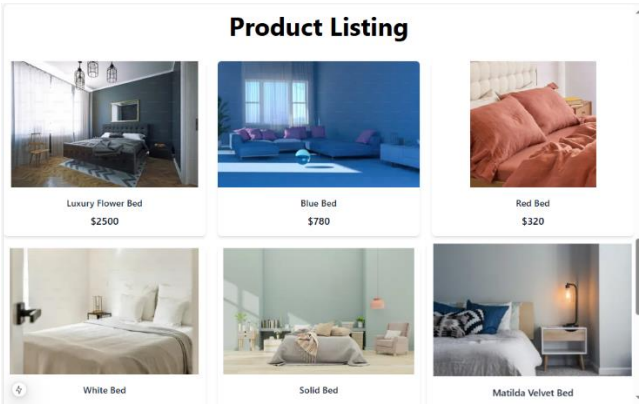
Day 4 - Dynamic Frontend Components - Furniture Store

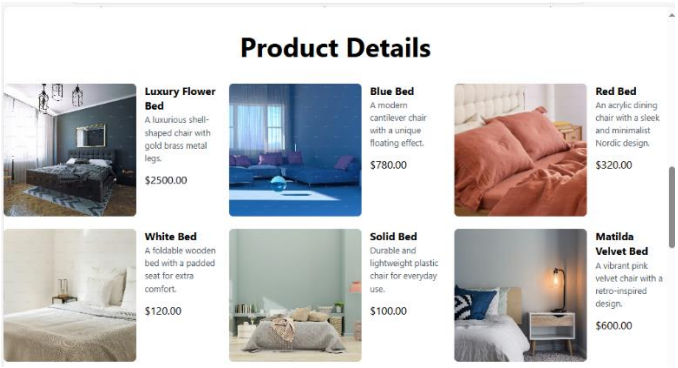
1. Functional Deliverables

Screenshots or Screen Recordings

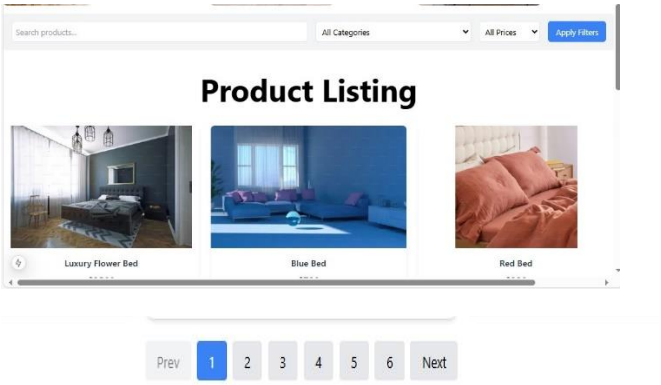
- **Product Listing Page**: Displays dynamic data from Sanity CMS.



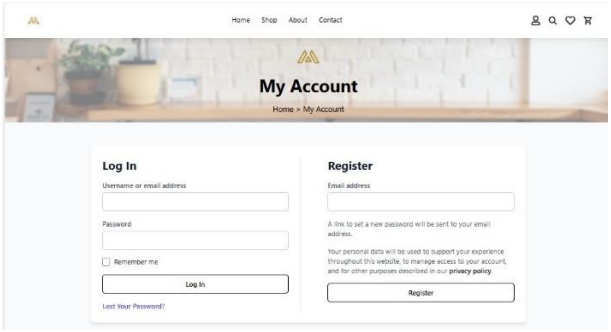
- **Product Detail Page**: Routes dynamically and renders product information correctly.



- **Category Filters, Search Bar, and Pagination**: Working as expected.



- **Additional Features** (if implemented): Related products, user profile, cart functionality, etc.



---

2. Code Deliverables

Key Component Code Snippets

ProductCard.tsx (Reusable Product Card Component)

```
import Image from 'next/image';
import Link from 'next/link';
import { Product } from '@/types';

interface ProductCardProps {
  product: Product;
}

const ProductCard: React.FC<ProductCardProps> = ({ product }) => {
  return (
    <div className="border p-4 rounded-lg shadow-md">
      <Link href={`/product/${product._id}`}>
        <Image src={product.image} alt={product.name} width={200} height={200} />
        <h3 className="font-bold mt-2">{product.name}</h3>
        <p className="text-gray-600">${product.price}</p>
```

```
    </Link>
  </div>
 );
};

export default ProductCard;
```

ProductList.tsx (Rendering Products)

```
import ProductCard from './ProductCard';
import { Product } from '@/types';

interface ProductListProps {
 products: Product[];
}

const Product List: React.FC<ProductListProps> = ({ products }) => {
 return (
  <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
    {products. Map((product) => (
      <ProductCard key={product._id} product={product} />
    ))}
  </div>
 );
};

export default ProductList;
```

---

3. Documentation

Technical Summary

Steps Taken to Build & Integrate Components

1. **Setup**: Ensured connection between Next.js project and Sanity CMS.
2. **Built Dynamic Components**:
    - Created reusable components like ProductCard, ProductList, SearchBar, CategoryFilter, etc.
    - Implemented useState and useEffect for fetching and displaying dynamic data.
    - Utilized Next.js dynamic routing for product detail pages.
3. **Integrated API Calls**:
    - Used getServerSideProps for fetching real-time product data.
    - Implemented a search filter function.
4. **State Management**:
    - Used React useContext for managing global states (e.g., cart, wishlist).
5. **Optimized Performance**:
    - Implemented lazy loading for images.
    - Used pagination to manage large datasets.

Challenges & Solutions

Challenge	Solution
API Data Formatting Issues	Mapped API fields correctly in the schema.
Search Performance Lag	Used debounce for optimizing search execution.
Responsive Design Tweaks	Used Tailwind's responsive utility classes.

Best Practices Followed

- **Modular & Reusable Components**: Ensured scalability.
- **Error Handling**: Added proper try-catch blocks for API calls.
- **Performance Optimization**: Used lazy loading & pagination.