Phase 1: Setting up the Foundation

Step 1: Project Initialization

1. **Install Node.js and npm:** Ensure you have Node.js and npm (Node Package Manager) installed on your system. You can download them from https://nodejs.org/.
2. **Open your terminal and create a new React project using Create React App:**

```
npx create-react-app my-ecommerce-app
cd my-ecommerce-app
```

Replace "my-ecommerce-app" with your preferred project name.

3. **Start the development server:**

```
npm start
```

This command starts the development server and typically opens your default web browser, displaying a basic React app.

4. **Explore the Project Structure:** Familiarize yourself with the generated files and folders. Pay close attention to:
    - **`public/index.html`** : The single HTML page of your React app. React will inject its components into the `root` element within this file.
    - **`src/index.js`** : The entry point of your React application, where you connect your root component to the `index.html` file.
    - **`src/App.js`** : The main component of your application where we will write the core logic for our e-commerce website.
    - **`src/index.css`** : Global CSS styles.
    - **`src/App.css`** : CSS styles specific to the `App` component.

Step 2: Crafting the Header

1. **Create a Header Component:** In the `src` folder, create a new file named `Header.js` .
2. **Build the Basic Structure:** Let's add a simple header with a logo and placeholder navigation links.

```
// src/Header.js
import React from "react";
import "./Header.css";

function Header() {
    return (
        <div className="header">
            <div className="header__logo">
                {/* Add a logo here */}
                <h2 className="header__logoTitle">eShop</h2>
            </div>
            <div className="header__nav">
                {/* Add navigation links here */}
            </div>
```

```
        </div>
    );
}

export default Header;
```

3. **Add CSS Styling:** Create a file named `Header.css` in the `src` folder. Paste the following CSS to give your header a basic style:

```
// src/Header.css
.header {
    height: 60px;
    display: flex;
    align-items: center;
    background-color: #131921;
    position: sticky;
    top: 0;
    z-index: 100;
}

.header__logo {
    display: flex;
    align-items: center;
    color: #ff9f00;
    margin: 0 25px;
}

.header__logoTitle {
    text-decoration: none;
    border: 0;
    color: white;
}

.header__nav {
    display: flex;
    justify-content: space-evenly;
}
```

Step 3: Constructing the Home Page

1. **Create a Home Component:** Similar to Step 2, create a file named `Home.js` inside the `src` folder.

2. **Set up the Basic Structure:** Inside `Home.js`, create a simple layout for your homepage:

```
// src/Home.js
import React from "react";
import "./Home.css";

function Home() {
    return (
        <div className="home">
```

```
            {/* Add home page content here */}
        </div>
    );
}

export default Home;
```

3. **Include Components in `App.js`:** Import the newly created `Header` and `Home` components into your main `App.js` file:

```
// src/App.js
import './App.css';
import Header from './Header';
import Home from './Home';

function App() {
    return (
        <div className="App">
            <Header />
            <Home />
        </div>
    );
}

export default App;
```

4. **Add Basic Styling:** Create `Home.css` in the `src` folder to add styling to your home page:

```
// src/Home.css
.home {
  display: flex;
  justify-content: center;
  margin-left: auto;
  margin-right: auto;
  max-width: 1500px;
}
```

At this point, you should have a basic React project set up with a header and a home page. Remember to stop and restart your development server whenever you make changes to see them reflected in your browser.

Phase 2: Dynamic Content and Routing

Step 4: Introducing Product Data

1. **Create a Product Data File:** In your `src` folder, create a new JavaScript file named `data.js`.
2. **Populate with Sample Data:** Within `data.js`, define an array of product objects. Each object should contain the following properties:
    - `id`: A unique identifier for the product (you can use numbers or strings).
    - `title`: The name of the product.

