Day 4 focuses on creating the frontend for a marketplace using dynamic components that fetch and display data from APIs or a CMS like Sanity. The primary goal is to ensure the marketplace is modular, scalable, and responsive, making it suitable for real-world use.

Key Objectives

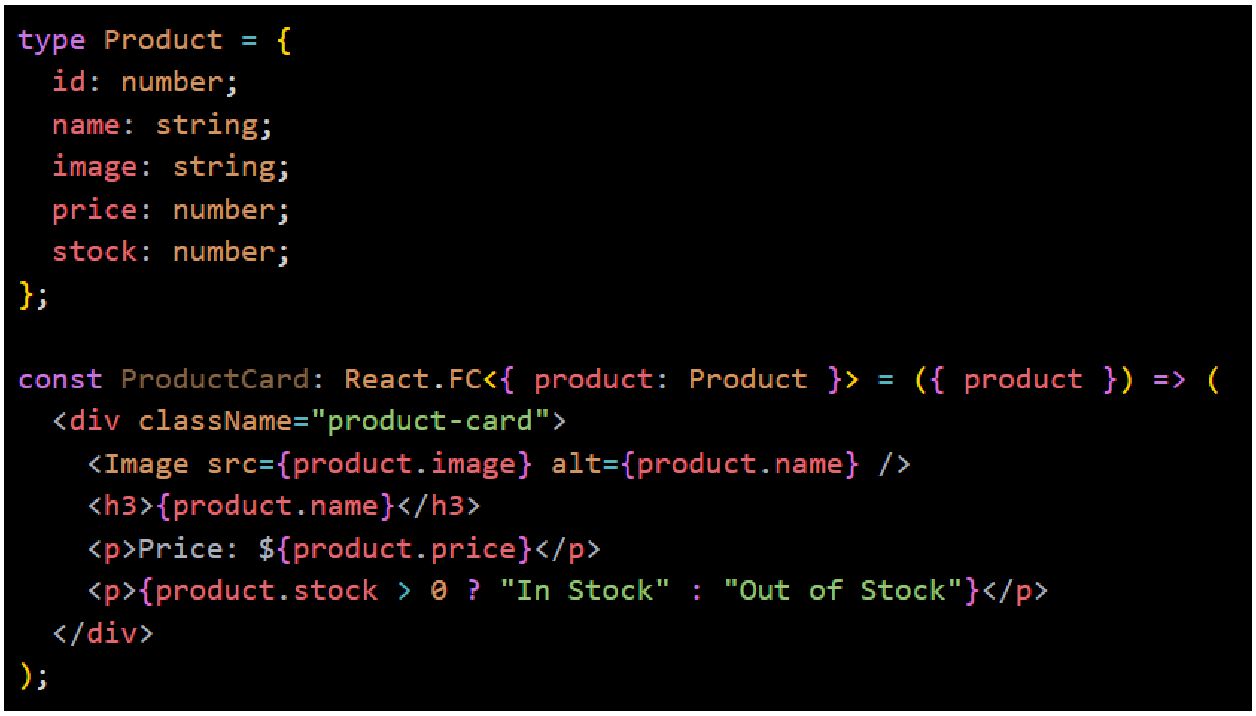
1. Build dynamic components to display data (e.g., products, categories, user profiles).
2. Create reusable and modular components for scalability.
3. Implement state management for tracking user interactions (e.g., cart, filters).
4. Apply responsive design principles for an optimal user experience on all devices.
5. Prepare a professional and functional frontend for a marketplace.

1. Key Components to Build

1.1 Product Listing Component

- **Purpose**: Display products in a grid layout dynamically fetched from an API or CMS.
- **Features**: Show product name, price, stock status, and an image.

Example Code:



```
type Product = {
  id: number;
  name: string;
  image: string;
  price: number;
  stock: number;
};

const ProductCard: React.FC<{ product: Product }> = ({ product }) => (
  <div className="product-card">
    <Image src={product.image} alt={product.name} />
    <h3>{product.name}</h3>
    <p>Price: ${product.price}</p>
    <p>{product.stock > 0 ? "In Stock" : "Out of Stock"}</p>
  </div>
);
```

1.2 Product Detail Component

- **Purpose**: Show detailed information about a selected product.
- **Features**: Display description, price, available sizes/colors, and related products.

