Final Project: Paradise Nursery Shopping Application

Skills Network

Estimated time needed: 110 minutes

Introduction

In this final project, you will create a shopping cart application for an online plant shop which offers a variety of house plants.

The *Paradise Nursery* shopping cart features will include:

- A Landing page with a button linking to the product listing page
- A navigation bar with links to the landing, product listing, and shopping cart pages
- A card for each plant that showcases the different plants along with their images, name, description, cost and an **Add to cart** button.
- A minimum of two sections describing the plants in that section. For example, "Aromatic Plants" and "Medicinal Plants".
- A cart page which displays the products in the cart.
- The cart should have a card for each type of plant in the cart. Each card should have the thumbnail, the unit cost, the cost for all of the plants of that type and buttons to increase and decrease the quantity along with **Delete** button.
- A **Continue Shopping** and **Checkout** buttons

You will implement the knowledge and skills you gained from working on the practice project to handle dynamic functionalities, like the show cart quantity in the icon on the navbar and updating the cost of all of the items in the cart when the user updates the number of items.

Project Submission

To submit your application for grading, you will need to deploy it. You will need to provide the URL as part of the peer review project. You can use GitHub Pages to host. You can find the instructions from the *Setting Up the GitHub Environment* lab earlier in this module.

You may choose to host it elsewhere and provide that link instead. Just make sure your peers can access the application easily using the URL you provide.

When you deploy, be sure to make a note of the application's URL.

Learning objectives

After completing this lab, you will be able to:

- React Components: Create functional React components using component composition and nesting.
- State Management with Hooks: Implement React Hooks, specifically the useState and useEffect hooks. You will manage component-level state using hooks to control the visibility of elements.
- Redux Integration: Integrate Redux within an application using Redux concepts like actions, reducers, and the store.
- Rendering Dynamic Data: Dynamically render data fetched from an array of objects into the UI. You will map over arrays to generate lists of components.
- Handling Events and Conditional Rendering: Handle user events such as button selection and trigger corresponding actions.

Prerequisites

- Basic knowledge GitHub and your own GitHub account
- Understanding of React function components, props, hooks and React Redux Toolkit
- Web browser with a console (such as Chrome DevTools or Firefox Console)

Important notice about this lab environment

Skills Network Cloud IDE (based on Theia and Docker) is an open-source IDE (Integrated Development Environment) that provides an environment for hands-on labs in course and project-related labs.

Please be aware that sessions for this lab environment are not persistent. Every time you connect to this lab, a new environment is created for you. You will lose data if you exit the environment without saving to GitHub or another external source. Plan to complete these labs in a single session to avoid losing your data.

Setting up your environment in GitHub

1. Fork the repository

You need to fork the GitHub repository for React your application. The GitHub repository with the skeleton code for this project is here:

- https://github.com/ibm-developer-skills-network/e-plantShopping.git

- After following the link above, click on the fork button.

e-plantShopping
Public
generated from ibm-developer-skills-network/coding-project-template
Watch 1
Fork 4
Star 0

2. This repository contains the basic layout of the React application for this project.

3. You will use this forked repository to push your latest code to keep a record of the work you do. Periodically save all your files. Your files must be saved to perform git commands.

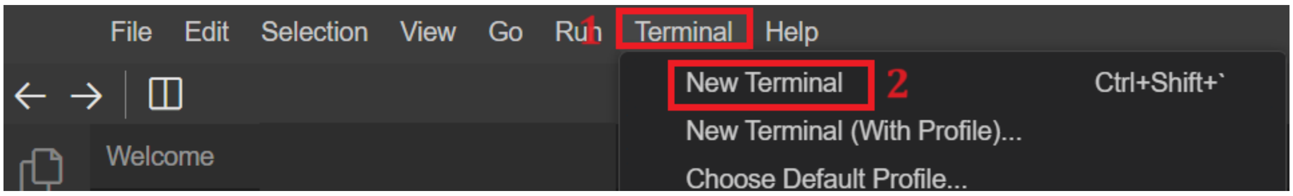
4. Perform `git add`, `git commit`, and `git push` commands to update changes from your application folder to your GitHub repository for proper code management.

5. As part of the peer review project, you will need to deploy your app with GitHub pages so your peers can review the UI. You can review these instructions for assistance with GitHub and for deployment instructions of react application.

If you've logged out and want to resume work on the project, clone the forked repository where you previously pushed the code. After cloning, you can use `git push origin` without the need to execute `git remote add...` to push your changes directly.

Create your React project

1. If you do not have an open terminal, select the "Terminal" tab at the top-right of the window, and then select "New Terminal". The screenshot below shows you where to locate these options on your screen.

