最终项目：天堂苗圃购物应用

预计所需时间: 110分钟

介绍

在这个最终项目中，您将为一个在线植物商店创建一个购物车应用程序，该商店提供各种室内植物。

*天堂苗圃*购物车的功能将包括：

- 一个带有链接到产品列表页面的按钮的登录页面
- 一个导航栏，包含链接到登录页面、产品列表页面和购物车页面
- 每种植物的卡片，展示不同的植物及其图像、名称、描述、价格和一个**添加到购物车**按钮。
- 至少两个部分描述该部分中的植物。例如，“芳香植物”和“药用植物”。
- 一个购物车页面，显示购物车中的产品。
- 购物车应为购物车中的每种植物提供一个卡片。每个卡片应包含缩略图、单价、该类型所有植物的总价以及增加和减少数量的按钮和**删除**按钮。
- 一个**继续购物**和**结账**按钮

您将实现从实践项目中获得的知识和技能，以处理动态功能，例如在导航栏图标中显示购物车数量，并在用户更新项目数量时更新购物车中所有项目的总价。

项目提交

要提交您的应用程序以进行评分，您需要部署它。您需要将URL作为同行评审项目的一部分提供。您可以使用GitHub Pages进行托管。您可以在本模块早期的*设置GitHub环境*实验中找到说明。

您可以选择在其他地方托管并提供该链接。只需确保您的同伴可以轻松访问您提供的应用程序URL。

部署时，请务必记录应用程序的URL。

学习目标

完成此实验后，您将能够：

- React组件：使用组件组合和嵌套创建功能性React组件。
- 使用Hooks进行状态管理：实现React Hooks，特别是useState和useEffect hooks。您将使用hooks管理组件级状态，以控制元素的可见性。
- Redux集成：在应用程序中集成Redux，使用Redux概念，如动作、减速器和存储。
- 渲染动态数据：动态渲染从对象数组中获取的数据到UI。您将遍历数组以生成组件列表。
- 处理事件和条件渲染：处理用户事件，如按钮选择并触发相应的动作。

先决条件

- 基本的GitHub知识和您自己的GitHub帐户
- 理解React函数组件、props、hooks和React Redux Toolkit
- 带有控制台的Web浏览器（如Chrome DevTools或Firefox控制台）

关于此实验环境的重要通知

Skills Network Cloud IDE（基于 Theia 和 Docker）是一个开源的集成开发环境（IDE），为课程和项目相关的实验提供了一个动手实验的环境。

请注意，此实验环境的会话不是持久的。每次您连接到此实验时，都会为您创建一个新的环境。如果您在未保存到 GitHub 或其他外部源的情况下退出环境，您将会丢失数据。请计划在单个会话中完成这些实验，以避免丢失数据。

在 GitHub 中设置您的环境

1. 复制仓库

您需要为您的 React 应用复制 GitHub 仓库。该项目的骨架代码的 GitHub 仓库在这里：

- https://github.com/ibm-developer-skills-network/e-plantShopping.git
- 在访问上述链接后，点击复制按钮。

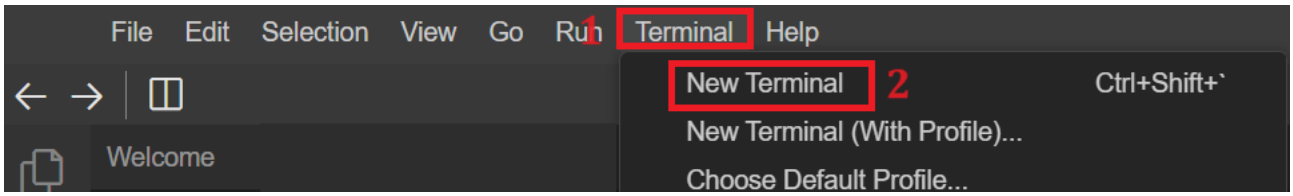


2. 该仓库包含此项目的 React 应用的基本布局。
3. 您将使用这个复制的仓库来推送您的最新代码，以记录您的工作。定期保存您的所有文件。您的文件必须保存才能执行 git 命令。
4. 执行 `git add`、`git commit` 和 `git push` 命令，将您应用文件夹中的更改更新到您的 GitHub 仓库，以便进行适当的代码管理。
5. 作为同行评审项目的一部分，您需要使用 GitHub Pages 部署您的应用，以便您的同伴可以审查 UI。您可以查看 这些说明 来获取有关 GitHub 的帮助以及 React 应用的部署说明。

如果您已注销并想继续进行项目，请克隆您之前推送代码的复制仓库。克隆后，您可以直接使用 `git push origin` 而无需执行 `git remote add...` 来推送您的更改。

