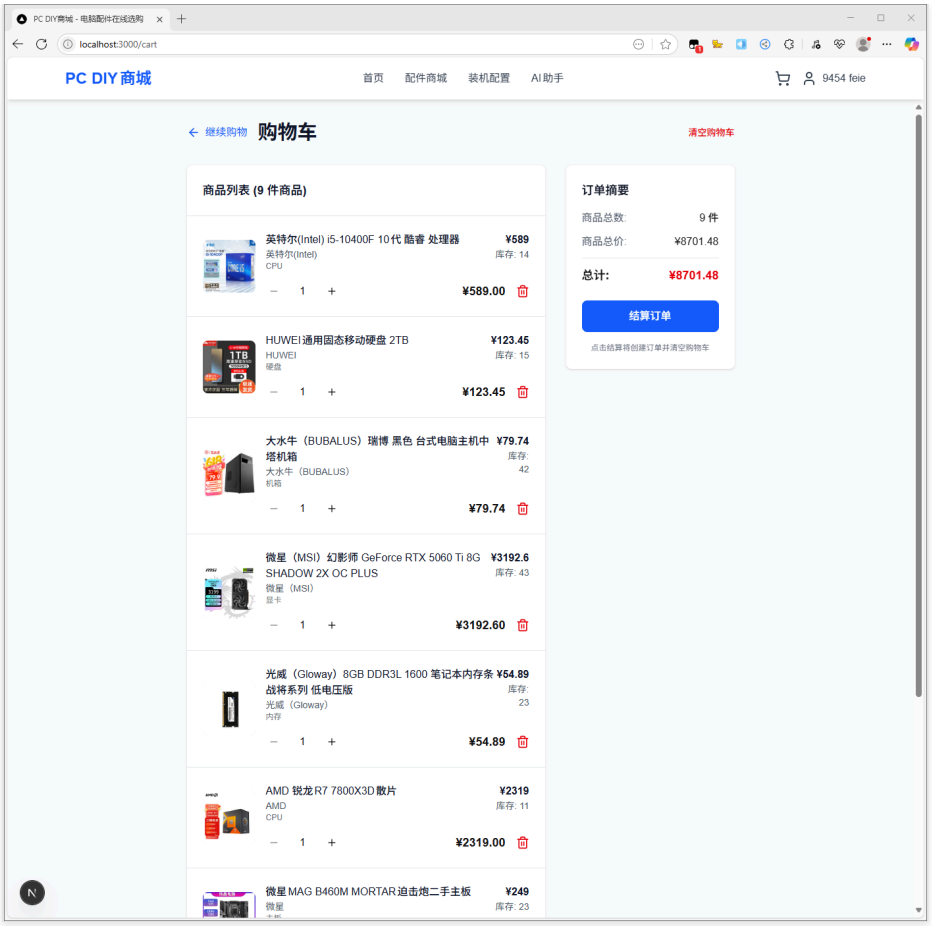

图4-19 购物车页面

订单管理实现：

```
// app/api/orders/route.ts - 订单创建API
export async function POST(request: NextRequest) {
  try {
    const { items } = await request.json()
    const authHeader = request.headers.get('authorization')
    const token = authHeader?.substring(7)
    const decoded = verifyToken(token!)

    if (!decoded) {
      return NextResponse.json({ message: '请先登录' }, { status: 401
})
    }

    // 生成订单号
    const orderNumber =
`ORD${Date.now()}${Math.random().toString(36).substr(2,
6).toUpperCase()}`

    let totalAmount = 0
    const orderItems: any[] = []
```

```
// 验证配件和计算总价
for (const item of items) {
  const component = await prisma.component.findUnique({
    where: { id: item.componentId }
  })

  if (!component) {
    return NextResponse.json(
      { message: `配件不存在: ${item.componentId}` },
      { status: 400 }
    )
  }

  if (component.stock < item.quantity) {
    return NextResponse.json(
      { message: `配件库存不足: ${component.name}` },
      { status: 400 }
    )
  }

  const itemTotal = component.price * item.quantity
  totalAmount += itemTotal

  orderItems.push({
    componentId: component.id,
    quantity: item.quantity,
    price: component.price
  })
}

// 创建订单
const order = await prisma.order.create({
  data: {
    orderNumber,
    totalAmount,
    userId: decoded.userId,
    orderItems: { create: orderItems }
  },
  include: {
    orderItems: {
      include: {
        component: {
          include: { componentType: true }
        }
      }
    }
  }
})

// 更新库存
for (const item of items) {
  await prisma.component.update({
    where: { id: item.componentId },
    data: { stock: { decrement: item.quantity } }
```

```
      })
    }

    return NextResponse.json(order, { status: 201 })
  } catch (error) {
    return NextResponse.json({ message: '创建订单失败' }, { status:
500 })
  }
}
```

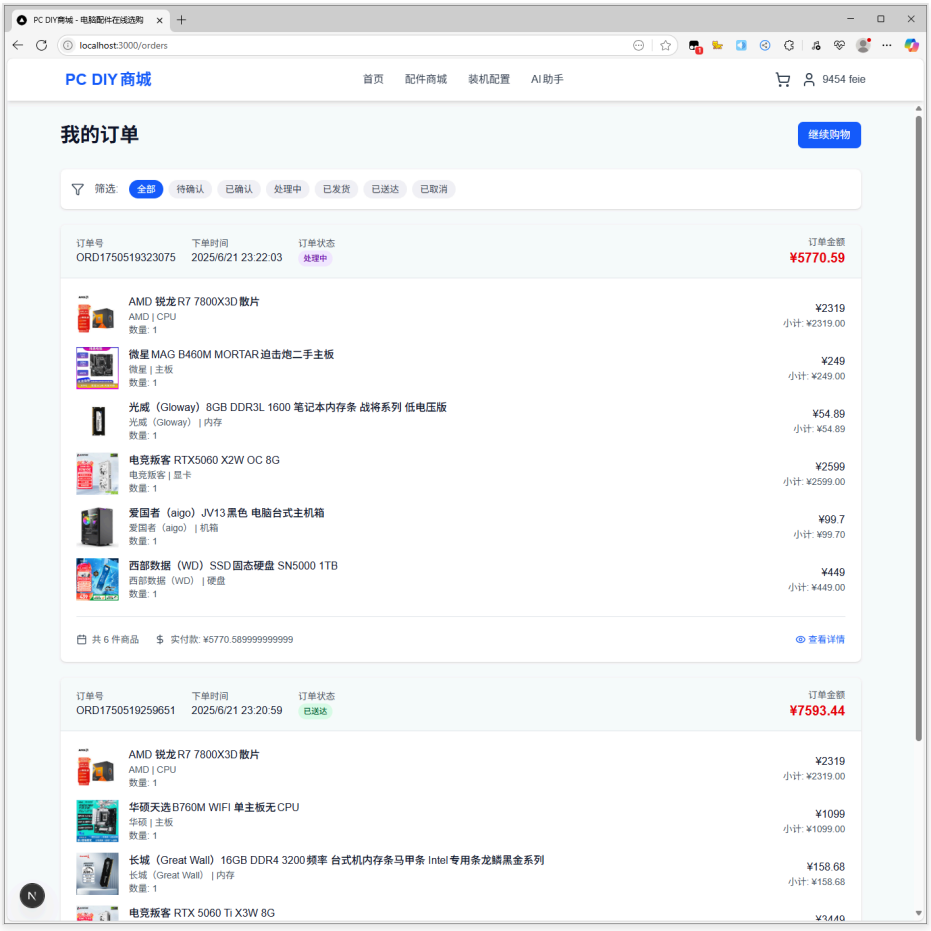

图4-20 订单管理页面

订单功能特点：

1. **购物车管理**：添加、删除、修改数量
2. **库存验证**：实时检查配件库存
3. **订单创建**：自动生成订单号，记录订单详情
4. **库存更新**：订单创建后自动减少库存