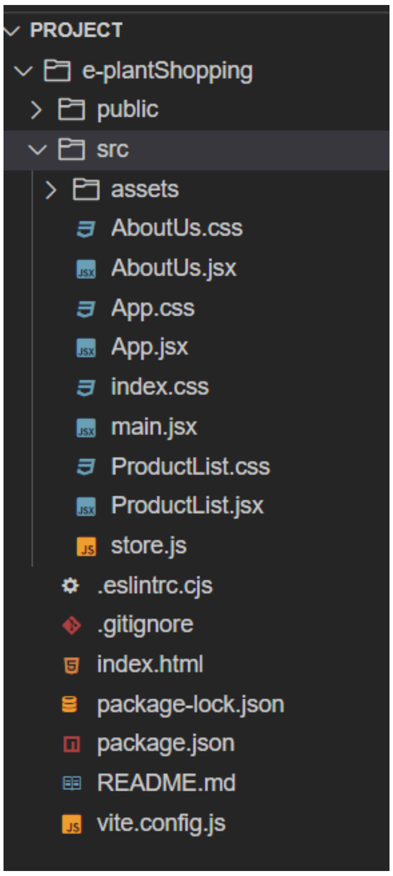


2. Now clone the forked repository using given command by replacing `<forked-repo-link>` with your own repository link.

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

3. Make sure your application name matches your project.

4. After cloning the repository you will see the folder structure like given screenshot.



```
PROJECT
  e-plantShopping
    public
    src
      assets
      AboutUs.css
      AboutUs.jsx
      App.css
      App.jsx
      index.css
      main.jsx
      ProductList.css
      ProductList.jsx
      store.js
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
    vite.config.js
```

5. Write the command to enter the application folder in the terminal. The command will set your terminal path to run the React application in the `<forked-folder-name>` folder.

```
cd e-plantShopping
```

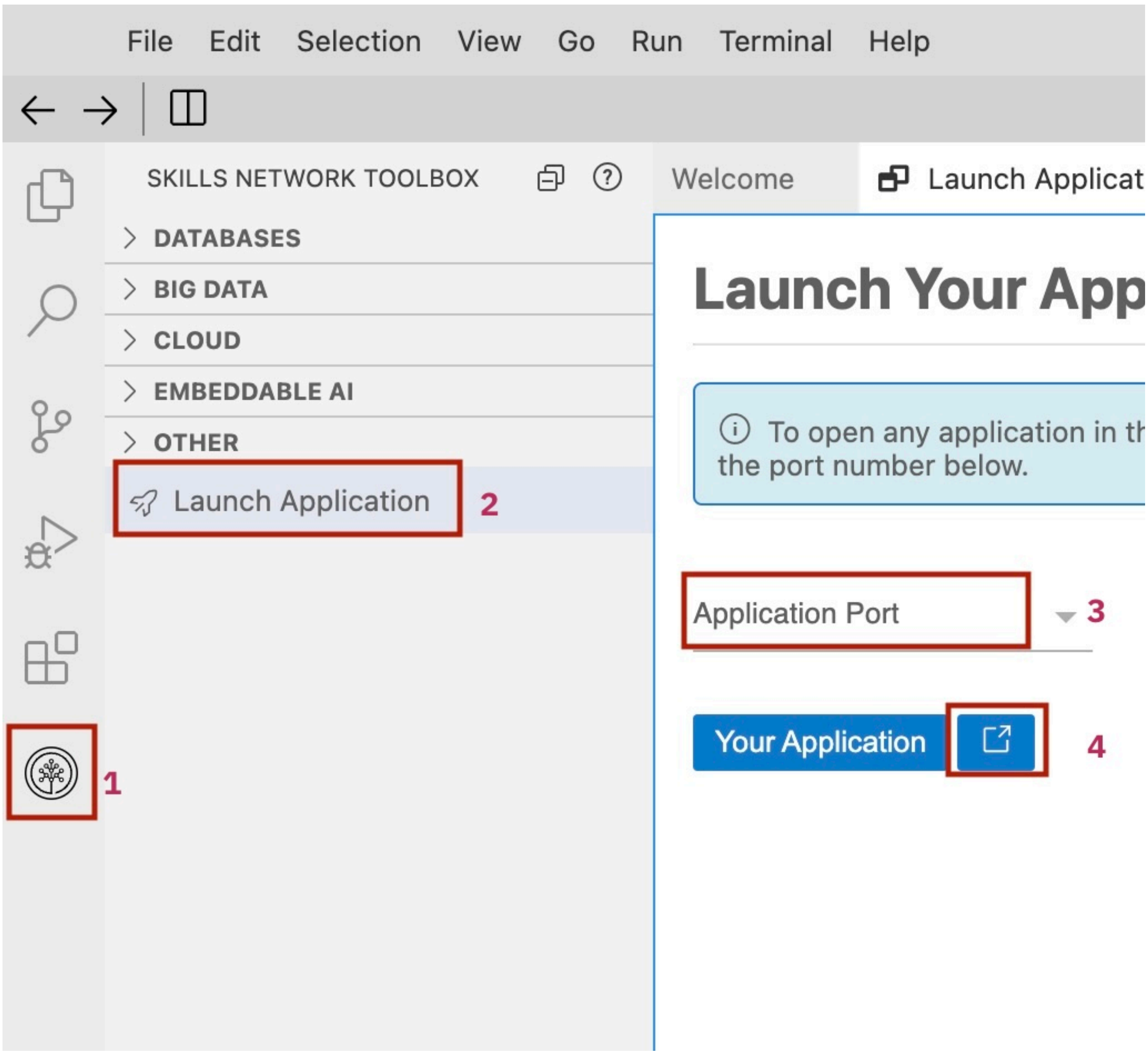
6. To make sure that the code you have cloned is working correctly, you need to follow the given steps:
   - Write the following command in the terminal and select Enter to install all the necessary packages to execute the application.