- `price` : The price of the product (you can use numbers).
- `rating` : A rating for the product (out of 5, using numbers).
- `image` : A URL pointing to an image of the product.

```
// src/data.js
const products = [
    {
        id: "12321341",
        title: "Bennett Mystic 15.6 inch Laptop Shoulder Messenger Sling Office Bag,
Water Repellent Fabric for Men and Women (Blue)",
        price: 11.96,
        rating: 5,
        image: "https://images-na.ssl-images-
amazon.com/images/I/71mEsHyzSCL._SL1000_.jpg"
    },
    {
        id: "49538094",
        title: "IFB 30 L Convection Microwave Oven (30BRC2, Black, With Starter
Kit)",
        price: 239.0,
        rating: 4,
        image: "https://images-na.ssl-images-
amazon.com/images/I/81D8pNFmWzL._SL1500_.jpg"
    },
    // Add more products here...
];

export default products;
```

Step 5: Implementing Product Display

1. **Modify the `Product.js` Component:** Open the existing `Product.js` file in your `src` folder.
2. **Replace Placeholders with Props:** Inside the `Product` function, replace the placeholders you created earlier with dynamic content retrieved from props. You'll receive the product data as an object named `props` .

```
// src/Product.js
import React from "react";
import "./Product.css";

function Product(props) {
    return (
        <div className="product">
            <div className="product__info">
                <p>{props.title}</p>
                <p className="product__price">
                    <small>$</small>
                    <strong>{props.price}</strong>
                </p>
                <div className="product__rating">
```

```
                    {Array(props.rating) // Create an array from the rating
value
                    .fill()           // Fill the array with empty values
                    .map((_, i) => ( // Map over the array to render stars
                        <p key={i}>⭐</p> // Use "key" for React to
efficiently update the list
                    ))}
                </div>
            </div>

            <img src={props.image} alt={props.title} />

            <button>Add to Basket</button>
        </div>
    );
}

export default Product;
```

Step 6: Utilizing Props for Dynamic Rendering

1. **Import Product Data and the `Product` component:** In your `Home.js` file, import the `products` array from `data.js` and the `Product` component.

```
// src/Home.js
import React from "react";
import "./Home.css";
import Product from "./Product";
import products from "./data"; // Import product data

// ... rest of your Home component
```

2. **Map Product Data to Components:** In your `Home.js` file, map over the `products` array to render a `Product` component for each product object. Pass the product data as props to each `Product` component.

```
// src/Home.js
// ... (import statements from previous step)

function Home() {
    return (
        <div className="home">
            <div className="home__container">
                {/* ... your existing image banner ... */}

                <div className="home__row">
                    {products.map(product => (
                        <Product
                            key={product.id} // Assign a unique key to each
product
                            id={product.id}
```

```
                            title={product.title}
                            price={product.price}
                            rating={product.rating}
                            image={product.image}
                        />
                    ))}
                </div>

                {/* You can add more home__row divs and map products as needed
*/}
            </div>
        </div>
    );
}

// ... (export statement)
```

Step 7: Integrating React Router

1. **Install React Router:** In your terminal, install the React Router DOM package.

```
npm install react-router-dom
```

2. **Set up the Router:** Wrap your main `App` component in `BrowserRouter` from `react-router-dom`.

```
// src/App.js
import React from 'react';
import { BrowserRouter as Router, Switch, Route } from "react-router-dom";
import './App.css';
import Checkout from './Checkout';
import Header from './Header';
import Home from './Home';

function App() {
return (
    <Router>
    <div className="App">
        {/* Header will remain consistent across pages */}
        <Header />

        <Switch>
        {/* Use exact for the home route to prevent it from matching all
routes */}
        <Route exact path="/">
            <Home />
        </Route>

        <Route path="/checkout">
            <Checkout />
        </Route>
```

```
        {/* Other routes can be added here later */}
        </Switch>
    </div>
    </Router>
);
}

export default App;
```