4.2.1.5 装机配置系统

装机配置页面实现：

```
// app/build/page.tsx - 装机配置核心功能
export default function BuildPage() {
  const [buildConfig, setBuildConfig] = useState<BuildConfiguration>
({})
  const [availableComponents, setAvailableComponents] = useState<{
[key: string]: Component[] }>({})

  // 一些辅助函数...

  return (
    <div className="min-h-screen bg-gray-50">
      <div className="max-w-7xl mx-auto px-4 py-8">
        <div className="text-center mb-12">
          <h1 className="text-4xl font-bold mb-4">PC装机配置</h1>
          <p className="text-xl text-gray-600 mb-8">选择每种配件，组装
您的专属电脑</p>

          {/* 配置进度 */}
          <div className="max-w-md mx-auto">
            <div className="flex justify-between text-sm text-gray-600
mb-2">
              <span>配置进度</span>
              <span>{getCompletionRate()}%</span>
            </div>
            <div className="w-full bg-gray-200 rounded-full h-2">
              <div
                className="bg-blue-600 h-2 rounded-full transition-all
duration-300"
                style={{ width: `${getCompletionRate()}%` }}
              />
            </div>
          </div>
        </div>

        <div className="grid grid-cols-1 lg:grid-cols-4 gap-8">
          {/* 配件选择区域 */}
          <div className="lg:col-span-3">
            <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
              {componentTypes.map((type) => {
                const selectedComponent = buildConfig[type.name]
                return (
                  <ComponentTypeCard
                    key={type.id}
                    type={type}
                    selectedComponent={selectedComponent}
                    onSelect={(component) =>
selectComponent(type.name, component)}
                    onRemove={() => removeComponent(type.name)}
```

```
                  />
                )
              })}
            </div>
          </div>

          {/* 配置摘要 */}
          <div className="bg-white p-6 rounded-lg shadow-sm h-fit">
            <h3 className="text-lg font-semibold mb-4">配置摘要</h3>
            <div className="space-y-3 mb-6">
              {componentTypes.map((type) => {
                const component = buildConfig[type.name]
                return (
                  <div key={type.id} className="flex justify-between
text-sm">
                    <span className="text-gray-600">{type.name}:
</span>
                    <span className={component ? "text-gray-900 font-
medium" : "text-gray-400"}>
                      {component ? `¥${component.price}` : '未选择'}
                    </span>
                  </div>
                )
              })}

              <div className="border-t pt-3">
                <div className="flex justify-between font-semibold">
                  <span>总价:</span>
                  <span className="text-red-600">¥
{getTotalPrice().toFixed(2)}</span>
                </div>
              </div>
            </div>

            <div className="space-y-3">
              <button
                onClick={addAllToCart}
                disabled=
{Object.values(buildConfig).filter(Boolean).length === 0}
                className="w-full bg-blue-600 text-white py-3 px-4
rounded-lg hover:bg-blue-700 disabled:opacity-50"
              >
                全部加入购物车
              </button>
            </div>
          </div>
        </div>
      </div>
    </div>
  )
}
```

装机配置特点：

1. **分类选择**：每种配件类型只能选择一个
2. **实时预览**：显示已选配件和总价
3. **进度跟踪**：可视化配置完成度
4. **批量操作**：一键添加所有配件到购物车