     ```
     npm install
     ```

   - Then execute the following command to run the application, providing you with port number 4173.

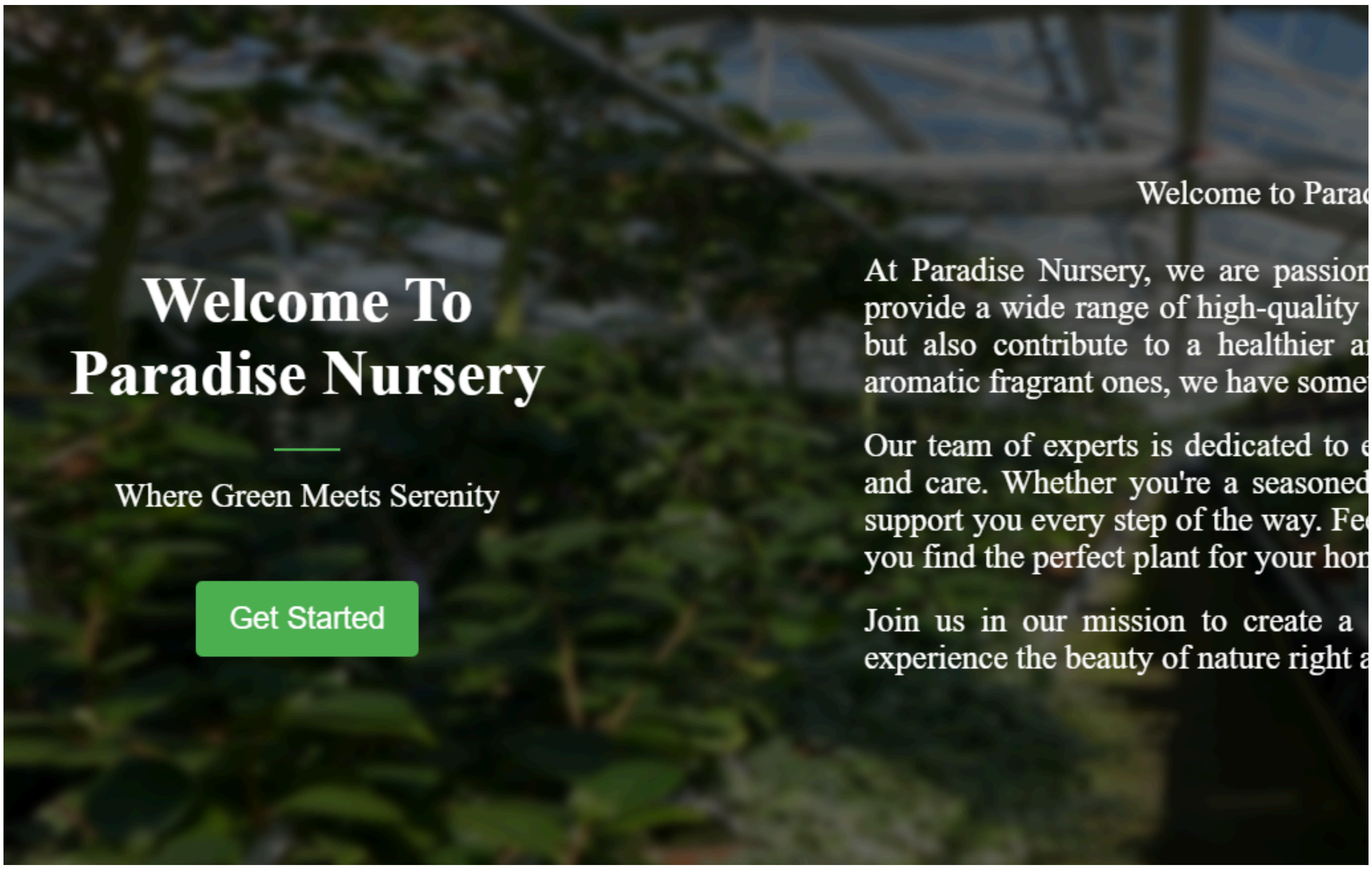
     ```
     npm run preview
     ```

7. To view your React application, click the Skills Network icon on the left panel (refer to number 1). This action will open the **Skills Network Toolbox**. Next, click **Launch Application** (refer to number 2). Enter port number **4173** in **Application Port** (refer to number 3) and click .