3. **Create Links:** Use `Link` from `react-router-dom` to navigate between the Home and Checkout pages.
    - In `Header.js`, wrap the shopping cart icon with a `Link` to `/checkout`.
    - You can create additional links in your application as needed.

```
// src/Header.js
import React from "react";
import "./Header.css";
import ShoppingBasketIcon from '@material-ui/icons/ShoppingBasket';
import StorefrontIcon from '@material-ui/icons/Storefront';
import SearchIcon from '@material-ui/icons/Search';
import { Link } from "react-router-dom"; // Import Link

function Header() {
    // ... your existing header content ...

    return (
        <div className="header">
            {/* ... your existing logo ... */}

            <div className="header__nav">
                {/* ... your existing navigation links ... */}

                <Link to="/checkout" style={{ textDecoration: "none" }}> {/*
Wrap with Link */}
                    <div className="nav__itemBasket">
                        <ShoppingBasketIcon/>
                        <span className="nav__itemLineTwo
nav__basketCount">0</span>
                    </div>
                </Link> {/* Close the Link tag */}
            </div>
        </div>
    );
}
```

At this point, your application should display dynamically generated products on the home page and allow navigation between the Home page and a basic Checkout page using React Router.

Phase 3: Building E-commerce Functionality

Step 8: Creating the Checkout Page

1. **Enhance the Checkout Page Structure:** Open your `Checkout.js` file. Let's add a more informative structure with sections for displaying checkout products and order summary.

```
// src/Checkout.js
import React from "react";
import "./Checkout.css";
import CheckoutProduct from "./CheckoutProduct";
import Subtotal from "./Subtotal.js";
import { useStateValue } from "./StateProvider";

function Checkout() {
  const [{ basket }, dispatch] = useStateValue();

  return (
    <div className="checkout">
      <div className="checkout__left">
        <img
          src="https://images-na.ssl-images-
amazon.com/images/G/02/UK_CCMP/TM/OCC_Amazon1._CB423492668_.jpg"
          alt=""
          className="checkout__ad"
        />
        <div>
          <h2 className="checkout__title">Your Shopping Basket</h2>
          {/*  We will map through the basket and render CheckoutProduct here */}
        </div>
      </div>

      <div className="checkout__right">
          {/* We will render our Subtotal component here */}
      </div>
    </div>
  );
}

export default Checkout;
```

Step 9: State Management with Context API

1. **Create a Context and Reducer:** Generate two files, `StateProvider.js` and `reducer.js`, within your `src` directory.
    - **`StateProvider.js`**:

```
// src/StateProvider.js
import React, { createContext, useContext, useReducer } from "react";
```

```
// This will be our data layer
export const StateContext = createContext();

// Build a Provider
export const StateProvider = ({ reducer, initialState, children }) => (
  <StateContext.Provider value={useReducer(reducer, initialState)}>
    {children}
  </StateContext.Provider>
);

// This is how we will pull information from the data layer
export const useStateValue = () => useContext(StateContext);
```

- **`reducer.js`**:

```
// src/reducer.js
export const initialState = {
    basket: [],
};

const reducer = (state, action) => {
    // We listen to different actions being dispatched
    console.log(action);

    switch(action.type) {
        case "ADD_TO_BASKET":
            return {
                // ...state means that we return the state as it originally
was
                ...state,
                // Except, we modify the basket by adding whatever product
(action.item)
                // was passed to this reducer with ADD_TO_BASKET action
                basket: [...state.basket, action.item],
            };

        default:
            return state;
    }
};

export default reducer;
```

2. **Wrap your Application with the Provider:** Open your main `index.js` file. Wrap your `<App />` component with the `StateProvider`.

```
// src/index.js
// Existing imports

import reducer, { initialState } from "./reducer";
```

```
import { StateProvider } from "./StateProvider";


ReactDOM.render(
  <React.StrictMode>
  {/* Here we provide the reducer to the root of the app  */}
    <StateProvider initialState={initialState} reducer={reducer}>
      <App />
    </StateProvider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

Step 10: Adding Items to the Cart

1. **Implement Add to Basket Functionality:** In `Product.js`, implement the `addToBasket` function to dispatch the "ADD_TO_BASKET" action when the "Add to Basket" button is clicked.