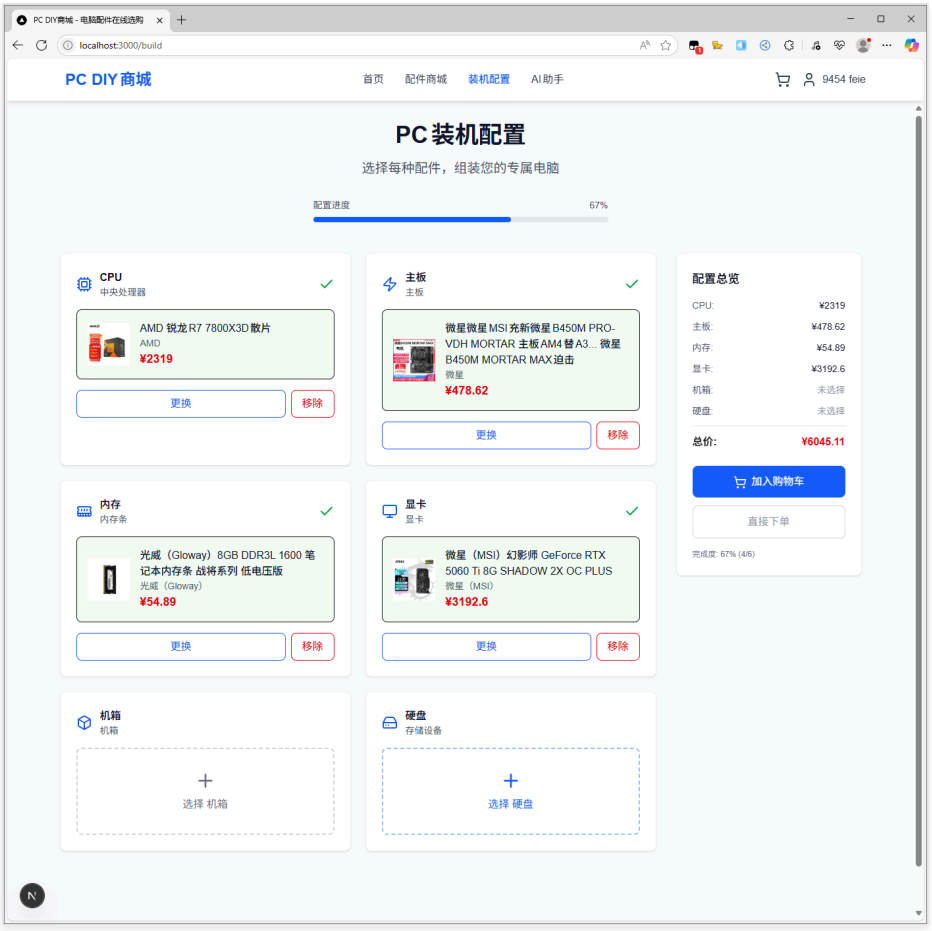

图4-21 整机装机页面

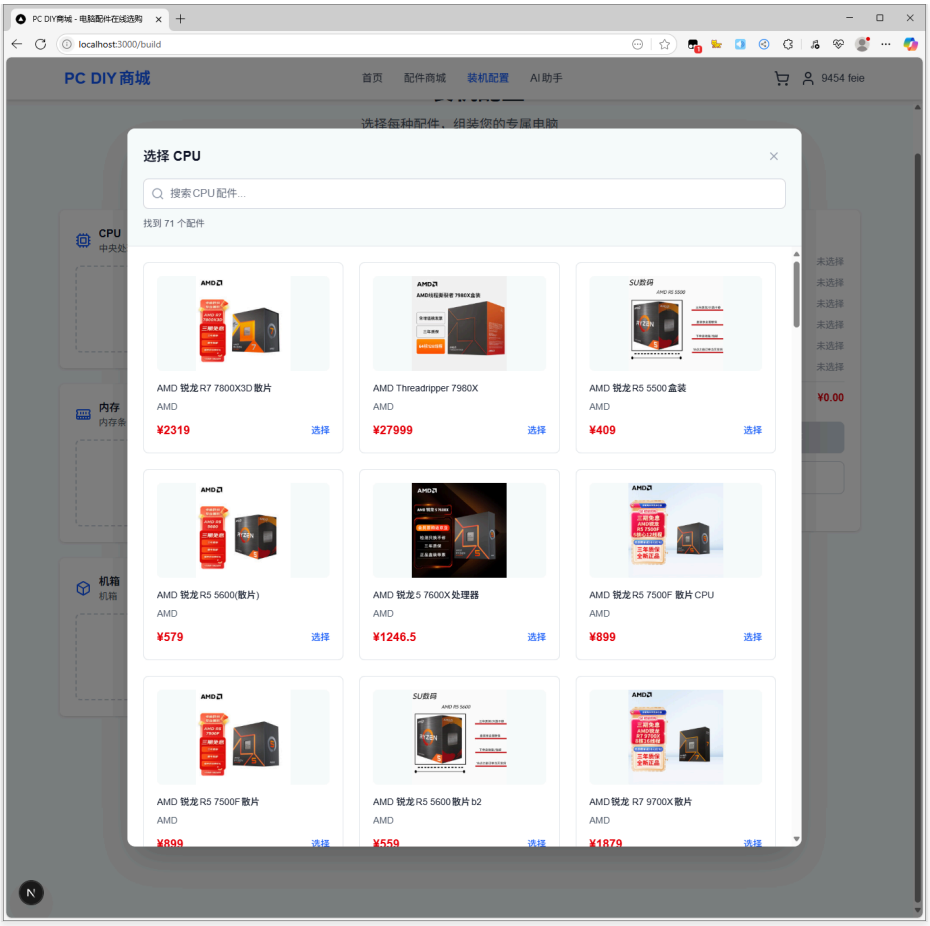
PC DIY商城 - 电脑配件在线选购
localhost:3000/build
PC DIY 商城
首页
配件商城
装机配置
AI 助手
9454 feie
选择每种配件，组装您的专属电脑
选择 CPU
搜索CPU配件...
找到 71 个配件
AMD 锐龙R7 7800X3D 散片
AMD
¥2319
选择
AMD Threadripper 7980X
AMD
¥27999
选择
AMD 锐龙R5 5500盒装
AMD
¥409
选择
AMD 锐龙R5 5600(散片)
AMD
¥579
选择
AMD 锐龙5 7600X处理器
AMD
¥1246.5
选择
AMD 锐龙R5 7500F 散片CPU
AMD
¥899
选择
AMD 锐龙R5 7500F 散片
AMD
¥899
选择
AMD 锐龙R5 5600散片b2
AMD
¥559
选择
AMD 锐龙 R7 9700X 散片
AMD
¥1879
选择
CPU
中央处
内存
内存条
机箱
机箱
未选择
¥0.00

图4-22 特定配件筛选

4.2.1.6 用户信息系统

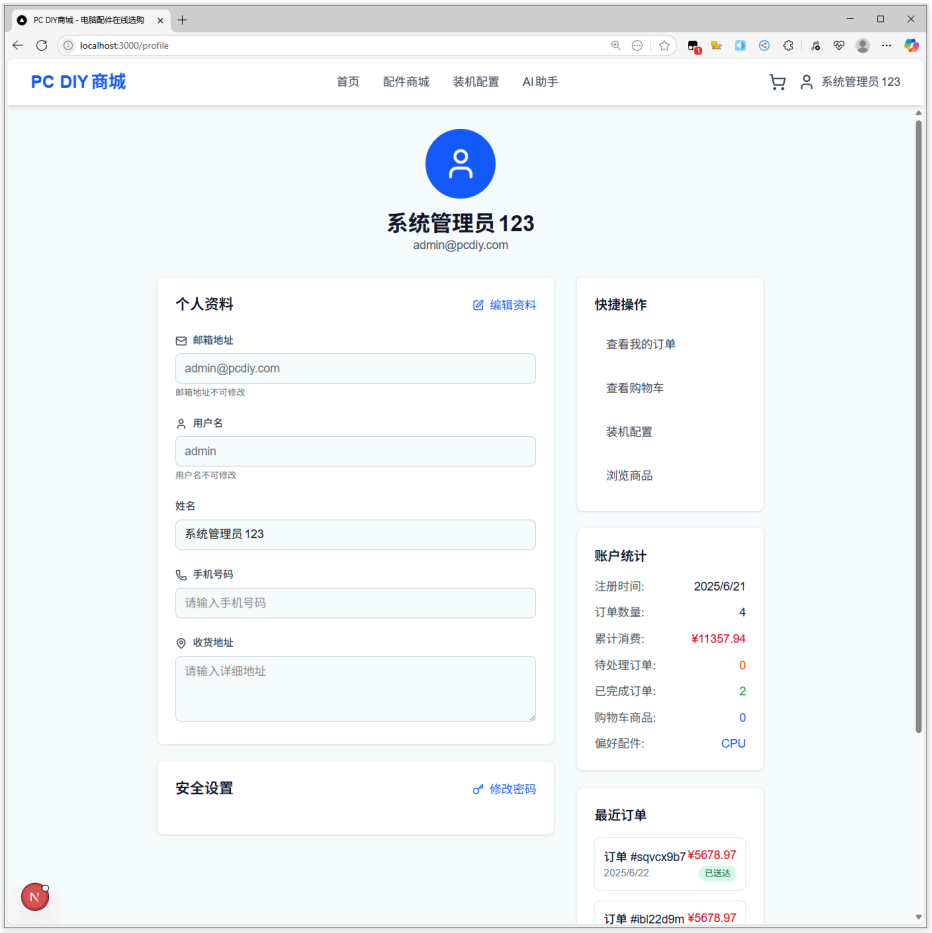

图4-23 个人信息查看与修改

4.2.2 AI智能助手功能实现（核心创新）

4.2.2.1 AI对话界面实现

对话界面核心组件：

```
// app/ai-assistant/page.tsx - AI助手页面
export default function AIAssistantPage() {
  const [messages, setMessages] = useState<Message[]>([])
  const [conversations, setConversations] = useState<Conversation[]>
([])
  const [currentConversationId, setCurrentConversationId] =
useState<string | null>(null)

  const sendMessage = async () => {
    if (!input.trim() || isLoading) return

    // 处理 input...

    try {
      // 发送请求...
```

```
      // 处理流式响应
      const reader = response.body?.getReader()
      const decoder = new TextDecoder()

      setMessages(prev => [...prev, {
        role: 'assistant',
        content: '',
        id: Date.now().toString()
      }])

      while (true) {
        const { done, value } = await reader.read()
        if (done) break

        const chunk = decoder.decode(value)
        const lines = chunk.split('\n')

        for (const line of lines) {
          if (line.startsWith('data: ')) {
            const data = line.slice(6)
            if (data === '[DONE]') {
              loadConversations()
              return
            }

            // 处理流式数据...
          }
        }
      }
    } catch (error) {
      console.error('发送消息失败:', error)
      setMessages(prev => [...prev, {
        role: 'assistant',
        content: '抱歉，发生了错误，请稍后再试。'
      }])
    } finally {
      setIsLoading(false)
    }
  }

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="flex gap-6 h-[calc(100vh-140px)]">
        {/* 对话历史侧边栏 */}
        <ConversationSidebar
          conversations={conversations}
          currentConversationId={currentConversationId}
          onSelectConversation={loadConversation}
          onNewConversation={startNewConversation}
          onDeleteConversation={deleteConversation}
        />

        {/* 主聊天区域 */}
```

```
        <div className="flex-1 bg-white rounded-lg shadow-sm border
flex flex-col">
          <ChatHeader />
          <MessageList messages={messages} isLoading={isLoading} />
          <ChatInput
            input={input}
            onInputChange={setInput}
            onSendMessage={sendMessage}
            isLoading={isLoading}
          />
        </div>
      </div>
    </div>
  )
}
```