To open any application in th
the port number below.

8. The output will be according to given screenshot with background picture.



Welcome To Paradise Nursery

Where Green Meets Serenity

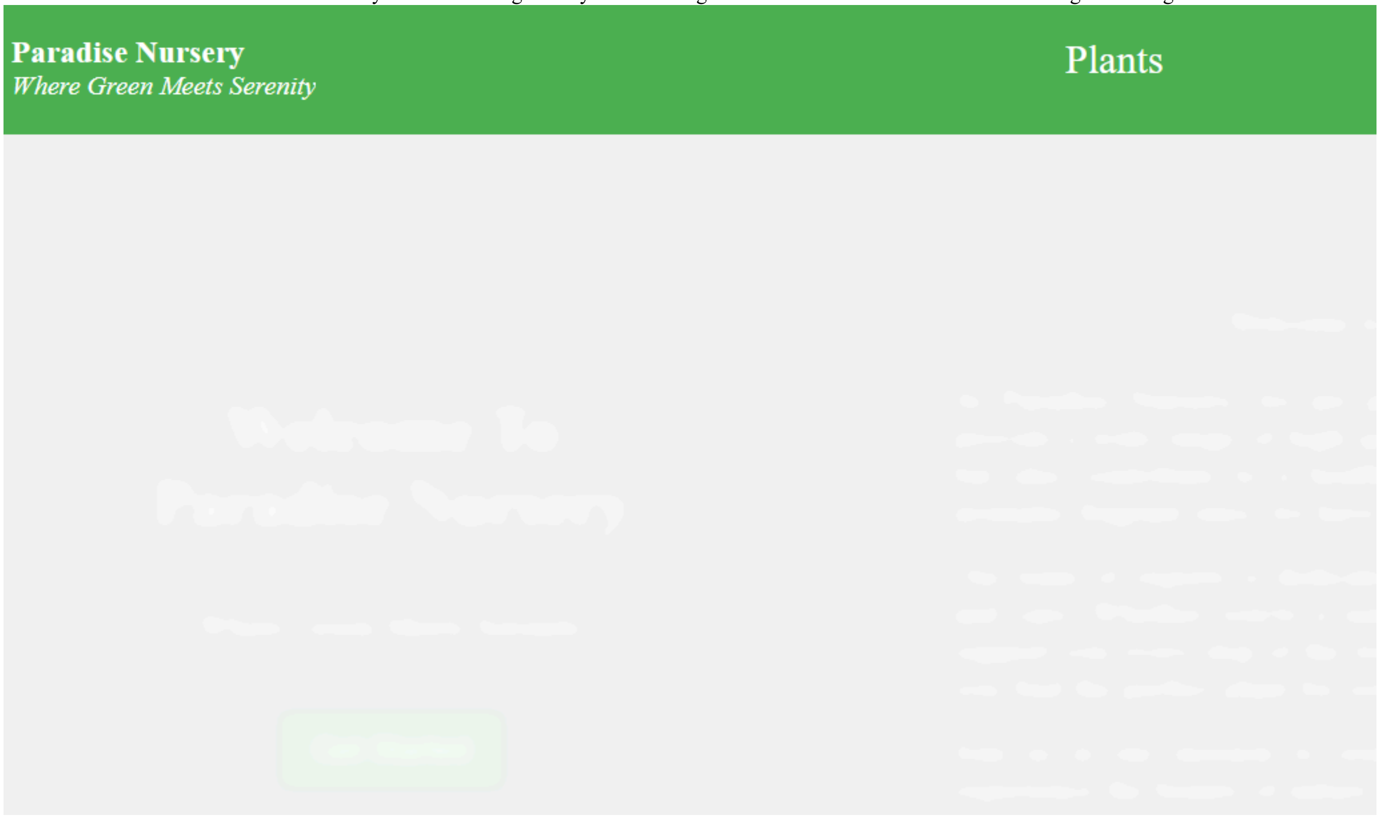
Welcome to Parad

At Paradise Nursery, we are passion
provide a wide range of high-quality
but also contribute to a healthier a
aromatic fragrant ones, we have some

