REPORT DAY 4

Steps to Build and Integrate Components

1. Objective

- Create a product listing page with search functionality.
- Enable navigation to individual product detail pages.

2. Components Built

- **Product List Component**: Displays all products in a grid, includes a search bar for filtering.
- **Product Detail Component**: Shows detailed information about a single product.

3. Integration Steps

- **Dynamic Routing**: Configured dynamic routes in Next.js (`pages/product/[id].tsx`) for individual products.
- **Data Fetching**: Used `getServerSideProps` to fetch data from Sanity CMS for both listing and detail pages.
- **Search Feature**: Added React state and filtering logic for live product searches.

4. Styling

- Used **Tailwind CSS** for a responsive and clean layout.

5. Testing

- Verified search functionality, navigation, and responsive design.

Challenges Faced and Solutions Implemented

1. **Challenge**: **Dynamic Routing for Product Details**

    - **Problem**: Linking the product listing to individual product detail pages.
    - **Solution**: Used Next.js dynamic routes (`/product/[id]`) to fetch product-specific data using `getServerSideProps`.

2. **Challenge**: **Search Functionality**
    - **Problem**: Real-time filtering of products without affecting performance.
    - **Solution**: Implemented client-side search with React `useState` and `useEffect` to filter products dynamically.

3. **Challenge**: **Image Handling**
    - **Problem**: Some products lacked images.
    - **Solution**: Added a fallback message (`No Image Found`) for products without images.

4. **Challenge**: **Styling and Responsiveness**
    - **Problem**: Ensuring the UI worked across all screen sizes.
    - **Solution**: Used Tailwind CSS for a clean and responsive design.

Best Practices Followed During Development

1. **Component Reusability**
    - Separated the product list and product detail into reusable components to promote code modularity and maintainability.
2. **Dynamic Routing**
    - Leveraged Next.js dynamic routing for seamless navigation and fetching of product-specific data.
3. **Responsive Design**
    - Used Tailwind CSS utility classes to ensure consistent and responsive layouts across different devices.
4. **Error Handling**
    - Added fallbacks for missing data, such as displaying "No Image Found" for products without images.
5. **Client-Side Optimization**
    - Implemented `useState` and `useEffect` for real-time search functionality without requiring server calls.
6. **Clean and Readable Code**
    - Used clear variable names, comments, and proper indentation to make the code understandable and easier to debug.
7. **Separation of Concerns**
    - Kept logic for fetching data server-side, while handling UI interactions client-side, ensuring better performance and scalability.