创建你的 React 项目

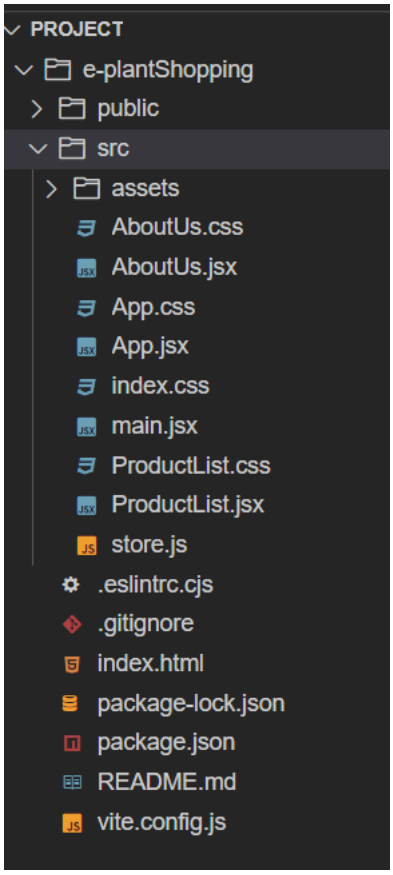
1. 如果你没有打开终端，请在窗口右上角选择“终端”选项卡，然后选择“新建终端”。下面的截图显示了你在屏幕上找到这些选项的位置。



2. 现在使用给定的命令克隆已分叉的仓库，将 `<forked-repo-link>` 替换为你自己的仓库链接。

```
git clone <forked-repo-link>
```

3. 确保你的应用程序名称与项目匹配。
4. 克隆仓库后，你将看到如下截图所示的文件夹结构。



5. 在终端中输入命令进入应用程序文件夹。该命令将设置你的终端路径，以在 `<forked-folder-name>` 文件夹中运行 React 应用程序。

```
cd e-plantShopping
```

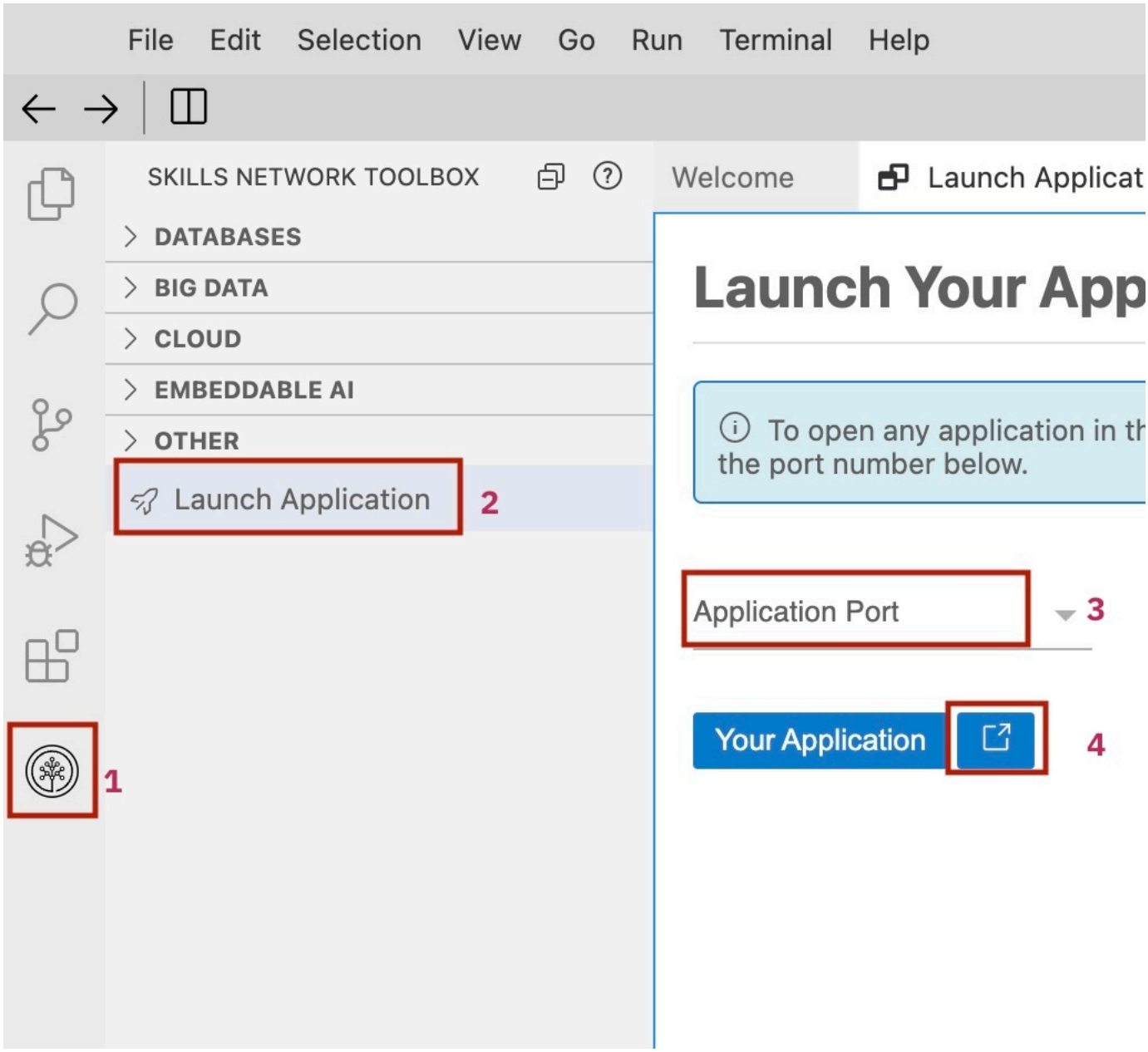
6. 为了确保你克隆的代码正常工作，你需要按照以下步骤进行操作：
   - 在终端中输入以下命令并按 Enter 安装执行应用程序所需的所有包。