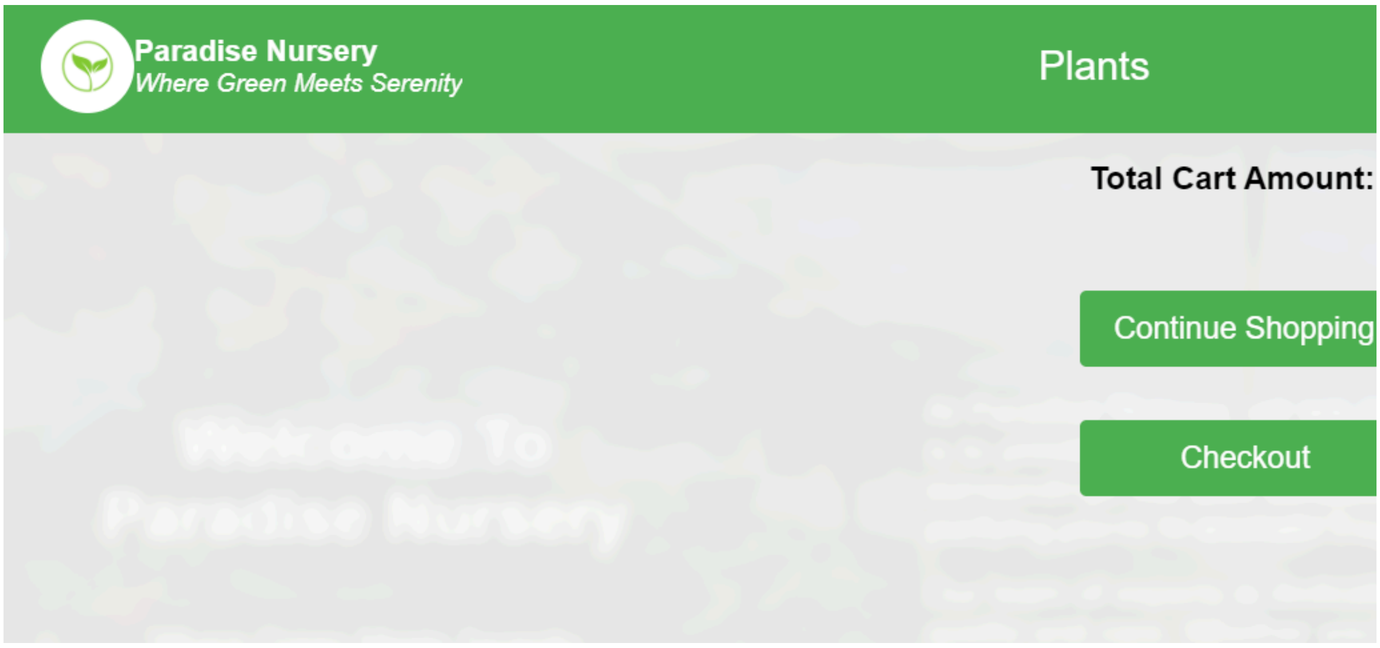
Our team of experts is dedicated to e
and care. Whether you're a seasoned
support you every step of the way. Fe
you find the perfect plant for your hor

Join us in our mission to create a
experience the beauty of nature right a

9. Now click on `Get Started` button and then you will see the given layout according to screenshot which includes navbar with green background color.



10. The navbar contains three links:
    - `Paradise Nursey`- This will take you back to the landing page of the application.
    - `Plants`- This will navigate you to the page where information related to page will be visible.
    - `Cart icon`- This will navigate you to the cart items section.
11. When you will click on `cart icon` output will be visible as per given screenshot.



Complete the required tasks explained on the pages that follow. Your peers will assess you using a rubric provided based on these tasks.

Task 1: ProductList component Layout

The product page will allow your users to shop for the different plants you sell. Each plant will display on its own "card" with its related data stored in the plant object. You will store the plant objects in an array. Follow these steps for the array and plant objects.

1. Display the Plant Array

- Navigate to the `ProductList.jsx` component and you will see an array named `plantsArray` with the plants details.
- Each plant object contains the categories, properties name, image URL, description, and cost.

2. Display Plant Details within `div` tag with class name **product-grid**.

- Utilize array methods to map over the plant array.

  Hint: use the `map()` method to iterate array.

- Render each plant's details on the page, including name, image, description, and cost.

3. Display an **Add to Cart** button for each plant.

▼ Display plants and add to cart button solution

```
{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>
))}
```

- Include above code within class name **product-grid**.

4. Create one variable named `addedToCart` for state management using the **useState** hook to track which products are added to the cart.

▼ Sample solution for useState hook

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

5. Add to Cart Functionality

- Create the `handleAddToCart` function to implement the functionality for adding a plant to the cart when the user selects the **Add to Cart** button. This function should take one parameter that contains the information of the selected plant. This information should then be dispatched to the addItem inside the function component `CartSlice`.
- Additionally, reflect the product has been added to the cart. Update the `setAddedToCart` state to by setting the product name as a key and its value to true.