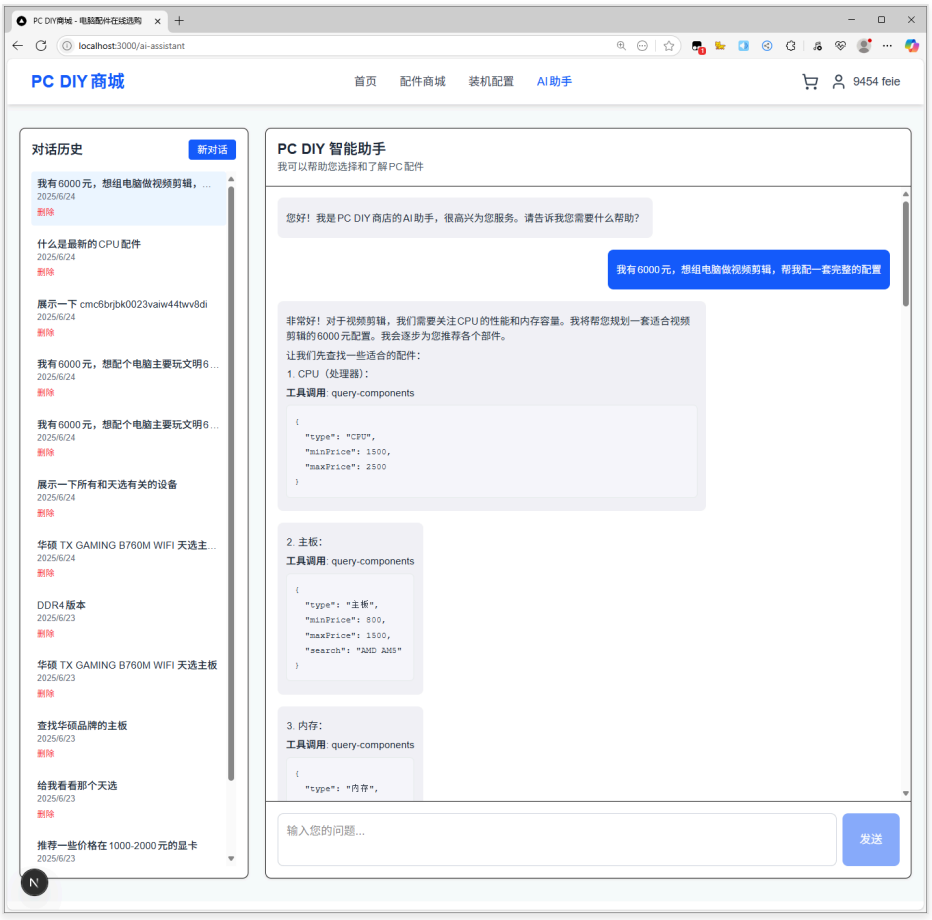

图4-24 AI 聊天页面 - 分析用户需求并搜索对应的配件

4.2.2.2 AI工具调用实现

配件查询工具实现：

```
// lib/ai-assistant.ts - AI工具调用核心逻辑

// 具体代码在 5.1.2 展示
```

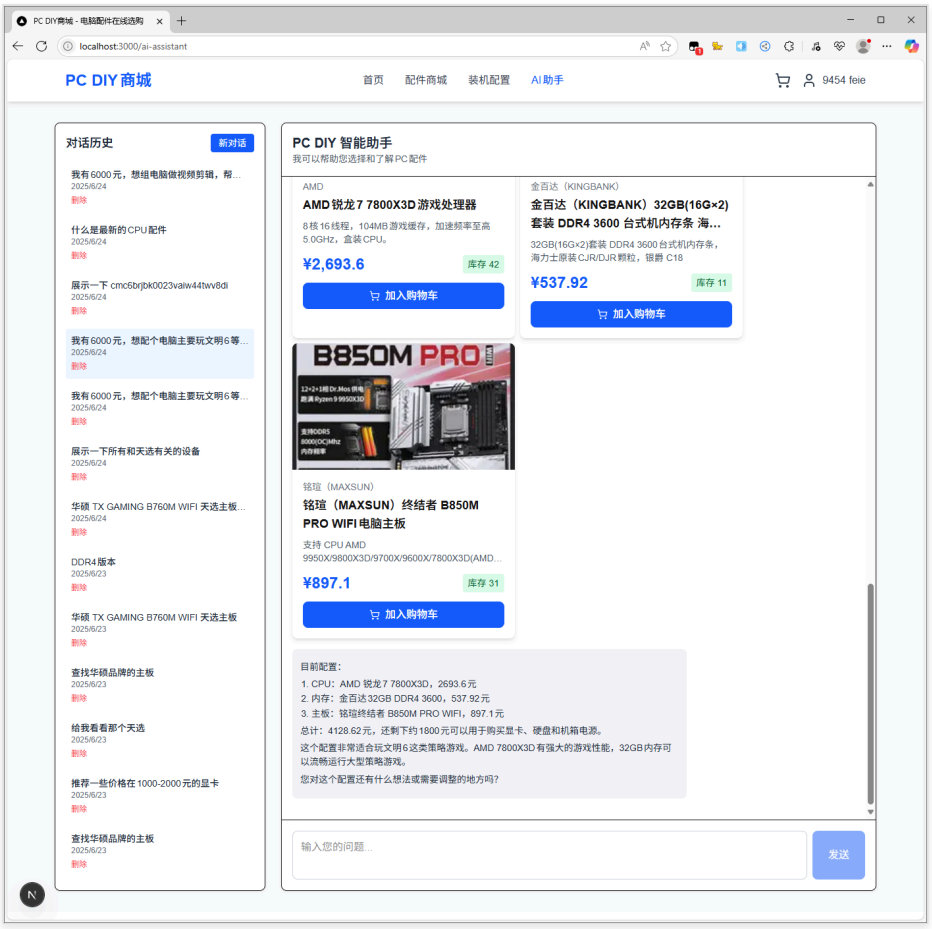

图4-25 AI 研究完成后以卡片形式向用户展示配件

AI助手功能特点：

1. **智能对话**：基于Claude AI的自然语言理解
2. **工具调用**：AI可以主动查询数据库获取实时信息
3. **流式响应**：实时打字机效果提升用户体验
4. **多轮对话**：保持上下文的连续对话
5. **配件展示**：以卡片形式展示查询到的配件
6. **对话管理**：完整的对话历史记录和管理