     ```
     npm install
     ```

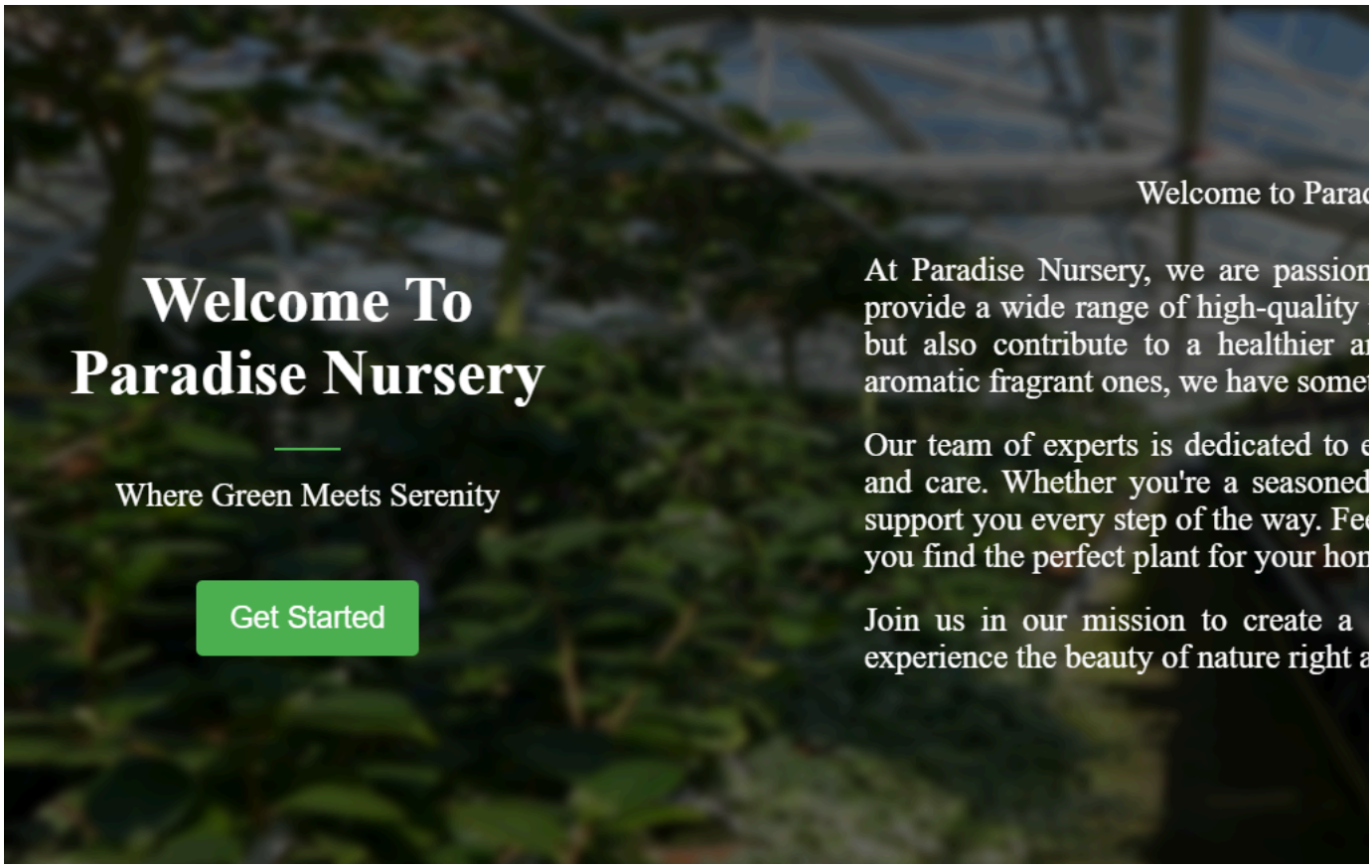
   - 然后执行以下命令运行应用程序，提供端口号 4173。

     ```
     npm run preview
     ```