▼ Sample solution for add to cart

```
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
   }));
};
```

Note: Make sure that you `import` the `addItem` reducer from `CartSlice.jsx`

6. The `handleAddToCart()` function will carry the details of that plant which user want to add in the cart. And the plant details to the cart at a global level using `CartSlice.jsx`.
7. Make sure that you save these changes by pushing your code to your GitHub repository.

Task 2: State management using Redux

1. You have the basic layout in the `CartSlice.jsx` file.
2. Define Reducer Functions

- Now implement the reducer property of the slice for adding, removing, and updating the number of items in the cart.
- These reducer functions will be called when user wants to add or remove the quantity of plants within the `cartItems` component.
- The `addItem()` reducer adds a new plant item to the `items` array which you initialized in the previous step.
- The `addItem()` function should get called when the user selects an **Add to cart** on the plant listing page. Subsequently, the `handleAddToCart()` gets called which has the plant type as a parameter.
- The `handleAddToCart()` function will then dispatch the plant details to the `addItem()` reducer function in `CartSlice.jsx`.

▶ addItem() reducer sample solution

- Now you need to complete code for the `removeItem()` and `updateQuantity()` reducers.
- `removeItem()`: This reducer removes an item from the cart based on its name and gets called when the user wants to remove products from the cart.

▼ removeItem() reducer sample solution

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

- `updateQuantity()`: To create this function, start by extracting the item's name and amount from the `action.payload`. Then, look for the item in the `state.items` array that matches the extracted name. If the item is found, update its quantity to the new amount provided in the payload. This ensures the item's quantity is correctly updated based on the action.

▼ updateQuantity() reducer sample solution

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

3. Handle Actions

- Export the action creators, `addItem()` to use in `ProductList.jsx`, `removeItem()`, and `updateQuantity()`, to use in the `CartItem.jsx`.
- Also export the reducer as the default to use in `store.js`.

4. Make sure you save these changes by pushing your code to your GitHub repository.

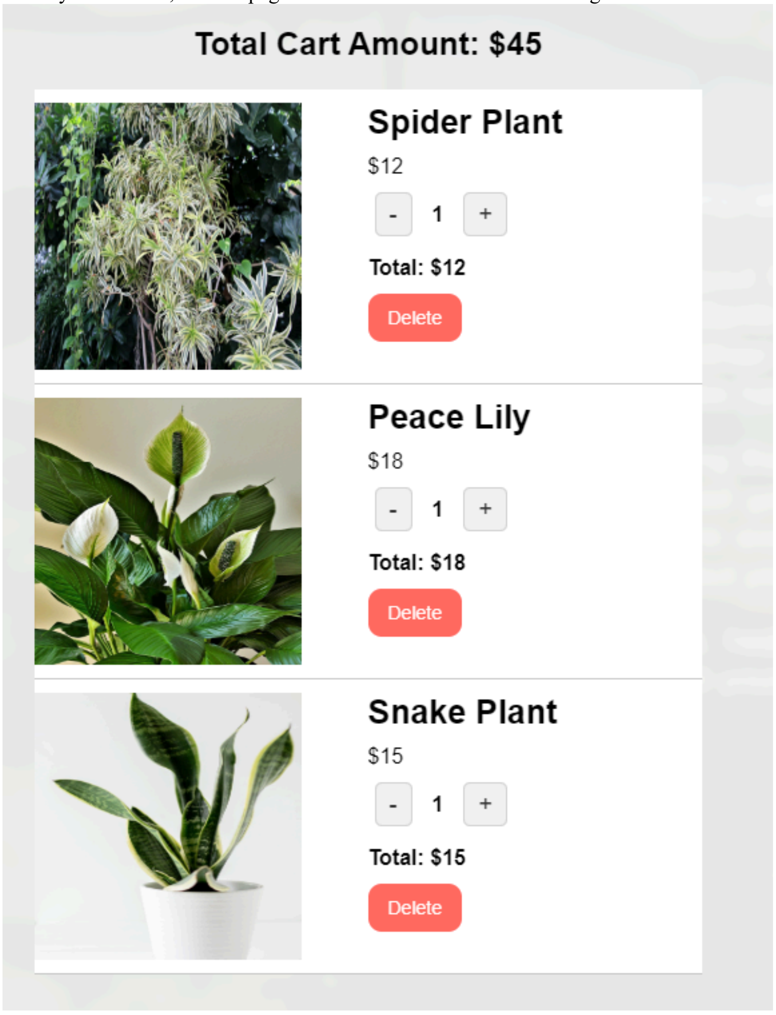
Task 3: CartItems component

Next, you will complete the development of the `CartItem.jsx`component which displays the items held in the shopping cart. This component has a number of functionalities that you find in a typical shopping cart:

- Calculate the total for all items in the cart.
- Calculate the subtotal for each plant type in the cart.
- Continue shopping
- Increment and decrement the number of each plant type in the cart
- Remove (delete) a plant type from the cart altogether.