4.2.3 管理员功能实现

4.2.3.1 管理员认证和权限控制

管理员认证组件：

```
import { NextRequest } from 'next/server'
import { verifyToken } from '@/lib/auth'
import { prisma } from '@/lib/prisma'

export async function requireAdmin(request: NextRequest) {
  const authHeader = request.headers.get('Authorization')
  const token = authHeader?.replace('Bearer ', '')

  if (!token) {
    throw new Error('未提供认证令牌')
  }

  const decoded = verifyToken(token)
  if (!decoded) {
    throw new Error('无效的认证令牌')
  }

  const user = await prisma.user.findUnique({
    where: { id: decoded.userId },
    select: { id: true, isAdmin: true }
  })

  if (!user || !user.isAdmin) {
    throw new Error('权限不足')
  }

  return user
}
```

4.2.3.2 配件管理功能

配件管理页面实现：

```
// app/admin/components/page.tsx - 管理员配件管理
function AdminComponentsPage() {
  const [components, setComponents] = useState<Component[]>([])
  const [showModal, setShowModal] = useState(false)
  const [editingComponent, setEditingComponent] = useState<Component |
null>(null)

  // 批量导入功能
  const handleBatchImport = async () => {
    if (!batchFile) {
```

```
      alert('请选择文件')
      return
    }

    setBatchUploading(true)
    try {
      const formData = new FormData()
      formData.append('file', batchFile)

      const response = await fetch('/api/components/batch', {
        method: 'POST',
        headers: getAuthHeaders(),
        body: JSON.stringify({ components: batchData })
      })

      // ...
    } catch (error) {
      alert('批量导入失败')
    } finally {
      setBatchUploading(false)
    }
  }

  // 单个配件操作
  const handleSubmit = async (e: React.FormEvent) => {
     // ...
  }

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="flex justify-between items-center mb-6">
        <h1 className="text-2xl font-bold">配件管理</h1>
        <div className="flex space-x-2">
          {/* 导入按钮 */}
        </div>
      </div>

      {/* 配件列表 */}
      <div className="bg-white rounded-lg shadow-sm overflow-hidden">
        <table className="min-w-full divide-y divide-gray-200">
          <thead className="bg-gray-50">
            <tr>
              <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                配件信息
              </th>
              <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                价格/库存
              </th>
              <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                类型
              </th>
```

```
              <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                操作
              </th>
            </tr>
          </thead>
          <tbody className="bg-white divide-y divide-gray-200">
            {components.map((component) => (
              <ComponentTableRow
                key={component.id}
                component={component}
                onEdit={handleEdit}
                onDelete={handleDelete}
              />
            ))}
          </tbody>
        </table>
      </div>

      {/* 配件编辑模态框 */}
      <ComponentEditModal
        show={showModal}
        component={editingComponent}
        onClose={() => setShowModal(false)}
        onSubmit={handleSubmit}
      />

      {/* 批量导入预览 */}
      <BatchImportPreview
        data={batchData}
        onImport={handleBatchImport}
        isUploading={batchUploading}
      />
    </div>
  )
}

export default withAdminAuth(AdminComponentsPage)
```

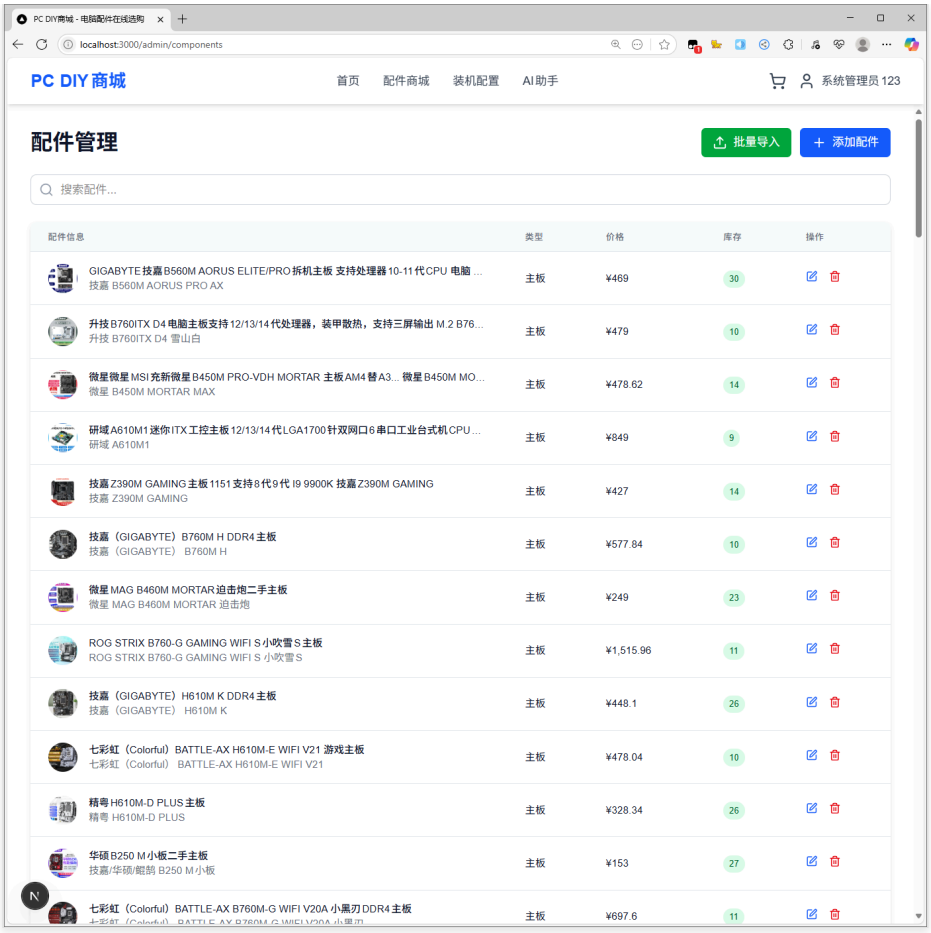
PC DIY商城 - 电脑配件在线选购
localhost:3000/admin/components
PC DIY 商城
首页
配件商城
装机配置
AI助手
系统管理员 123
配件管理
批量导入
添加配件
搜索配件...
配件信息
类型
价格
库存
操作
GIGABYTE 技嘉B560M AORUS ELITE/PRO 拆机主板 支持处理器 10-11 代CPU 电脑 ...
技嘉 B560M AORUS PRO AX
主板
¥469
30
升技 B760ITX D4 电脑主板支持 12/13/14 代处理器，装甲散热，支持三屏输出 M.2 B76...
升技 B760ITX D4 雪山白
主板
¥479
10
微星微星 MSI 充新微星 B450M PRO-VDH MORTAR 主板 AM4 替 A3... 微星 B450M MO...
微星 B450M MORTAR MAX
主板
¥478.62
14
研域 A610M1 迷你 ITX 工控主板 12/13/14 代 LGA1700 针双网口 6 串口工业台式机 CPU ...
研域 A610M1
主板
¥849
9
技嘉 Z390M GAMING 主板 1151 支持 8 代 9 代 I9 9900K 技嘉 Z390M GAMING
技嘉 Z390M GAMING
主板
¥427
14
技嘉（GIGABYTE）B760M H DDR4 主板
技嘉（GIGABYTE） B760M H
主板
¥577.84
10
微星 MAG B460M MORTAR 迫击炮二手主板
微星 MAG B460M MORTAR 迫击炮
主板
¥249
23
ROG STRIX B760-G GAMING WIFI S 小吹雪 S 主板
ROG STRIX B760-G GAMING WIFI S 小吹雪 S
主板
¥1,515.96
11
技嘉（GIGABYTE）H610M K DDR4 主板
技嘉（GIGABYTE） H610M K
主板
¥448.1
26
七彩虹（Colorful）BATTLE-AX H610M-E WIFI V21 游戏主板
七彩虹（Colorful） BATTLE-AX H610M-E WIFI V21
主板
¥478.04
10
精粤 H610M-D PLUS 主板
精粤 H610M-D PLUS
主板
¥328.34
26
华硕 B250 M 小板二手主板
技嘉/华硕/鳄鹅 B250 M 小板
主板
¥153
27
七彩虹（Colorful）BATTLE-AX B760M-G WIFI V20A 小黑刃 DDR4 主板
主板
¥697.6
11

