- 角色 1：前端架构师 (Frontend Architect)
- 角色 2：视觉设计师 (Visual Designer)
- 角色 3：数据工程师 (Model Engineer)
- 角色 4：界面渲染师 (View Specialist)
- 角色 5：全栈控制器 (Controller / Logic Master)
- 建议的展示流程 (Presentation Flow)

这是一个非常标准的 **MVC (Model-View-Controller)** 架构项目，且代码结构清晰，非常适合用来做团队展示。

为了让你们小组的5个人都能参与并有话可说，我将整个项目按照**技术栈和功能模块**拆分为 5 个角色。

以下是具体的任务分配、代码范围以及对应的中文注释解释：

角色 1：前端架构师 (Frontend Architect)

负责板块：HTML 结构搭建 (`index.html`) 职责描述： 负责构建网页的骨架，定义页面的分区（Shop, Orders, Account, Admin），以及模态框和侧边栏的结构。

核心代码与解释：

```
<!-- index.html -->

<!-- 1. 导航栏结构：定义了四个主要功能的切换按钮 -->
<nav class="main-nav">
  <!-- data-view 属性用于后续 JS 判断要切换到哪个视图 -->
  <button class="nav-btn" data-view="shop">Shop</button>
  <button class="nav-btn" data-view="orders">My Orders</button>
  <button class="nav-btn" data-view="account">Account</button>
  <button class="nav-btn" data-view="admin">Admin Dashboard</button>
</nav>

<!-- 2. 单页面应用(SPA)视图容器 -->
<main class="app-main">
  <!-- 商店视图：包含侧边栏筛选器和商品展示区 -->
  <section class="view active" id="view-shop">
    <!-- ...搜索框、筛选按钮、商品网格容器... -->
  </section>

  <!-- 订单视图：用于展示历史订单 -->
  <section class="view" id="view-orders">
    <div id="orders-container" class="card-list"></div>
  </section>
```

```
  <!-- 购物车抽屉：默认隐藏，通过CSS动画滑出 -->
  <aside class="cart-drawer" id="cart-drawer">
    <div id="cart-items" class="cart-items"></div>
    <!-- ...总价计算与结账按钮... -->
  </aside>

  <!-- 结账弹窗：模态框结构 -->
  <div class="modal-backdrop" id="checkout-modal">
     <!-- ...表单输入区域... -->
  </div>
</main>
```

演讲要点：

- 我负责构建“地基”。
- 使用了语义化标签 (`header`, `main`, `section`, `aside`)。
- 利用 `id` 和 `data-` 属性为后续 CSS 样式和 JS 逻辑提供钩子 (Hooks)。
- 采用了单页面结构，所有页面内容都在一个 HTML 文件中，通过 JS 控制显示/隐藏。

角色 2：视觉设计师 (Visual Designer)

负责板块：CSS 样式设计 (`style.css`) 职责描述： 负责网站的“颜值”，定义咖啡主题配色、布局（Grid/Flexbox）、响应式适配以及动画效果。

核心代码与解释：

```
/* style.css */

/* 1. 全局配色与背景：营造咖啡氛围 */
body {
  /* 径向渐变背景，模拟咖啡色的深邃感 */
  background: radial-gradient(circle at top, #5a381f 0%, #2b1a12 40%, #140b07 80%);
  color: #f6f1e9; /* 字体使用米白色，对比度高 */
}

/* 2. 布局系统：使用 Grid 实现侧边栏+内容区的两栏布局 */
.layout-two-columns {
  display: grid;
  grid-template-columns: minmax(220px, 280px) minmax(0, 1fr); /* 左侧固定宽度范围，右侧自适应 */
  gap: 18px;
}

/* 3. 卡片组件设计：商品卡片的样式 */
.card {
  background: radial-gradient(circle at top left, rgba(78, 48, 26, 0.98), rgba(24,
```

```
12, 7, 0.98));
  border-radius: 14px;
  box-shadow: 0 14px 30px rgba(0, 0, 0, 0.6); /* 悬浮阴影效果 */
}

/* 4. 交互动画：购物车抽屉的滑入滑出 */
.cart-drawer {
  transform: translateX(120%); /* 默认移出屏幕右侧 */
  transition: transform 0.22s ease-out; /* 定义平滑过渡动画 */
}
.cart-drawer.open {
  transform: translateX(0); /* 激活时回到屏幕内 */
}

/* 5. 响应式适配：在手机端将两栏布局改为单列 */
@media (max-width: 900px) {
  .layout-two-columns {
    grid-template-columns: minmax(0, 1fr);
  }
}
```