You will dispatch the increment, decrement, and update quantity details from a Redux file.

When you are done, the cart page should look similar to the following:



1. Cost of all items in cart

- In the `calculateTotalAmount()` you need a function to calculate the cost of all of the items in the cart. There are a number of ways you can calculate this.

▼ Sample algorithm description for calculating the total cost

- Initialize a variable `total` .
- Loop through the `cart` array using `cart.forEach()`.
- Extract the `quantity` and `cost` of each item.
- Multiply `quantity` by `cost`, use `parseFloat(item.cost.substring(1))` to convert price strings like `"$10.00"` into numbers.
- Return the total sum.

2. Continue shopping

- Users should be able to return to the plant listing page to continue shopping while on the shopping cart page. So, in the `handleContinueShopping()` function, call the `onContinueShopping(e)` function passed from the parent component.

3. Checkout

- In this project, you are not required to provide the `handleCheckoutShopping()` function, but you may wish to for further exploration and practice. For now, just add in the following code to alert the user this function will get added at a later date.

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

4. Increment and decrement

- For the `handleIncrement()` and `handleDecrement()` functions, you need to dispatch the `updateQuantity()` reducer in the `CartSlice.jsx` file. In the function argument, either add one to the `item.quantity` value or subtract one, respectively.

- Also, for the `handleDecrement()` you will need an if-else to handle the case.
  – If the item's quantity is greater than 1, dispatch `updateQuantity` to decrease the quantity by 1.
  – Else if the quantity would drop to 0, dispatch the `removeItem` action to remove the plant type from the cart.

▼ Click here to see hints

- Dispatch the `updateQuantity` action (from your Redux slice) to increase or decrease the item's quantity by 1.
- Use the `name` parameter of the item to identify which item's quantity needs to be updated.
- Example code:

```
dispatch(updateQuantity({ name: item.name, quantity: item.quantity + 1 }));
```