```
// src/Product.js
import React from "react";
import "./Product.css";
import { useStateValue } from "./StateProvider"; // Import useStateValue

function Product({ id, title, image, price, rating }) {
  // Access dispatch to update the global state
  const [{ basket }, dispatch] = useStateValue();

  const addToBasket = () => {
    // Dispatch the action to add the product to the basket
    dispatch({
      type: "ADD_TO_BASKET",
      item: {
        id: id,
        title: title,
        image: image,
        price: price,
        rating: rating,
      },
    });
  };

  return (
    <div className="product">
        {/* ... rest of your Product component... */}
        <button onClick={addToBasket}>Add to Basket</button>
    </div>
  );
}
```

2. **Display Products in Checkout:** Update `Checkout.js` to map through the `basket` array and render `CheckoutProduct` components for each item.

```
// src/Checkout.js
// ... existing imports ...

function Checkout() {
    const [{ basket }, dispatch] = useStateValue();

    return (
        <div className="checkout">
            {/* ... (rest of your Checkout component) ... */}

            <div>
                <h2 className="checkout__title">Your Shopping Basket</h2>
                {basket.map(item => (
                    <CheckoutProduct
                        id={item.id}
                        title={item.title}
                        image={item.image}
                        price={item.price}
                        rating={item.rating}
                    />
                ))}
            </div>

            {/* ... (rest of your Checkout component) ... */}
        </div>
    );
}

// ... (export statement) ...
```

Step 11: Removing Items from the Cart

1. **Implement Remove from Basket Functionality:** In `CheckoutProduct.js`, create a `removeFromBasket` function to dispatch the "REMOVE_FROM_BASKET" action.
2. **Add Remove Button:** Add a button to `CheckoutProduct` that, when clicked, calls `removeFromBasket`.
3. **Update Reducer:** Modify the reducer in `reducer.js` to handle the "REMOVE_FROM_BASKET" action and update the basket state accordingly.

```
// src/reducer.js
// ... (initialState and other code from previous steps) ...

const reducer = (state, action) => {
  // ... (other cases) ...

  case "REMOVE_FROM_BASKET":
    // Logic to remove item from basket
    // Find the index of the item to remove
    const index = state.basket.findIndex(
      (basketItem) => basketItem.id === action.id
    );
```

```
    // Create a copy of the basket
    let newBasket = [...state.basket];

    if (index >= 0) {
      // If item exists in basket, remove it
      newBasket.splice(index, 1);
    } else {
      console.warn(
        `Can't remove product (id: ${action.id}) as its not in basket!`
      );
    }

    return {
      ...state,
      basket: newBasket,
    };

  default:
    return state;
  }
};

// ... (export default reducer) ...
```

```
// src/CheckoutProduct.js
import React from 'react';
import "./CheckoutProduct.css";
import { useStateValue } from './StateProvider';

function CheckoutProduct({ id, image, title, price, rating }) {
  const [{ basket }, dispatch] = useStateValue();

  const removeFromBasket = () => {
    // dispatch the action to remove item from basket
    dispatch({
      type: 'REMOVE_FROM_BASKET',
      id: id, // Pass the id of the product to remove
    });
  };

  return (
    <div className="checkoutProduct">
      <img className="checkoutProduct__image" src={image} alt="" />

      <div className="checkoutProduct__info">
        <p className="checkoutProduct__title">{title}</p>
        <p className="checkoutProduct__price">
          <small>$</small>
          <strong>{price}</strong>
        </p>
        <div className="checkoutProduct__rating">
```