图4-26 配件管理列表

图4-27 批量导入配件（结合爬虫）

4.2.3.3 统计功能实现

管理员统计页面：

```
// app/api/admin/stats/route.ts - 统计数据API
export async function GET(request: NextRequest) {
  try {
    await requireAdmin(request)

    // 按消费金额排序的前十用户
    const topUsers = await prisma.user.groupBy({
      by: ['id'],
      _sum: {
        orders: {
          totalAmount: true
        }
      },
      orderBy: {
        _sum: {
          orders: {
            totalAmount: 'desc'
          }
        }
      },
```

```
      take: 10
    })

    // 按销售量排序的前十配件
    const topComponents = await prisma.orderItem.groupBy({
      by: ['componentId'],
      _sum: {
        quantity: true
      },
      orderBy: {
        _sum: {
          quantity: 'desc'
        }
      },
      take: 10
    })

    // 获取详细信息
    const usersWithSpending = await Promise.all(
      topUsers.map(async (user) => {
        const userInfo = await prisma.user.findUnique({
          where: { id: user.id },
          select: { username: true, name: true, email: true }
        })
        return {
          ...userInfo,
          totalSpending: user._sum.orders?.totalAmount || 0
        }
      })
    )

    const componentsWithSales = await Promise.all(
      topComponents.map(async (item) => {
        const component = await prisma.component.findUnique({
          where: { id: item.componentId },
          include: { componentType: true }
        })
        return {
          ...component,
          totalSales: item._sum.quantity || 0
        }
      })
    )

    // 总体统计
    const totalUsers = await prisma.user.count({ where: { isAdmin:
false } })
    const totalOrders = await prisma.order.count()
    const totalComponents = await prisma.component.count()
    const totalRevenue = await prisma.order.aggregate({
      _sum: { totalAmount: true }
    })

    return NextResponse.json({
```

```
      overview: {
        totalUsers,
        totalOrders,
        totalComponents,
        totalRevenue: totalRevenue._sum.totalAmount || 0
      },
      topUsers: usersWithSpending,
      topComponents: componentsWithSales,
      recentOrders: await getRecentOrders()
    })

  } catch (error: any) {
    return NextResponse.json(
      { message: error.message || '获取统计数据失败' },
      { status: error.message === '权限不足' ? 403 : 500 }
    )
  }
}
```

数据可视化实现：

```
// 使用Chart.js和react-chartjs-2实现图表展示
import { Bar, Doughnut } from 'react-chartjs-2'

function AdminStatsPage() {
  const [stats, setStats] = useState<AdminStats | null>(null)

  // 用户消费排行榜配置
  const userChartData = {
    labels: stats?.topUsers.map(u => u.username || u.name) || [],
    datasets: [{
      label: '消费金额',
      data: stats?.topUsers.map(u => u.totalSpending) || [],
      backgroundColor: 'rgba(59, 130, 246, 0.5)',
      borderColor: 'rgba(59, 130, 246, 1)',
      borderWidth: 1
    }]
  }

  // 配件销量排行榜配置
  const componentChartData = {
    labels: stats?.topComponents.map(c => c.name) || [],
    datasets: [{
      label: '销售数量',
      data: stats?.topComponents.map(c => c.totalSales) || [],
      backgroundColor: 'rgba(34, 197, 94, 0.5)',
      borderColor: 'rgba(34, 197, 94, 1)',
      borderWidth: 1
    }]
  }
```

```
  return (
    <div className="container mx-auto px-4 py-8">
      <h1 className="text-2xl font-bold mb-6">统计分析</h1>

      <div className="grid grid-cols-1 md:grid-cols-4 gap-6 mb-8">
        {/* 总体数据卡片 */}
      </div>

      {/* 图表区域 */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {/* 用户消费排行榜 */}
        <div className="bg-white p-6 rounded-lg shadow-sm">
          <h3 className="text-lg font-semibold mb-4">用户消费排行榜（前
10名）</h3>
          <Bar
            data={userChartData}
            options={{
              responsive: true,
              plugins: {
                legend: { position: 'top' },
                title: { display: false }
              }
            }}
          />
        </div>

        {/* 配件销量排行榜 */}
        <div className="bg-white p-6 rounded-lg shadow-sm">
          <h3 className="text-lg font-semibold mb-4">配件销量排行榜（前
10名）</h3>
          <Bar
            data={componentChartData}
            options={{
              responsive: true,
              plugins: {
                legend: { position: 'top' },
                title: { display: false }
              }
            }}
          />
        </div>
      </div>
    </div>
  )
}
```

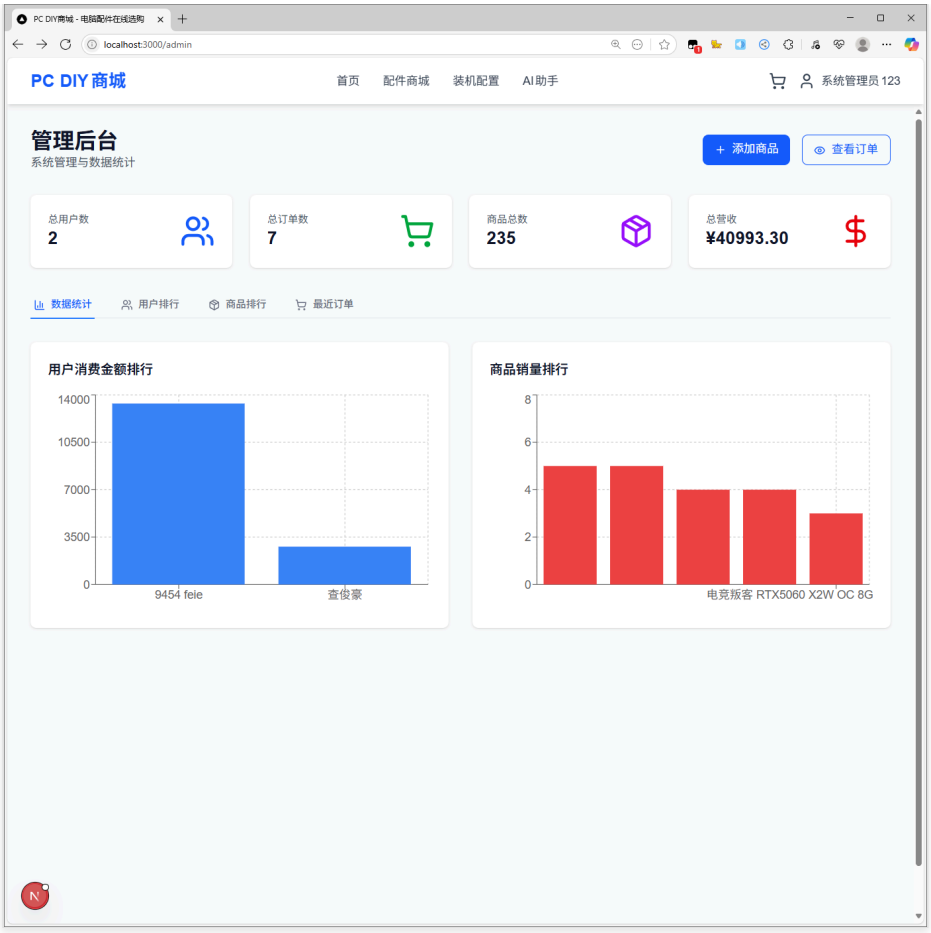

图4-28 统计页面

管理员功能特点：

1. **权限控制**：严格的管理员身份验证
2. **配件管理**：完整的CRUD操作
3. **批量操作**：支持CSV/JSON批量导入
4. **统计分析**：用户消费和配件销量统计
5. **数据可视化**：图表展示统计结果
6. **订单管理**：查看和管理所有订单