演讲要点：

- 我负责“装修”。
- 使用了 Flexbox 和 Grid 布局系统，保证页面整齐。
- 配色方案模仿了咖啡豆的烘焙色（深棕、金色、米白）。
- 制作了平滑的交互动画（如购物车滑出），并适配了手机端显示。

角色 3：数据工程师 (Model Engineer)

负责板块：数据逻辑与业务规则 (`model.js`) 职责描述： 模拟后端数据库，管理商品数据、购物车状态、用户信息以及核心业务计算（如总价、库存扣减）。

核心代码与解释：

```
// model.js

const AppModel = (() => {
  // 1. 数据源：模拟数据库中的商品列表
  const products = [
    { id: 1, name: "Ethiopia Yirgacheffe", price: 18.5, stock: 20, ... },
    // ...其他商品数据
  ];

  // 内存中的购物车和订单数组
  const cart = [];
  const users = [];
```

```
  // 2. 核心业务逻辑：计算购物车总价
  const calculateCartSummary = () => {
    let count = 0;
    let total = 0;
    cart.forEach((item) => {
      const product = findProduct(item.productId); // 关联商品信息
      total += product.price * item.quantity;      // 累加金额
    });
    return { count, total };
  };

  // 3. 核心业务逻辑：创建订单
  const createOrder = (customerInfo) => {
    // 扣减库存逻辑
    cart.forEach((item) => {
      const product = findProduct(item.productId);
      product.stock -= item.quantity; // 模拟实时库存更新
    });
    // 生成订单对象
    const order = { ...customerInfo, status: "Pending", ... };
    orders.unshift(order); // 存入历史订单
    clearCart(); // 清空购物车
    return order;
  };

  // 4. 数据持久化：使用 LocalStorage 保存用户注册信息
  const saveUsersToStorage = () => {
    localStorage.setItem("coffee_users", JSON.stringify(users));
  };

  // 暴露对外的接口
  return { getProducts, addToCart, createOrder, loginUser, ... };
})();
```

演讲要点：

- 我是“幕后大脑”和“虚拟数据库”。
- 虽然没有真正的后端，但我用 JavaScript 对象模拟了数据库表（Products, Orders, Users）。
- 负责处理所有纯数据逻辑，比如“计算总金额”、“检查库存是否充足”、“验证登录密码”。
- 数据与界面完全解耦，我不关心页面长什么样，只关心数据对不对。

角色 4：界面渲染师 (View Specialist)

负责板块：DOM 操作与页面渲染 (`view.js`) 职责描述： 负责将 Model 层的数据转化为用户能看见的 HTML 元素，处理页面的动态更新（如重新渲染商品列表、更新购物车数

字)。

核心代码与解释：

```
// view.js

const AppView = (() => {
  // 1. 缓存 DOM 元素，避免重复查询，提高性能
  const els = {
    productGrid: document.getElementById("product-grid"),
    cartCount: document.getElementById("cart-count"),
    // ...其他元素
  };

  // 2. 动态渲染商品列表
  const renderProducts = (products, wishlistIds) => {
    els.productGrid.innerHTML = ""; // 清空现有内容
    products.forEach((p) => {
      // 创建商品卡片的 HTML 字符串
      const card = document.createElement("article");
      card.className = "card";
      card.innerHTML = `
        <div class="card-title">${p.name}</div>
        <div class="price">$${p.price}</div>
        <!-- 根据库存状态禁用按钮 -->
        <button class="js-add-cart" ${p.stock === 0 ? "disabled" : ""}>
          Add to Cart
        </button>
      `;
      els.productGrid.appendChild(card); // 插入页面
    });
  };

  // 3. 视图切换逻辑
  const setActiveView = (viewId) => {
    // 遍历所有视图，移除 active 类，只给当前视图添加 active
    els.views.forEach((v) => v.classList.toggle("active", v.id === viewId));
  };

  // 暴露渲染方法供 Controller 调用
  return { renderProducts, renderCart, setActiveView, ... };
})();
```