```
          {Array(rating)
            .fill()
            .map((_, i) => (
              <p>⭐</p>
            ))}
        </div>
        <button onClick={removeFromBasket}>Remove from Basket</button>
      </div>
    </div>
  );
}

export default CheckoutProduct;
```

By the end of Phase 3, users should be able to add and remove products from the cart, with these actions reflected in the global state and the Checkout page. This setup provides a solid foundation for expanding e-commerce features in the following phases.

Phase 4: User Authentication

Step 12: Setting up Firebase Authentication

1. **Create a Firebase Project:**
   - Go to https://firebase.google.com/ and create a new Firebase project.
   - Follow the instructions to set up your project. You'll likely need to create a new Firebase web app.
2. **Enable Email/Password Authentication:**
   - In your Firebase project console, navigate to "Authentication".
   - Select "Get Started" and enable the "Email/Password" provider.
3. **Install Firebase in your React Project:**

```
npm install firebase
```

1. **Create a Firebase Configuration File:**
   - Create a file named `firebase.js` inside your `src` folder.
   - Copy your Firebase web app configuration from the Firebase console (it should look like the code snippet below) and paste it into `firebase.js`.
   - Initialize Firebase and export the necessary objects (like `auth` for authentication and `db` for Firestore if you plan to use it).

```
// src/firebase.js
import firebase from "firebase";

const firebaseConfig = {
    // Your Firebase project configuration goes here
};

const firebaseApp = firebase.initializeApp(firebaseConfig);
```

```
const db = firebaseApp.firestore(); // Export Firestore if you'll use it
const auth = firebase.auth(); // Export auth for user authentication

export { db, auth }; // Export the objects you need
```

Step 13: Creating a Login Page

1. **Build the Login Component Structure:**
   - Open the `Login.js` file.
   - Craft a simple login form using HTML, including input fields for email and password, and buttons for "Sign In" and "Create Account".

```
// src/Login.js
import React, { useState } from 'react';
import './Login.css'; // Import your CSS file for styling
import { Link, useHistory } from "react-router-dom";
import StorefrontIcon from '@material-ui/icons/Storefront';
import { auth } from "./firebase";

function Login() {
    // ... we will add state variables and functions here later

    return (
        <div className='login'>
            <Link to='/' style={{ textDecoration: "none" }}>
                <div className="login__logo">
                    <StorefrontIcon className="login__logoImage"
fontSize="large" />
                    <h2 className="login__logoTitle">eSHOP</h2>
                </div>
            </Link>

            <div className='login__container'>
                <h1>Sign-in</h1>

                <form>
                    <h5>E-mail</h5>
                    <input type='text'  /> {/* Add onChange handler later */}

                    <h5>Password</h5>
                    <input type='password' /> {/* Add onChange handler later
*/}

                    <button type='submit' className='login__signInButton'
>Sign In</button> {/* Add onClick handler later */}
                </form>

                <p>
                    By signing-in you agree to the eShop Website Conditions of
Use & Sale. Please
                    see our Privacy Notice, our Cookies Notice and our
```

```
Interest-Based Ads Notice.
                </p>

                <button className='login__registerButton'>Create your eShop
Account</button> {/* Add onClick handler later */}
            </div>
        </div>
    );
}

export default Login;
```

2. **Implement Login Functionality:**

- Within `Login.js`, import `useState` to manage the email and password input values.
- Create two state variables: `email` and `password`, initialized as empty strings.
- Attach `onChange` handlers to the input fields to update these state variables whenever the input values change.
- Implement the `signIn` function that will be called when the "Sign In" button is clicked.
- Inside `signIn`, prevent default form submission and use `auth.signInWithEmailAndPassword(email, password)` from Firebase to sign in the user.
- Handle successful login (e.g., redirect to the home page) and potential errors (display an error message).