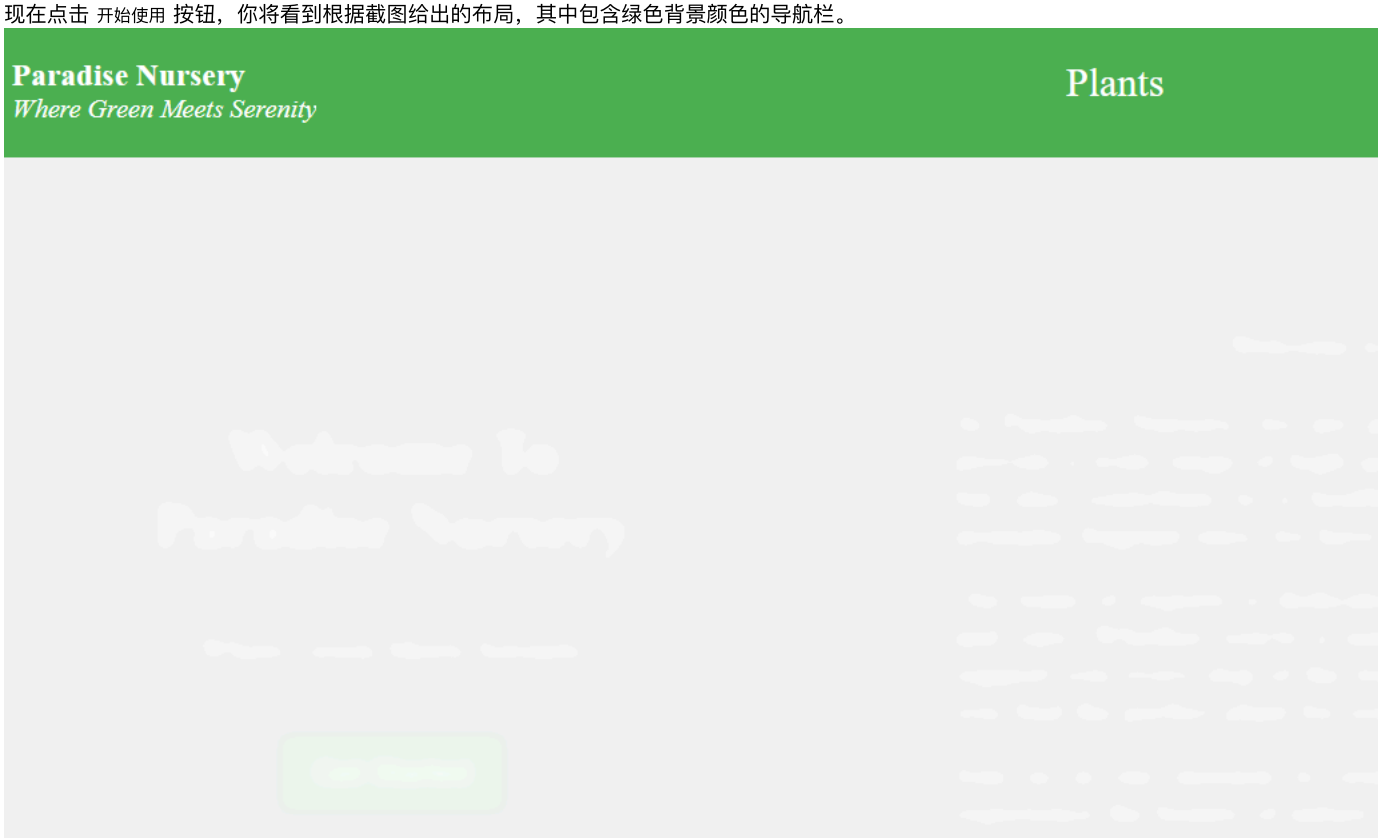
7. 要查看你的 React 应用程序，请点击左侧面板上的技能网络图标（参见第 1 点）。此操作将打开 **技能网络工具箱**。接下来，点击 **启动应用程序**（参见第 2 点）。在 **应用程序端口** 中输入端口号 **4173**（参见第 3 点），然后点击 。