通过以上详细的功能实现展示，可以看出本系统完全满足了课程设计的所有要求，并在此基础上实现了诸多创新功能，特别是AI智能助手的集成，大大提升了用户体验和系统的实用性。系统采用现代化的技术栈，代码结构清晰，功能模块化，具有良好的可维护性和扩展性。

第五章 总结

5.1 项目开发过程中遇到的困难与解决方法

5.1.1 技术选型和架构设计挑战

遇到的困难：

在项目初期，面临着技术栈选择的难题。传统的Java Web开发虽然成熟稳定，但在现代化用户界面和开发效率方面存在不足。同时，如何在满足课程设计基本要求的同时，实现更具创新性的功能，是一个重要的技术挑战。

解决方案：

经过深入调研和比较，最终选择了Next.js全栈框架作为核心技术栈：

1. **框架优势分析**：
    1. Next.js 15.3.4提供了完整的全栈开发能力
    2. 支持服务端渲染(SSR)和静态站点生成(SSG)，提升性能和SEO
    3. App Router架构提供了更清晰的项目结构
    4. 内置的API Routes功能简化了后端开发
2. **架构设计决策**：
    1. 采用三层架构：表现层、业务逻辑层、数据访问层
    2. 服务层模式：将业务逻辑封装到独立的服务类中
    3. 微服务理念：每个功能模块相对独立，便于维护和扩展

经验总结：

技术选型应该综合考虑项目需求、开发效率、团队技能和未来维护等多个因素。选择现代化的技术栈虽然有学习成本，但能够带来更好的开发体验和最终产品质量。

5.1.2 AI助手功能集成的技术挑战

本项目中的AI助手旨在为用户提供智能化的PC配件查询和推荐服务。其核心是基于大型语言模型（Anthropic Claude系列），并通过工具调用（Tool Use）与网站后端的配件数据库进行交互。在集成过程中，我们遇到了多个技术挑战，尤其是在如何将模型的流式响应、工具调用、多轮对话和状态管理等特性无缝融合，以提供流畅且功能强大的用户体验。

遇到的困难

AI助手功能的集成是本项目最具挑战性的部分，主要困难包括：

1. **实时流式响应与工具调用的融合**：
   流式响应（Streaming）要求服务器持续向客户端推送数据，以实现打字机效果，提升用户体验。而工具调用本质上是一个“请求-等待-响应”的阻塞过程。如何在一个持续的流中，优雅地暂停文本输出、执行工具、再将工具结果返回给模型，并继续流式输出后续内容，是最大的技术难点。
2. **复杂的多轮工具调用与状态管理**：
   用户的单个问题可能需要AI进行多次工具调用才能完全解答。例如，“帮我找一款500元左右的AMD CPU，并展示出来”。这需要AI先调用 `query-components` 查找，然后根据结果再调用 `show-components` 展示。这要求后端能够维护一个复杂的、动态增长的对话上下文（`messages` 数组），确保每一次与AI的交互都包含完整的历史记录，包括之前的AI思考、工具调用和工具执行结果。
3. **前后端通信协议设计**：
   仅靠简单的文本流无法满足功能需求。前端需要明确知道当前流式数据是普通文本、AI正在思考使用哪个工具、还是一个需要渲染成特殊UI（如配件卡片）的指令。这需要设计一套清晰的、超越纯文本的通信协议。
4. **健壮的对话历史管理与错误恢复**：
   对话是持续的，用户随时可能返回之前的对话继续提问。系统必须能高效地加载、更新和持久化对话历史。同时，在复杂的AI与工具交互过程中，任何一步都可能出错（如API网络问题、工具执行失败）。系统需要具备容错能力，在出错时能给出清晰的反馈，并尽可能保存当前对话状态，避免用户丢失全部上下文。

解决方案（结合具体代码分析）

为了克服上述挑战，我们设计并实现了 `AIClient` 类，其核心逻辑集中在 `processQuery` 异步生成器函数中。

1. 流式响应与工具调用事件的统一处理

挑战：融合流式输出和阻塞的工具调用。

解决方案：我们充分利用了 Anthropic API 的流式响应特性，它会将一次完整的AI响应拆分成多个事件（`chunk`）。我们在 `for await (const chunk of response)` 循环中对这些事件进行精细化处理，实现了在流式输出的同时，解析并准备工具调用。

代码分析：

```
// processQuery 方法内
for await (const chunk of response) {
    console.log(chunk);
```

```
        switch (chunk.type) {
            // ... 其他 case ...
            case "content_block_start":
                const contentBlock = chunk.content_block;
                if (contentBlock.type === "tool_use") {
                    // ...
                    yield `\n\n**使用工具**:
  ${contentBlock.name}\n\n\`\`\`\n`; // (1)
                }
                break;

            case "content_block_delta":
                const contentBlockDelta = chunk.delta;
                if (contentBlockDelta.type === "input_json_delta") {
                    // ...
                    yield contentBlockDelta.partial_json || ""; // (2)
                }
                break;

            // ... 其他 case ...
        }
    }
```

- **即时反馈**：当AI决定使用工具时，API会发送一个 `content_block_start` 且类型为 `tool_use` 的事件。我们捕捉到这个事件后，立刻 `yield` 一段表示“正在使用工具”的Markdown文本 **(1)**。这让用户能实时看到AI的“思考过程”，而不是面对长时间的无响应等待。
- **流式展示工具参数**：紧接着，API会通过一系列 `content_body_delta` 且类型为 `input_json_delta` 的事件，流式地发来工具调用的参数JSON。我们将其直接 `yield` 给前端 **(2)**，前端可以实时展示AI正在构建的查询参数，进一步增强了透明度和用户体验。
- **构建完整响应**：在处理流的同时，我们使用 `tempMsg` 对象，根据接收到的 `chunk` 逐步拼装出AI的完整响应（包括文本和工具调用请求）。这为后续的工具执行和状态更新做好了准备。

通过这种方式，我们将“思考-调用-执行”的非流式过程，巧妙地包装在了对用户的流式反馈之中。

2. 迭代式多轮工具执行循环

挑战：处理需要多次工具调用的复杂查询，并管理好上下文。

解决方案：我们设计了一个 `while (currentIteration < maxIterations)` 循环，将用户的单次查询过程变成了一个“AI思考 -> 工具执行 -> AI再思考”的迭代循环。