```
// Inside src/Login.js
// ... (other imports)

function Login() {
    const history = useHistory(); // For redirecting after login
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');

    const signIn = e => {
        e.preventDefault(); // Prevent default form submission

        // Firebase login logic
        auth
            .signInWithEmailAndPassword(email, password)
            .then(auth => {
                // Successfully signed in, redirect to home page
                history.push('/');
            })
            .catch(error => alert(error.message)); // Handle errors
    };

    // ... (rest of your Login component)
}
```

3. **Implement Registration Functionality:**

- Implement the `register` function, which will be called when the "Create Account" button is clicked.
- Inside `register`, use `auth.createUserWithEmailAndPassword(email, password)` to create a new user in Firebase.
- Like with `signIn`, handle success (e.g., redirect to home page) and potential errors (display an error message).

```
// Inside src/Login.js
// ... (other code)

const register = e => {
    e.preventDefault();

    // Firebase registration logic
    auth
        .createUserWithEmailAndPassword(email, password)
        .then((auth) => {
            // Successfully created a new user
            if (auth) {
                history.push('/'); // Redirect to home page
            }
        })
        .catch(error => alert(error.message)); // Handle errors
};

// ... (rest of your Login component)
```

4. **Add the Login Route:**

- In `App.js`, import the `Login` component.
- Add a new route for `/login` within your `Switch` component to render the `Login` page when the URL matches.

```
// src/App.js
// ... other imports

function App() {
    // ...

    return (
        <div className="App">
            <Router>
                <Switch>
                    {/* ... other routes ... */}
                    <Route path="/login">
                        <Login />
                    </Route>
                    {/* ... other routes ... */}
                </Switch>
            </Router>
        </div>
```

```
    );
}
```

After completing Phase 4, users should be able to register, log in, and be redirected accordingly. Remember that this phase focuses on setting up basic user authentication. For a production-ready app, you'll need to implement more robust error handling, input validation, and potentially password reset functionality.

By providing this level of detail and code snippets, students can focus on understanding the core concepts of React and Firebase authentication while also gaining practical experience in building a real-world feature. They are encouraged to consult external documentation and explore further to solidify their understanding.

Phase 5: Final Touches (Detailed)

Step 14: Deployment

This step will guide you through preparing and deploying your React e-commerce application using Netlify.

14.1 Preparing for Deployment

1. **Build Your Project:**
    - In your terminal, navigate to your project's root directory and run the following command:

```
npm run build
```

    - This command will create a `build` folder in your project directory containing all the optimized and bundled files ready for deployment.

14.2 Deploying with Netlify (Recommended)

1. **Sign up for Netlify:**
    - Head over to https://www.netlify.com/ and sign up for a free account.
2. **Drag and Drop:**
    - Once logged in, Netlify provides a simple drag-and-drop interface. Drag your project's `build` folder directly into the designated area.
3. **Configuration (Automatic):**
    - Netlify will automatically detect that you're deploying a React app and configure the settings accordingly.
4. **Deploy!**
    - Review the settings and hit the "Deploy" button. Netlify will handle the rest!
5. **Access Your Site:**
    - After a short period, your site will be live. Netlify will provide you with a unique URL to access your deployed e-commerce application.

Congratulations! You've successfully built and deployed a basic e-commerce website using React!

Further Enhancements:

1. **Styling and Responsiveness:** Improve the visual appearance and user experience of your website by applying more sophisticated CSS styles and ensuring responsiveness across different screen sizes.
2. **Product Filtering and Sorting:** Implement functionality to filter products based on categories, price ranges, or other criteria. Add sorting options to help users find the products they're looking for more

easily.

3. **Payment Integration:** For a fully functional e-commerce website, integrate a payment gateway (e.g., Stripe, PayPal) to enable secure online transactions.
4. **Order Management:** Develop a backend system (or integrate with existing e-commerce platforms) to manage orders, track inventory, and handle customer data.

Remember, this is just the beginning! The world of web development offers endless possibilities for learning and building amazing things. Continue to explore, experiment, and enhance your skills to create even more complex and feature-rich applications. Happy coding!