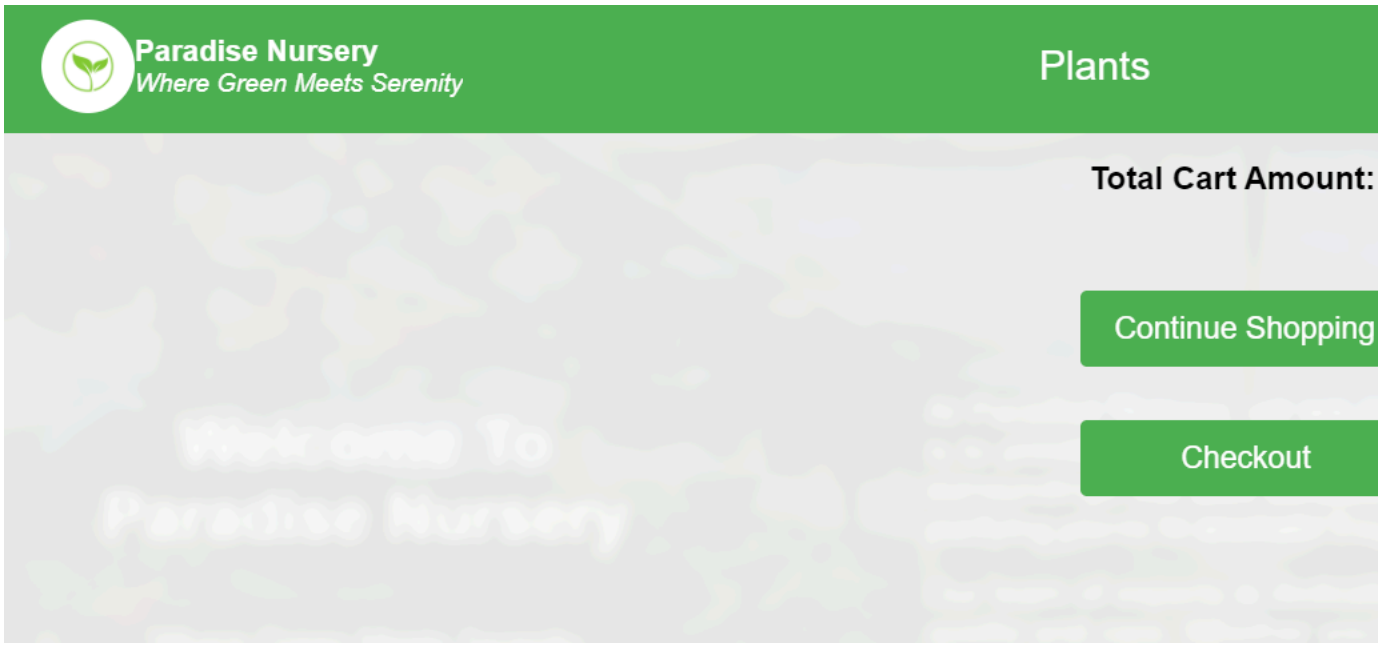
8. 输出将根据给定的截图显示背景图片。



9. 现在点击 `开始使用` 按钮，你将看到根据截图给出的布局，其中包含绿色背景颜色的导航栏。



10. 导航栏包含三个链接：
    - `Paradise Nursey` – 这将带你返回应用程序的登录页面。
    - `Plants` – 这将导航到显示与页面相关信息的页面。
    - `购物车图标` – 这将导航到购物车项目部分。
11. 当你点击 `购物车图标` 时，输出将根据给定的截图显示。



完成后续页面上解释的所需任务。你的同伴将根据这些任务使用提供的评分标准进行评估。

任务 1：ProductList 组件布局

产品页面将允许您的用户购买您销售的不同植物。每种植物将以自己的“卡片”形式显示，相关数据存储在植物对象中。您将把植物对象存储在一个数组中。按照以下步骤操作数组和植物对象。

1. 显示植物数组

- 导航到 `ProductList.jsx` 组件，您将看到一个名为 `plantsArray` 的数组，其中包含植物的详细信息。
- 每个植物对象包含类别、属性名称、图像 URL、描述和成本。

2. 在类名为 **product-grid** 的 `div` 标签内显示植物详情。

- 利用数组方法遍历植物数组。

  提示：使用 `map()` 方法迭代数组。

- 在页面上渲染每种植物的详细信息，包括名称、图像、描述和成本。

3. 为每种植物显示一个 **添加到购物车** 按钮。

▼ 显示植物和添加到购物车按钮的解决方案

```
{plantsArray.map((category, index) => (
<div key={index}>
    <h1><div>{category.category}</div></h1>
    <div className="product-list">
        {category.plants.map((plant, plantIndex) => (
        <div className="product-card" key={plantIndex}>
            <img className="product-image" src={plant.image} alt={plant.name} />
            <div className="product-title">{plant.name}</div>
            {/*Similarly like the above plant.name show other details like description and cost*/}
            <button  className="product-button" onClick={() => handleAddToCart(plant)}>Add to Cart</button>
        </div>
        ))}
    </div>
</div>
))}
```

- 将上述代码包含在类名 **product-grid** 中。

4. 创建一个名为 `addedToCart` 的变量，用于状态管理，使用 **useState** 钩子来跟踪哪些产品已被添加到购物车。

▼ useState 钩子的示例解决方案

```
const [addedToCart, setAddedToCart] = useState({});
```

5. 添加到购物车功能

- 创建 `handleAddToCart` 函数，以实现用户点击 **添加到购物车** 按钮时将植物添加到购物车的功能。该函数应接受一个参数，该参数包含所选植物的信息。然后将这些信息派发到函数组件 `CartSlice` 中的 addItem。
- 此外，反映产品已添加到购物车。通过将产品名称设置为键，并将其值设置为 true，来更新 `setAddedToCart` 状态。

▼ 添加到购物车的示例解决方案

```
const handleAddToCart = (product) => {
  dispatch(addItem(product));
```

```
    setAddedToCart((prevState) => ({
       ...prevState,
       [product.name]: true, // Set the product name as key and value as true to indicate it's added to cart
     }));
  };
```