代码分析：

```
// processQuery 方法内
let maxIterations = 20;
let currentIteration = 0;

while (currentIteration < maxIterations) { // (1) 核心循环
    currentIteration++;

    // ... 调用 aic.messages.create 并处理流式响应 ...

    // 将AI的完整响应（可能是文本或工具调用）加入到 messages 历史中
    messages.push({ role: tempMsg.role, content: ... });

    // (2) 检查AI响应中是否包含工具调用
    if (!tempMsg.content.find(c => c.type === "tool_use")) {
        break; // 如果没有工具调用，说明AI已完成回答，退出循环
    }

    // (3) 准备工具执行结果的消息体
    let tempUserMsg: MessageParam = { role: "user", content: [] };
    let needRerun = false;

    // (4) 遍历并执行所有工具调用
    for (const toolUse of tempMsg.content.filter(c => c.type ===
"tool_use")) {
        // ... 执行 queryComponents 或 show-components ...
        const result = await this.queryComponents(toolArgs);

        // (5) 将工具执行结果格式化为 tool_result 消息
        tempUserMsg.content.push({
            type: "tool_result",
            tool_use_id: toolUse.id,
            content: result
        });
        needRerun = true;
    }

    // (6) 将工具结果消息加入历史，准备下一次循环
    messages.push(tempUserMsg);
    if (!needRerun) break;
}
```

1. **核心循环 (1)**：`while` 循环是实现多轮工具调用的引擎。只要AI还在请求使用工具，循环就会继续。
2. **中断条件 (2)**：当AI的回复中不再包含 `tool_use` 类型的 `content_block` 时，意味着它认为任务已完成，可以生成最终答案了。此时 `break` 语句会终止循环。
3. **工具执行 (4)**：循环内部会遍历AI请求的所有工具调用，并执行对应的本地方法（如 `this.queryComponents`）。

4. **上下文注入 (5, 6)**：最关键的一步。工具的执行结果不会直接展示给用户，而是被包装成一个 `role: "user"` 且 `type: "tool_result"` 的消息，并添加到 `messages` 数组中。在下一次循环开始时，这个包含工具结果的 `messages` 数组会被再次发送给AI。这相当于告诉AI：“你上次想调用这个工具，这是它的执行结果，请基于这个结果继续你的思考和回答。”

这个迭代循环完美地模拟了AI的思维链，使其能够处理复杂的、需要分步解决的问题。

3. 自定义流式通信协议

挑战：前端需要结构化信息来渲染不同UI。

解决方案：我们通过 `yield` 语句，在文本流中嵌入了带特定前缀的“指令”，形成了一套简单有效的通信协议。

代码分析：

```
// 创建新对话时
yield `conversation_id:${currentConversationId}`;

// 调用 show-components 工具后
yield `show_card: ${JSON.stringify(components)}`;

// 在一个工具执行完毕，准备让AI生成最终回复前
yield 'next_block';
```

- `conversation_id:`：在对话创建之初，立即将新ID发送给前端，以便前端在后续请求中携带此ID。
- `show_card:`：这是一个明确的UI渲染指令。当AI调用 `show-components` 工具后，后端获取到配件的详细信息，并将其序列化为JSON字符串，附在 `show_card:` 之后 `yield` 出去。前端接收到这个格式的字符串后，就知道应该解析其后的JSON数据，并渲染成配件卡片UI，而不是当成普通文本显示。

图5-29 show_card 指令展示

- `next_block`：这是一个流程控制信号。它告诉前端，一个工具执行和结果返回的阶段已经完成，接下来AI将要生成总结性的文本。前端利用这个信号来创建新的消息气泡。

图5-30 next_block 指令展示

这套协议将后端的业务逻辑（如展示卡片）与前端的UI实现解耦，使得通信清晰且可扩展。

4. 原子化的对话状态管理与持久化

挑战：保证对话历史的完整性和面对错误时的健壮性。

解决方案：我们将对话管理抽象为 `ConversationService`，并在 `processQuery` 的开始和结束（包括异常情况）时进行调用，确保了对话状态的原子性更新。

代码分析：

```
// processQuery 方法开头
if (conversationId) {
    // 加载现有对话
    const conversation = await
ConversationService.getConversation(conversationId, userId);
    messages = [...conversation.messages];
    messages.push({ role: "user", content: query });
} else {
    // 创建新对话
    currentConversationId = await
ConversationService.createConversation(userId, query);
    // ...
}

// processQuery 方法 try-catch-finally 结构
try {
    // ... 核心的 while 循环逻辑 ...

    // (1) 成功后保存
    if (currentConversationId) {
        await
ConversationService.updateConversationMessages(currentConversationId,
userId, messages);
    }
} catch (error) {
    // ... 错误处理 ...
    // (2) 失败时也尝试保存
    if (currentConversationId) {
        try {
            await
ConversationService.updateConversationMessages(currentConversationId,
userId, messages);
        } catch (saveError) {
            console.error('保存对话失败:', saveError);
        }
    }
}
```

- **加载与启动**：在处理查询前，代码首先会根据 `conversationId` 决定是加载旧对话还是创建新对话，为本次交互准备好完整的上下文 `messages`。
- **事务性保存**：整个复杂的多轮交互（包含多次AI调用和工具执行）被视为一个事务。只有当 `while` 循环完全结束（即AI生成了最终答案）后，我们才调用 `updateConversationMessages` **(1)**，将包含用户提问、所有AI思考步骤、工具调用、工具结果和最终答案的完整 `messages` 数组一次性存入数据库。
- **错误恢复**：通过 `try...catch` 结构，即使在处理过程中发生错误，我们依然会尝试保存截至出错前的所有 `messages` **(2)**。这极大地提升了系统的健壮性，用户不会因为一次偶然的失败而丢失整个对话历史，同时也为问题排查提供了宝贵的上下文记录。

5.2 结语

通过本次课程设计的完整实施，不仅成功构建了一个功能完善、技术先进的在线电脑DIY系统，更重要的是在这个过程中获得了全面的技术能力提升和深入的工程实践体验。

项目成果的多重价值：

1. **技术价值：**展示了现代Web开发技术栈的综合应用，特别是AI技术与传统电商系统的深度融合，为同类项目提供了宝贵的技术参考。
2. **教育价值：**作为一个完整的学习项目，它涵盖了从需求分析到系统部署的全过程，为软件工程教育提供了优秀的实践案例。
3. **商业价值：**系统具备实际的商业应用潜力，AI助手功能的创新应用为传统电商模式带来了新的可能性。
4. **社会价值：**项目体现了技术创新对用户体验提升的促进作用，展示了人工智能技术普惠化应用的广阔前景。

个人成长的深度反思：

在技术能力方面，通过项目实践掌握了全栈开发的完整技能体系，特别是在AI技术集成、现代前端框架应用、数据库设计优化等方面有了质的提升。更重要的是，培养了系统性思考问题的能力，学会了在复杂项目中平衡技术选型、开发效率和产品质量的关系。

在工程思维方面，深刻理解了软件工程不仅仅是编写代码，更是一个涉及需求分析、架构设计、质量控制、用户体验、安全性考虑等多个维度的综合性活动。良好的工程实践是确保项目成功的重要保障。

对未来发展的展望：

技术发展日新月异，特别是AI技术的快速进步为软件开发带来了新的机遇和挑战。未来将继续关注技术发展趋势，不断学习新技术、新方法，将这个项目作为一个起点，在更大的技术舞台上发挥更大的价值。