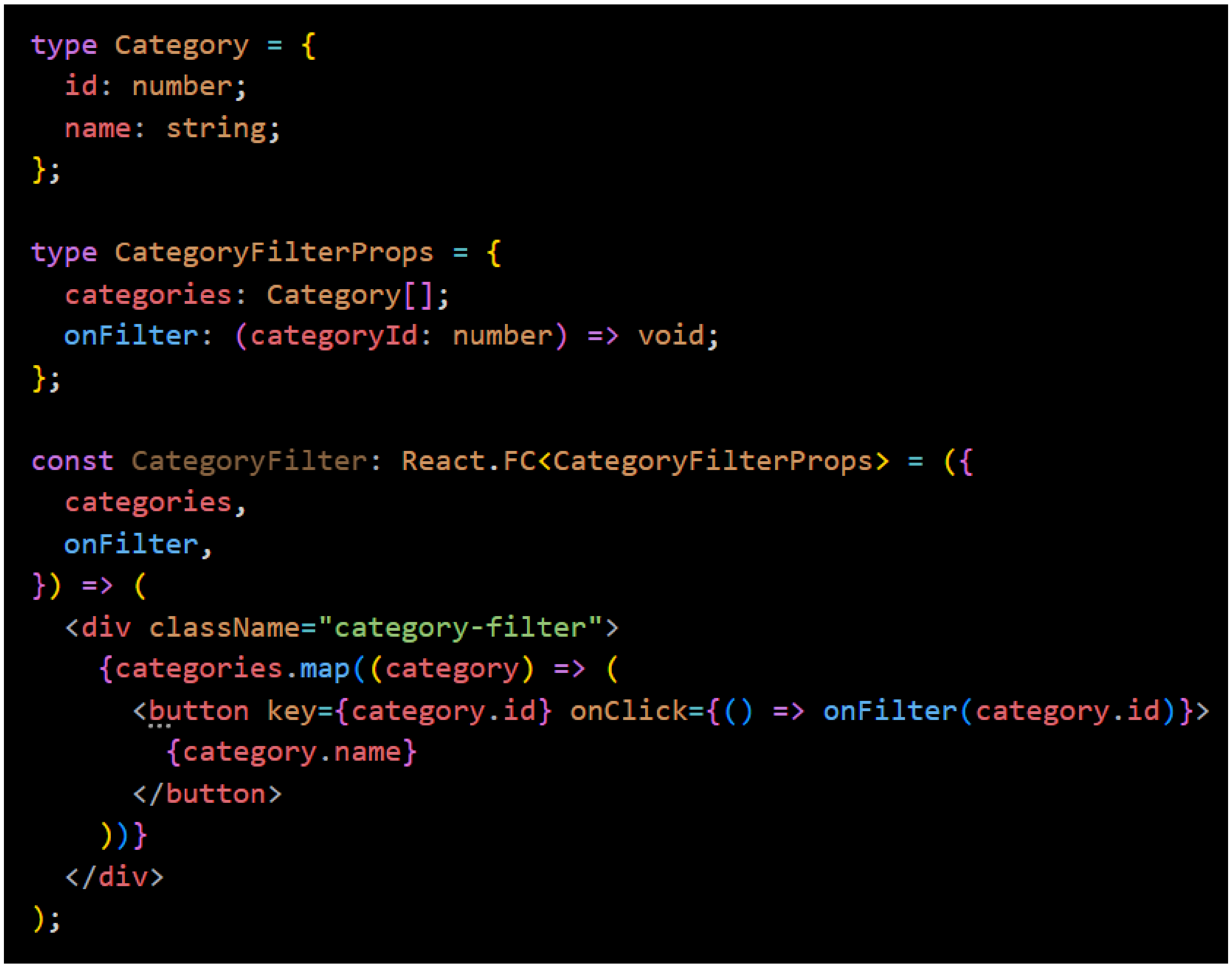
Example Code:

```
type ProductDetailProps = {
  product: {
    id: number;
    name: string;
    description: string;
    price: number;
    sizes: string[];
  };
};

const ProductDetail: React.FC<ProductDetailProps> = ({ product }) => (
  <div className="product-detail">
    <h1>{product.name}</h1>
    <p>{product.description}</p>
    <p>Price: ${product.price}</p>
    <p>Available Sizes: {product.sizes.join(", ")}</p>
  </div>
);

export async function getServerSideProps({
  params,
}: {
  params: { id: string };
}) {
  const res = await fetch(`/api/products/${params.id}`);
  const product = await res.json();
  return { props: { product } };
}
```

1.3 Category Component

- **Purpose**: Display product categories and enable filtering based on user selection.
- **Features**: Fetch categories dynamically, filter products by category.