注意：确保从 `CartSlice.jsx` 中 `import addItem` reducer。

6. `handleAddToCart()` 函数将携带用户想要添加到购物车中的植物的详细信息。并将植物详细信息通过 `CartSlice.jsx` 在全局级别添加到购物车中。
7. 确保通过将代码推送到你的 GitHub 仓库来保存这些更改。

任务 2：使用 Redux 的状态管理

1. 您在 `CartSlice.jsx` 文件中有基本布局。
2. 定义 Reducer 函数

- 现在实现切片的 reducer 属性，用于添加、移除和更新购物车中商品的数量。
- 当用户想要添加或移除 `cartItems` 组件中的植物数量时，将调用这些 reducer 函数。
- `addItem()` reducer 将新植物项添加到您在上一步中初始化的 `items` 数组中。
- 当用户在植物列表页面选择 **添加到购物车** 时，应调用 `addItem()` 函数。随后，`handleAddToCart()` 被调用，并传入植物类型作为参数。
- `handleAddToCart()` 函数将把植物详情分发给 `CartSlice.jsx` 中的 `addItem()` reducer 函数。

▼ addItem() reducer 示例解决方案

```
addItem: (state, action) => {
  const { name, image, cost } = action.payload;
  const existingItem = state.items.find(item => item.name === name);
  if (existingItem) {
    existingItem.quantity++;
  } else {
    state.items.push({ name, image, cost, quantity: 1 });
  }
},
```

- 现在你需要完成 `removeItem()` 和 `updateQuantity()` reducers 的代码。
- `removeItem()`：这个 reducer 根据商品名称从购物车中移除一个商品，当用户想要从购物车中移除产品时会被调用。

▼ removeItem() reducer 示例解决方案

```
state.items = state.items.filter(item => item.name !== action.payload);
```

- `updateQuantity()`: 创建此函数时，首先从 `action.payload` 中提取项目的名称和数量。然后，在 `state.items` 数组中查找与提取的名称匹配的项目。如果找到该项目，将其数量更新为 payload 中提供的新数量。这确保项目的数量根据操作正确更新。

▼ updateQuantity() reducer 示例解决方案

```
const { name, quantity } = action.payload;
const itemToUpdate = state.items.find(item => item.name === name);
if (itemToUpdate) {
  itemToUpdate.quantity = quantity;
}
```

3. 处理操作

- 导出操作创建者，`addItem()` 以在 `ProductList.jsx` 中使用，`removeItem()` 和 `updateQuantity()` 以在 `CartItem.jsx` 中使用。
- 还要将 reducer 作为默认导出，以便在 `store.js` 中使用。

4. 确保通过将代码推送到您的 GitHub 存储库来保存这些更改。

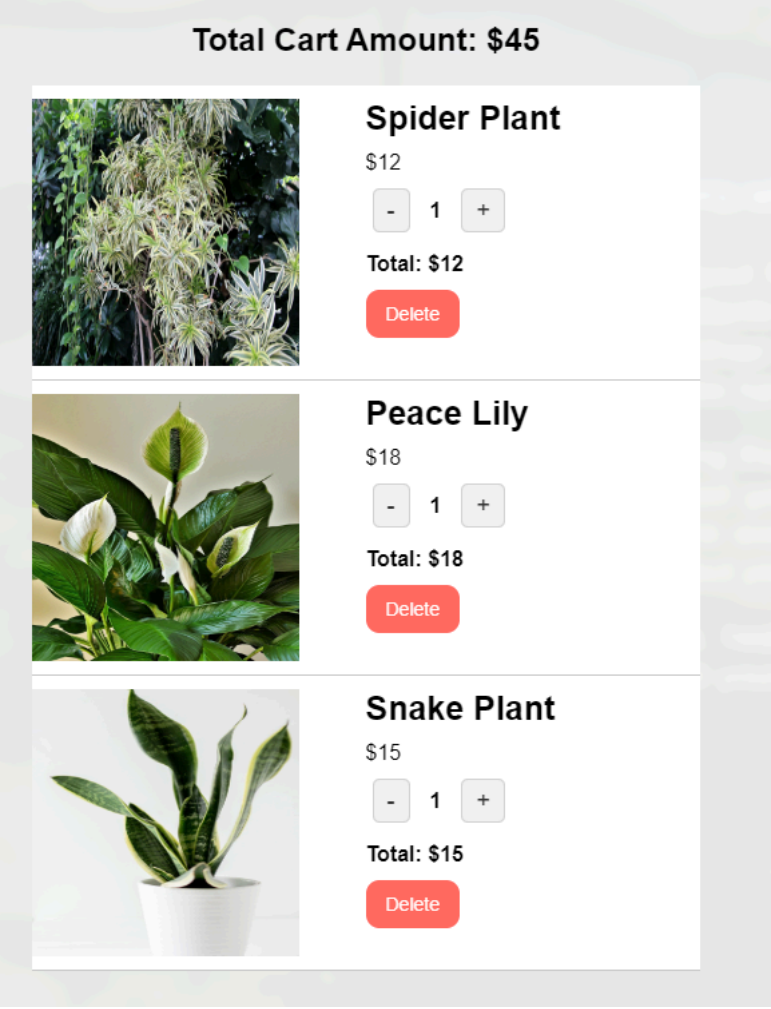
任务 3: CartItems 组件

接下来，您将完成 `CartItem.jsx` 组件的开发，该组件显示购物车中持有的商品。该组件具有典型购物车的一些功能：

- 计算购物车中所有商品的总价。
- 计算购物车中每种植物类型的 subtotal。
- 继续购物
- 增加和减少购物车中每种植物类型的数量
- 从购物车中完全移除（删除）某种植物类型。

