Week 13: Assignment Work - GadgetShop (Frontend)

Day 1: Project Architecture and Component Planning

Theoretical Overview

The development of the **GadgetShop** frontend began with a modular architectural plan. Following the React Component Model, the UI was decomposed into a hierarchy of independent, reusable pieces. This approach ensures that logic is encapsulated and the codebase remains maintainable as the shop scales.

The Component Tree

- **Root Component (App.js)**: Serves as the main container and manages the overall state of the application.
- **Navigation Component (NavBar.js)**: Contains links for "Home," "Shop," and "Cart," utilizing react-router for SPA navigation.
- **Product Gallery (ProductList.js)**: A container component responsible for fetching data and mapping it into individual cards.
- **Product Item (ProductCard.js)**: A functional component that receives product details (name, price, image) via props.

Day 2: Dynamic Data Rendering and List Keys

Mapping the Product Catalog

Using the logic stored in the **"React New"** and **"GadgetShop"** folders, we implemented the rendering of the product list.

- **The .map() Function**: JavaScript's .map() method was used to iterate through the array of gadgets fetched from the API.
- **Unique Keys**: To optimize the Virtual DOM's diffing process, each product card was assigned a unique key (the product ID), ensuring only changed elements re-render.
- **Conditional Rendering**: Logic was added to display a "Loading..." spinner while the data is being fetched and a "No Products Found" message if the array is empty.

Day 3: State Management for the Shopping Cart

Lifting State Up

The most critical part of the GadgetShop is the cart functionality. Since both the ProductCard (to add items) and the NavBar (to show the cart count) need access to the same data, we implemented "Lifting State Up".

- **State Placement**: The cartItems state was moved to the App component to serve as the "Single Source of Truth".
- **The useState Hook**: Initialized as an empty array: const [cart, setCart] = useState([]).
- **Immutability**: When adding items, we used the Spread Operator to create a new array rather than pushing to the existing one, adhering to React's immutability principles.

Day 4: Navigation and Routing with React Router

Single Page Application (SPA) Flow

We integrated react-router to allow users to navigate between the product list and a detailed product view without a page refresh.

- **The <Routes> Container**: Defines paths like / for the shop and /product/:id for details.
- **Retrieving Parameters**: The useParams hook was utilized in the detailed view to identify which product to fetch based on the URL ID.

- **Programmatic Navigation**: After a user clicks "Checkout," the useNavigate hook is used to send them to a "Success" page.

Day 5: Styling with CSS and Responsive Design

UI/UX Implementation

Drawing from the foundations in **Week 2**, the GadgetShop was styled using modern CSS techniques.

- **Flexbox Layouts**: The product gallery was built using display: flex with flex-wrap: wrap to ensure a responsive grid across mobile and desktop.
- **CSS Classes vs. Direct Styles**: For maintainability, external CSS classes were preferred over inline styles.
- **Hover Effects**: CSS transitions were added to buttons and cards to provide immediate visual feedback to the user.