Example Code:

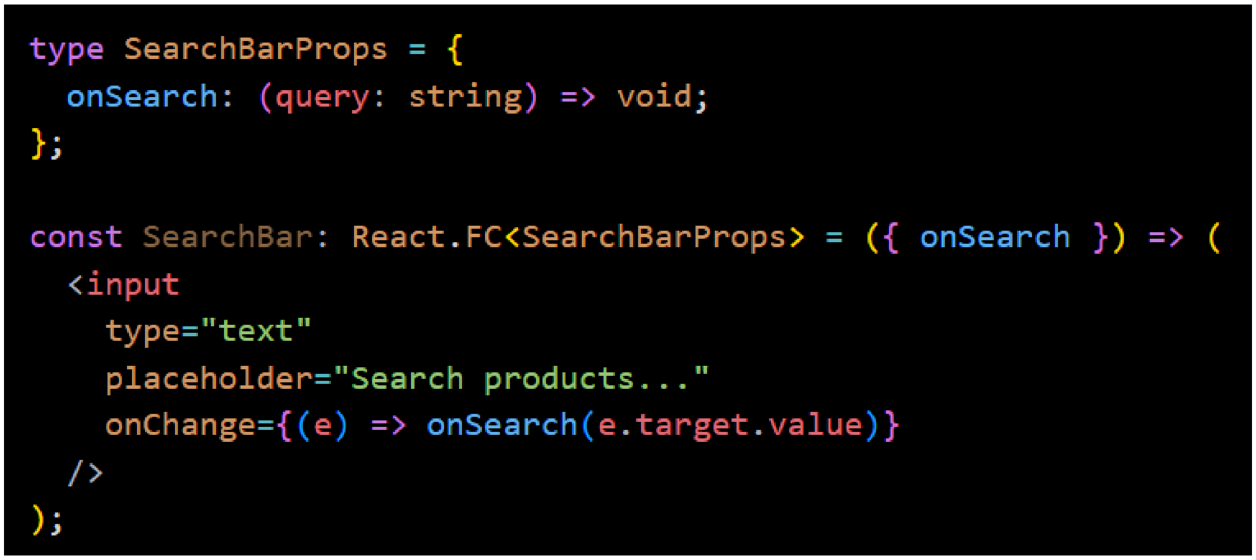


```
type Category = {
  id: number;
  name: string;
};

type CategoryFilterProps = {
  categories: Category[];
  onFilter: (categoryId: number) => void;
};

const CategoryFilter: React.FC<CategoryFilterProps> = ({
  categories,
  onFilter,
}) => (
  <div className="category-filter">
    {categories.map((category) => (
      <button key={category.id} onClick={() => onFilter(category.id)}>
        {category.name}
      </button>
    ))}
  </div>
);
```

1.4 Search Bar

- **Purpose**: Allow users to search products by name or tags.
- **Features**: Filter the product listing dynamically as the user types.

Example Code:

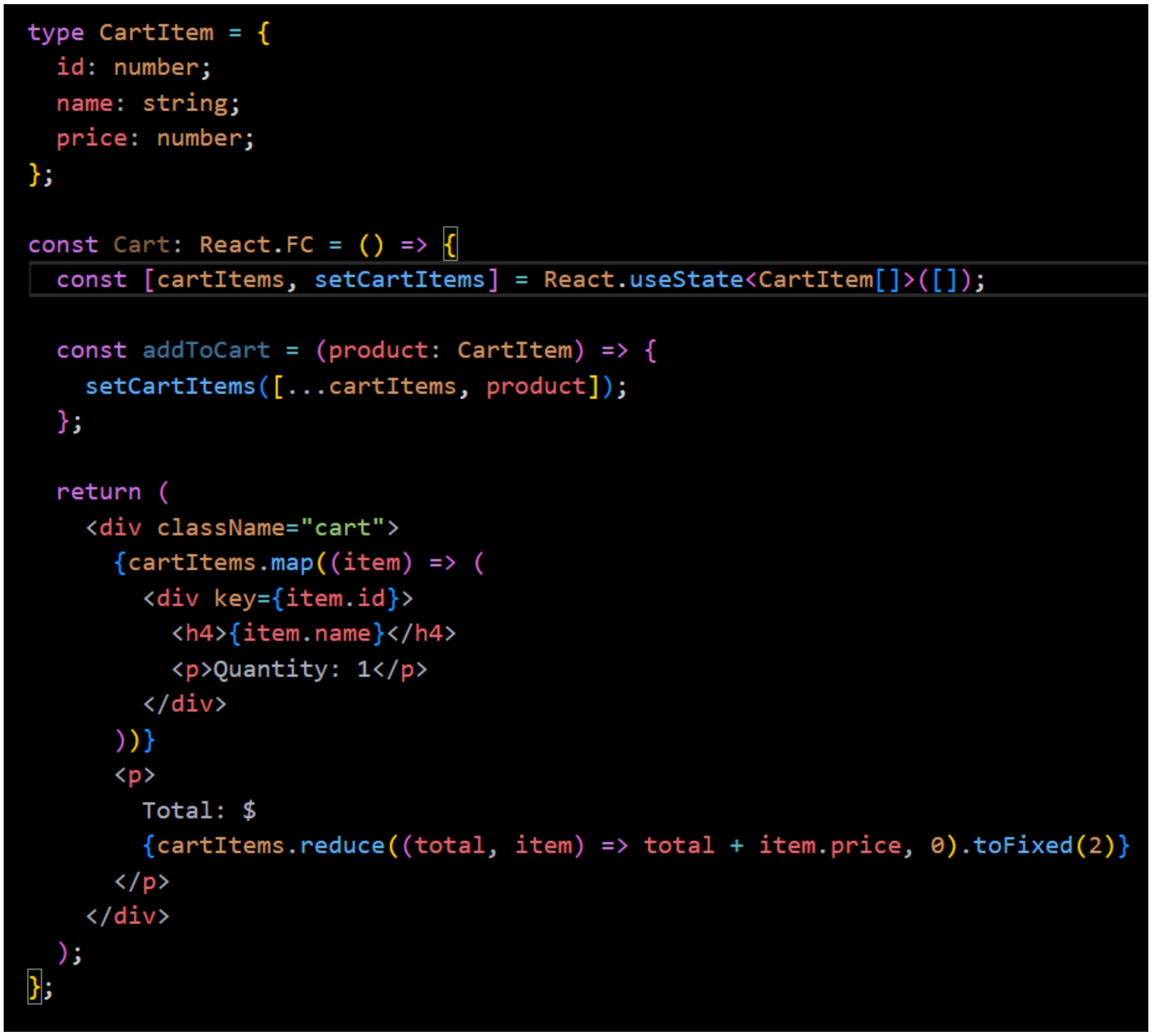


```tsx
type SearchBarProps = {
  onSearch: (query: string) => void;
};

const SearchBar: React.FC<SearchBarProps> = ({ onSearch }) => (
  <input
    type="text"
    placeholder="Search products..."
    onChange={(e) => onSearch(e.target.value)}
  />
);
```

1.5 Cart Component

- **Purpose**: Display added items, quantity, and total price.
- **Features**: Track cart items using state management.

Example Code:

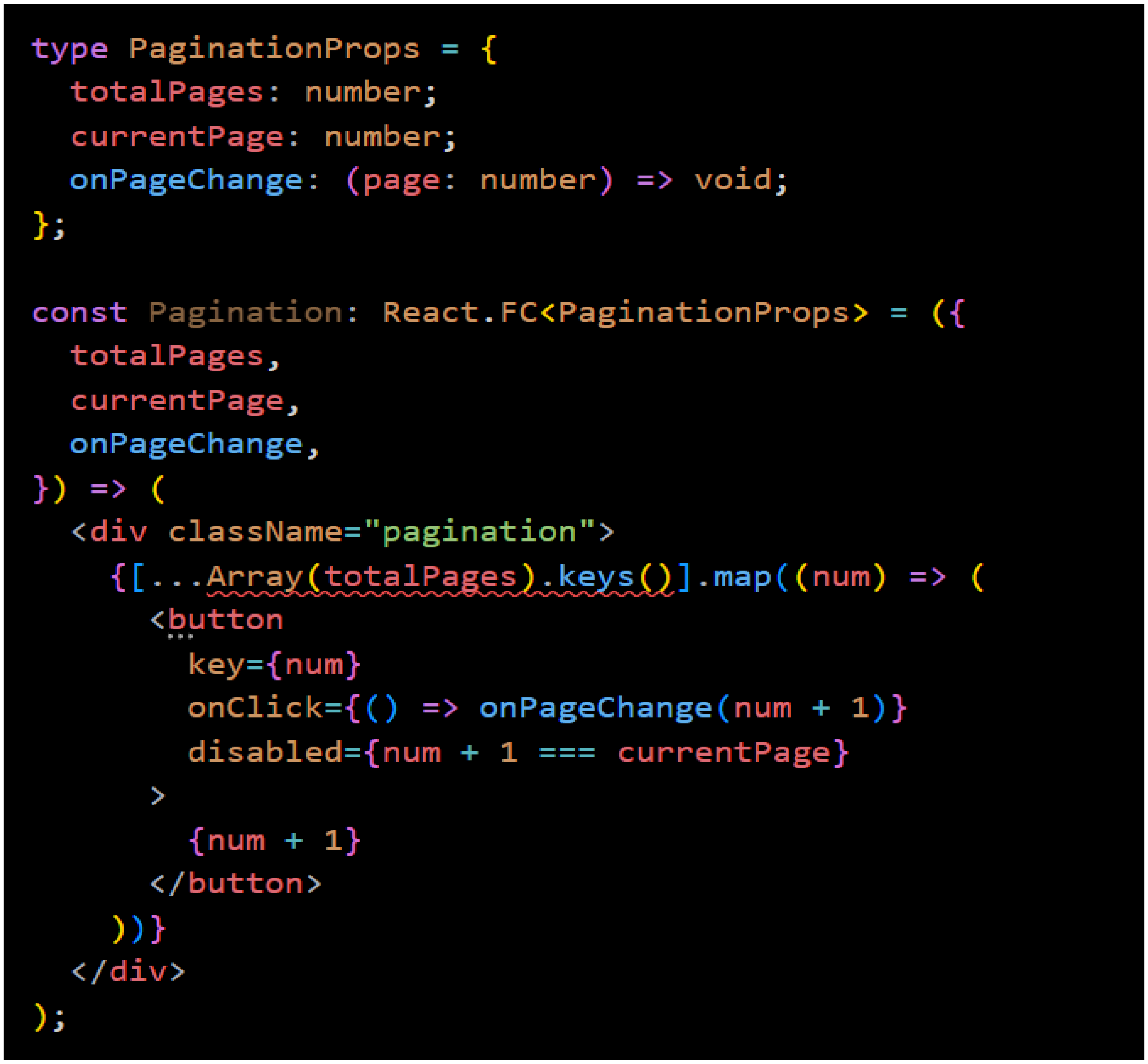


```tsx
type CartItem = {
  id: number;
  name: string;
  price: number;
};

const Cart: React.FC = () => {
  const [cartItems, setCartItems] = React.useState<CartItem[]>([]);

  const addToCart = (product: CartItem) => {
    setCartItems([...cartItems, product]);
  };

  return (
    <div className="cart">
      {cartItems.map((item) => (
        <div key={item.id}>
          <h4>{item.name}</h4>
          <p>Quantity: 1</p>
        </div>
      ))}
      <p>
        Total: $
        {cartItems.reduce((total, item) => total + item.price, 0).toFixed(2)}
      </p>
    </div>
  );
};
```

1.6 Pagination Component

- **Purpose**: Break down product listings into multiple pages.
- **Features**: Implement "Previous" and "Next" buttons or page numbers.