您将从 Redux 文件中派发增加、减少和更新数量的细节。

完成后，购物车页面应类似于以下内容：



1. 购物车中所有商品的费用

- 在 `calculateTotalAmount()` 中，您需要一个函数来计算购物车中所有商品的费用。计算方式有很多种。

▼ 计算总费用的示例算法描述

您可以遍历商品数组，使用 `item.quantity` 属性找到每个商品的数量，并用 `item.cost` 属性乘以该植物类型的费用。

2. 继续购物

- 用户应该能够在购物车页面返回植物列表页面继续购物。因此，在 `handleContinueShopping()` 函数中调用从父组件传递的函数。

3. 结账

- 在本项目中，您不需要提供 `handleCheckoutShopping()` 函数，但您可以为进一步探索和练习添加该功能。现在，只需添加以下代码以提醒用户该功能将在稍后添加。

```
const handleCheckoutShopping = (e) => {
  alert('Functionality to be added for future reference');
};
```

4. 增加和减少

- 对于 `handleIncrement()` 和 `handleDecrement()` 函数，您需要在 `CartSlice.jsx` 文件中派发 `updateQuantity()` reducer。在函数参数中，分别将 `item.quantity` 值加一或减一。
- 此外，对于 `handleDecrement()`，您需要一个 if-else 语句来处理当商品数量减少到 0 的情况。在这种情况下，您需要派发 removeItem() 方法。

5. 从购物车中移除植物

- 对于 `handleRemove()` 函数，您需要派发 `removeItem()` 方法。

6. 项目小计

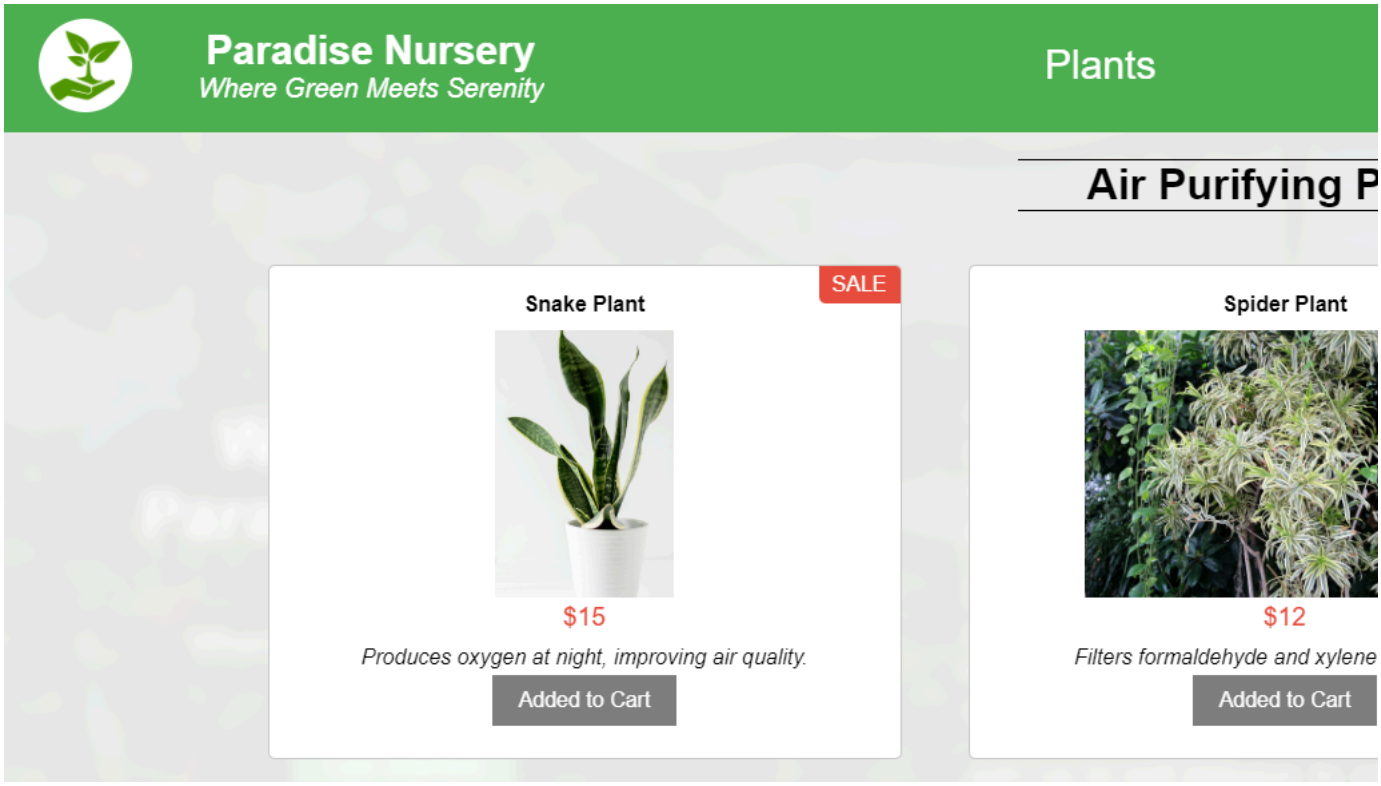
- 使用 `calculateTotalCost()` 函数计算特定类型植物的总成本，通过将植物的单价乘以购物车中该类型植物的数量。