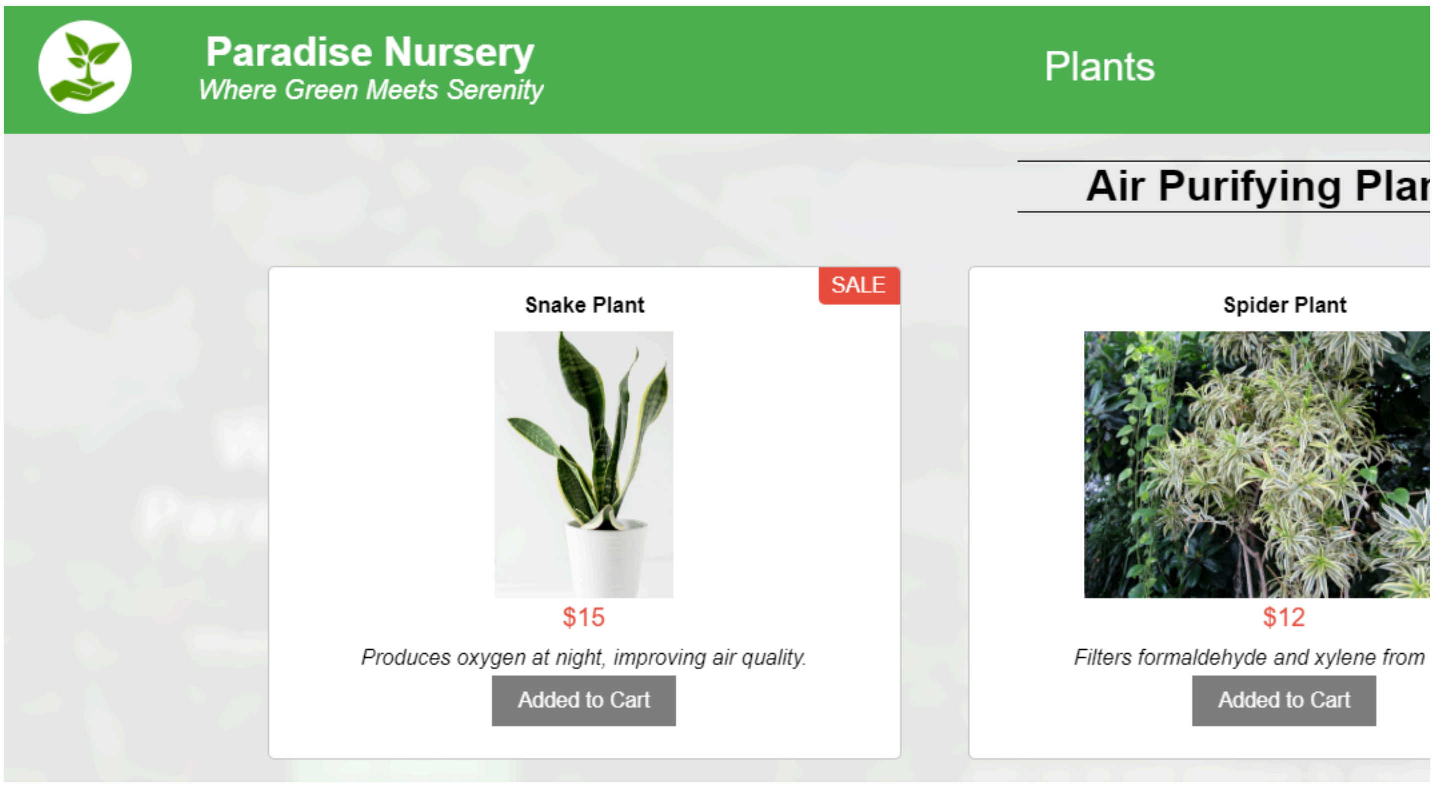
5. Remove plant from the cart

- For the `handleRemove()` function you need to dispatch the `removeItem` action to delete the item from the cart.

6. Item subtotal

- Calculate the total cost for an item by multiplying its quantity with its unit price in the `calculateTotalCost()` function.
- Extract the numeric value from the item's cost string using `parseFloat(item.cost.substring(1))` before performing the multiplication.

  Note: Ensure that these event handlers update the UI in real time. When the user changes the number of a plant type, the following should update accordingly:
  • The individual plant quantity.
  • The item's subtotal.
  • The overall total cost.
  • The the total number of items in the cart icon.


Paradise Nursery
Where Green Meets Serenity
Plants
Air Purifying Plar
SALE
Snake Plant
$15
Produces oxygen at night, improving air quality.
Added to Cart
Spider Plant
$12
Filters formaldehyde and xylene from
Added to Cart

7. Make sure that you save these changes by pushing your code to your GitHub repository.

Task 4: Integrate Redux functionality in your components

1. ProductList Component

    - Initialize the cart items state in the Redux store.
    - Dispatch the `addItem` action to add items to the cart.
    - Retrieve the quantity of all the items in the cart from the Redux store.

2. CartItem Component

    - Dispatch the `updateQuantity` action to update the quantity of the cart item.
    - Dispatch the `addItem` action to add the item from the cart.
    - Dispatch the `removeItem` action to remove the item from the cart.

Note: Make sure that you save these changes by pushing your code to your GitHub repository

Task 5: Import details to store.js

1. Importing Necessary Functions and Files:

    - The `configureStore()` function from the `@reduxjs/toolkit` package is imported to set up the Redux store.
    - The `cartReducer` from the `CartSlice.jsx` file which is imported, manages the state slice related to the shopping cart.

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

2. Configuring the Store:

    - The `configureStore()` function is used to setup the Redux store.
    - Inside the configuration object passed to `configureStore()`, the `reducer` key specifies the reducer functions. In this case, the `cartReducer` is assigned to manage the `cart` slice of the state.

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

3. Exporting the Store:

    - The configured Redux store is exported using `export default store;`, so it can be used throughout the application to manage state.

    ```
    export default store;
    ```

Note:
- This code is pre-configured in the repository and ready for use.
- Make sure that you save these changes by pushing your code to your GitHub repository

Task 6: Set up the global store

1. Navigate to the `main.jsx` file in the `src` folder.

2. The `Provider` component from the `react-redux` library is already imported. This component enables all components in the application to access the Redux store.

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

3. The Redux store is imported from the `store.js` file. This store holds the application's state, using the reducer defined in the `CartSlice.jsx` file.

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

4. The `App` component is wrapped with the `Provider` component, with the Redux store passed as a prop. This allows all components in the app to access and interact with the global state managed by Redux.

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

5. **Deploy using GitHub Pages:** Instructions to deploy your application using GitHub pages can be found in the Task 5 of the lab Setting up the GitHub Environment, covered earlier in this course.

Note:
- This code is pre-configured in the repository and ready for use.
- Make sure that you save these changes by pushing your code to your GitHub repository

Project Completion

Congratulations!

You completed this project! Great work.

If you have not already done so, deploy your application. You can deploy it using GitHub Pages or your own hosting site. Make a note of the URL to the application, you will need that to submit your project for peer review.

Deployment of your react application is important as your peer will review your work based on this.

Summary

- You created function React components, composed, and nested them.
- You implemented React Hooks, specifically the `useState` and `useEffect` hooks to manage component-level states.
- You integrated Redux within your React application and applied Redux concepts like actions, reducers, and the store.
- You dynamically rendered data fetched from an array of objects onto the UI. You mapped over arrays to generate lists of components.
- You handled user events such as button selection and triggered corresponding actions to add and remove items in the cart.

Author

Richa Arora