Example Code:

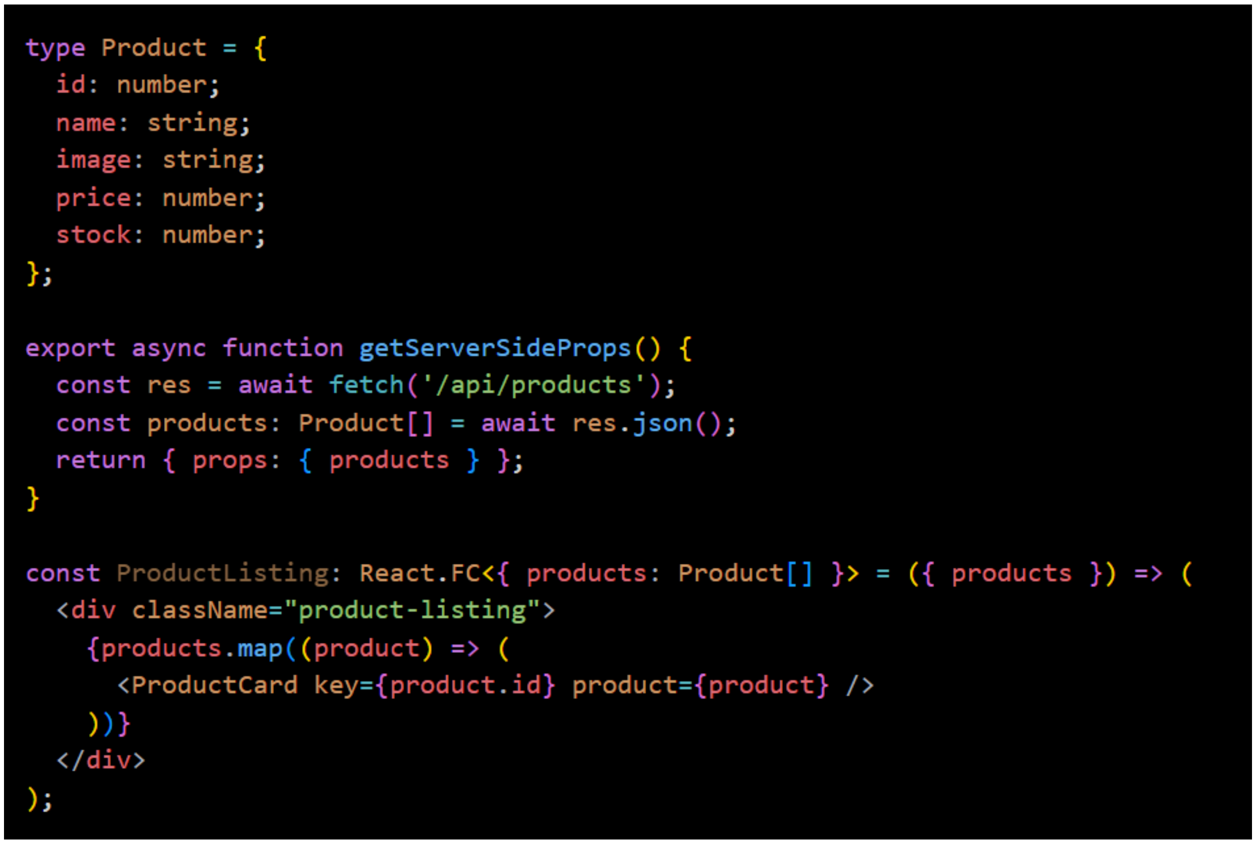


```
type PaginationProps = {
  totalPages: number;
  currentPage: number;
  onPageChange: (page: number) => void;
};

const Pagination: React.FC<PaginationProps> = ({
  totalPages,
  currentPage,
  onPageChange,
}) => (
  <div className="pagination">
    {[...Array(totalPages).keys()].map((num) => (
      <button
        key={num}
        onClick={() => onPageChange(num + 1)}
        disabled={num + 1 === currentPage}
      >
        {num + 1}
      </button>
    ))}
  </div>
);
```

Step 3: Fetch Data Dynamically

- Use `getServerSideProps` or `getStaticProps` to fetch data for each page.

Example:



```
type Product = {
  id: number;
  name: string;
  image: string;
  price: number;
  stock: number;
};

export async function getServerSideProps() {
  const res = await fetch('/api/products');
  const products: Product[] = await res.json();
  return { props: { products } };
}

const ProductListing: React.FC<{ products: Product[] }> = ({ products }) => (
  <div className="product-listing">
    {products.map((product) => (
      <ProductCard key={product.id} product={product} />
    ))}
  </div>
);
```