7. 事件处理

- 实现事件处理程序，用于增加和减少购物车中商品的数量。
- 当用户更改购物车中某种植物的数量时，以下数据需要更新：购物车图标、该植物类型的数量、小计和总成本。
- 实现一个事件处理程序以从购物车中移除该商品。

8. 总数量计数器

- 维护一个专门用于计数添加到购物车的商品总数的变量。
- 当用户添加或移除植物时，更新此变量。
- 在导航栏的购物车图标上显示总数量。



9. 确保通过将代码推送到您的 GitHub 仓库来保存这些更改。

任务 4：在组件中集成 Redux 功能

1. ProductList 组件
    - 在 Redux 存储中初始化购物车商品状态。
    - 派发 `addItem` 动作以将商品添加到购物车。
    - 从 Redux 存储中检索购物车中所有商品的数量。
2. CartItem 组件
    - 派发 `updateQuantity` 动作以更新购物车商品的数量。
    - 派发 `addItem` 动作以将购物车中的商品添加。
    - 派发 `removeItem` 动作以从购物车中移除商品。

注意：确保通过将代码推送到您的 GitHub 仓库来保存这些更改

任务 5：导入细节到 store.js

1. 导入必要的函数和文件：
    - 从 `@reduxjs/toolkit` 包中导入 `configureStore()` 函数以设置 Redux 存储。
    - 从 `CartSlice.jsx` 文件中导入 `cartReducer`，管理与购物车相关的状态切片。

```
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from './CartSlice';
```

2. 配置商店：
    - `configureStore()` 函数用于设置 Redux 存储。
    - 在传递给 `configureStore()` 的配置对象中，`reducer` 键指定了 reducer 函数。在这种情况下，`cartReducer` 被分配来管理状态的 `cart` 切片。

```
const store = configureStore({
    reducer: {
        cart: cartReducer,
    },
});
```

3. 导出商店：
    - 配置好的 Redux 商店使用 `export default store;` 导出，以便在整个应用程序中管理状态。

```
export default store;
```

注意：
– 该代码已在仓库中预配置并准备使用。
– 确保通过将代码推送到您的 GitHub 仓库来保存这些更改

任务 6：设置全局存储

1. 导航到 `src` 文件夹中的 `main.jsx` 文件。
2. `react-redux` 库中的 `Provider` 组件已经被导入。该组件使应用程序中的所有组件都可以访问 Redux 存储。

```
import { Provider } from 'react-redux';
```

3. Redux 存储从 `store.js` 文件导入。这个存储保存应用程序的状态，使用在 `CartSlice.jsx` 文件中定义的 reducer。

```
import store from './store.js';
```

4. `App` 组件被 `Provider` 组件包裹，Redux 存储作为一个属性传递。这允许应用中的所有组件访问和与 Redux 管理的全局状态进行交互。

```
<Provider store={store}>
  <App />
</Provider>
```

注意：
– 该代码已在仓库中预配置并准备使用。
– 确保通过将代码推送到您的 GitHub 仓库来保存这些更改

项目完成

恭喜！

您完成了这个项目！干得不错。

如果您还没有这样做，请部署您的应用程序。您可以使用 GitHub Pages 或您自己的托管网站进行部署。请记下应用程序的 URL，您将在提交项目以进行同行评审时需要它。

有关使用 GitHub Pages 部署应用程序的说明，请参见本实验的 最后一步, *设置 GitHub 环境*，在本课程的早期。

部署您的 React 应用程序很重要，因为您的同行将根据此评审您的工作。

总结

- 您创建了功能性 React 组件，并进行了组合和嵌套。
- 您实现了 React Hooks，特别是 `useState` 和 `useEffect` hooks 来管理组件级状态。
- 您在 React 应用程序中集成了 Redux，并应用了 Redux 概念，如动作、 reducers 和存储。
- 您动态渲染了从对象数组中获取的数据到 UI 上。您遍历数组以生成组件列表。
- 您处理了用户事件，例如按钮选择，并触发相应的操作以添加和移除购物车中的项目。

Author

Richa Arora

© IBM Corporation. All rights reserved.