演讲要点：

- 我负责“粉刷匠”的工作。
- Role 1 只是写了空的容器，我的工作是把数据填进去。
- 使用 JavaScript 动态生成 HTML（例如：有一个商品数组，我就生成对应数量的卡片)。
- 我不做数学计算，我只负责“显示”。Model 给我什么数据，我就画什么。

角色 5：全栈控制器 (Controller / Logic Master)

负责板块：事件监听与流程控制 (`controller.js`) 职责描述： 它是 Model 和 View 的桥梁。负责监听用户的鼠标点击、输入等操作，指挥 Model 处理数据，然后指挥 View 更新界面。

核心代码与解释：

```
// controller.js

document.addEventListener("DOMContentLoaded", () => {
  const M = AppModel;
  const V = AppView;

  // 1. 初始化：页面加载时刷新所有数据
  const refreshAll = () => {
    V.renderProducts(M.getProducts(), M.getWishlist()); // 从 Model 拿数据，给 View
渲染
    V.renderCart(M.getCart(), ...);
  };

  // 2. 事件监听：处理“添加到购物车”点击
  V.els.productGrid.addEventListener("click", (e) => {
    // 检查点击的是否是 "Add to Cart" 按钮
    const btn = e.target.closest(".js-add-cart");
    if (btn) {
      const id = parseInt(btn.dataset.id);
      M.addToCart(id); // 指挥 Model：更新购物车数据
      V.renderCart(M.getCart(), ...); // 指挥 View：更新购物车界面
      alert("Added!");
    }
  });

  // 3. 事件监听：处理“搜索筛选”
  V.els.searchInput.addEventListener("input", (e) => {
    // 获取用户输入 -> 过滤数据 -> 重新渲染 View
    const keyword = e.target.value;
    // ...筛选逻辑...
    V.renderProducts(filteredData);
  });

  // 4. 事件监听：处理“结账”
  V.els.checkoutForm.addEventListener("submit", (e) => {
    e.preventDefault(); // 阻止表单默认提交
    const order = M.createOrder({ ... }); // 调用 Model 创建订单
    if(order) {
       V.setCheckoutOpen(false); // 关闭弹窗
       refreshAll(); // 刷新所有视图
    }
  });
});
```

演讲要点：

- 我是“交通指挥官”或“中枢神经”。
- HTML/CSS 是死的，Model 只是数据，View 只是画图工具，是我让整个应用“动”起来的。
- 工作流程是：监听用户操作 (Click) -> 调用 Model 修改数据 (Update Data) -> 调用 View 更新界面 (Re-render)。
- 负责将前端的事件（View层）和后端的数据（Model层）连接起来。

建议的展示流程 (Presentation Flow)

1. **Role 1 (HTML)** 开场，展示页面基础结构，解释为什么分了这几个区域。
2. **Role 2 (CSS)** 展示页面最终效果，强调设计风格和响应式布局，可以现场演示一下缩小浏览器窗口。
3. **Role 3 (Model)** 解释：“既然我们没有真正的服务器，我是如何用 JS 对象来模拟数据库和库存管理的。”
4. **Role 4 (View)** 解释：“我是如何把 Role 3 的数据变成用户能看到的卡片的。”
5. **Role 5 (Controller)** 总结：“我把大家串联在了一起。” 现场演示一个完整的流程：**搜索咖啡 -> 加入购物车 -> 结账下单 -> 在管理员后台查看订单**。