TECHNICAL REPORT

Steps taken to build and integrate components:

- Dynamic routing aur product detail page banaya peers youtube aur seniors sy guide ly kar.
- Search bar integration and category filtering aur price filtering bhi ChatGPT sy aur seniors sy guide lyty hovy banaye.
- Sab kuch build tu horha tha par errors myn phans gye thit u phir help lykr solve kiye thy.
- Sabsy pehly myny pdf parh kar ek todo components banaye apni list myn, issy mujhy yh asani hogye k mujhy mery template myn kia kia chezien integrate krni hongi
- Ek ek kar ky myny sary components ko create karna shro kia, aur jhn mushkil aye pehly jitna mujhy aata tha ussy solve krny ki khoshish ki, mujhsy hogya tu aagy barh gye, nhn hova tu ChatGPT sy help li , chatgpt sy solve hogya tu good, phir warna group k seniors sy help li.

Components Integrated in Website:

- Fetch Data from provide API
- Display this Data to the UI of website
- Add dynamic routing of product detail page
- Integrate Cart functionality
- Add search bar functionality
- Add category dropdown functionality

- Add price filtering functionality
- Add checkout functionality with Shipping information form
- Add counter of amount functionality
- Add contact us functionality through FormSpree
- Add subscribe by Sign Up functionality
- Add user page integration

Tools Using for Components Integrated in Website:

- **PROVIDED API:** Use provided API to fetch data from Api to Sanity then from Sanity to Ui of website
- **React**: React is used for building the user interface and managing the state (useState, useEffect).
- **SweetAlert2 (Swal)**: **SweetAlert2** is used to display popups (alerts) for confirming actions such as removing items from the cart or proceeding to checkout.
- **Next.js**: project is built with **Next.js** (example: useRouter hook), which handles server-side rendering and routing.
- **Sanity**: **Sanity** is used for fetching images via the urlFor function, which is specifically designed to fetch and optimize images from the Sanity CMS.
- **Custom Functions (getCartItems, removeFromCart, updateCartQuantity)**: These are custom functions used to manage cart data. They might be defined elsewhere in your app, and they handle getting, removing, and updating cart items.
- **Next Image**: The **Next.js Image component** is used to display optimized images on the page.
- **Categories**: fetch Categories fetches categories from Sanity CMS on component mount.

- **All Products**: fetch All Products fetches all products for initial display and for the "You might also like" section.
- **Products by Category**: fetchProductsByCategory fetches products filtered by a selected category.

Category Selection:

- When a user selects a category from the dropdown, the handleCategorySelect function is triggered, filtering products by the selected category.

Price Sorting:

- The handlePriceSort function sorts the filteredData either in ascending or descending order based on the selected price sorting option.
- **Search**:
- The handleSearchChange function filters products based on the entered search term. It looks for matches in the product's name or tags.

Rendering:

- **Dropdowns**: Displays two dropdowns: one for selecting a category and the other for sorting by price.
- **Products**: Products are displayed in a grid. Each product has a name, price, and image, and is linked to a product detail page.
- **You Might Also Like Section**: Displays all products in a grid layout below the main product list.

Forms:

Formspree: A service used to handle form submissions via email without needing a backend.

- **Tailwind CSS**: A utility-first CSS framework for styling the form and layout.
- **useState Hook**: For managing the state of the form status

Challenges faced and solutions implemented:

- **Form Submission Handling**:

  Challenge: Sending data to Formspree and managing form state.

  Solution: Used fetch API to send data, tracked status with useState, and displayed success/error messages.

- **Asynchronous Operations**:

  Challenge: Ensuring proper handling of async requests.

  Solution: Wrapped form submission in async/await and used .then()/.catch() for proper feedback.

- **State Management Across Multiple Components:**
- *Challenge:* Managing the state of the cart items, category data, form submissions, and products across different components.
- *Solution:* Utilized React's useState and useEffect hooks to manage and fetch data dynamically across different components (e.g., cart items, categories, products).
- **Asynchronous Data Fetching:**
- *Challenge:* Fetching data from external sources like Sanity and Formspree and managing the asynchronous nature of these requests.
- *Solution:* Used async/await with useEffect to fetch categories, products, and form submissions. Data is updated and displayed once the fetch is complete.
- **Conditional Rendering & Dynamic Content:**
- *Challenge:* Dynamically rendering product lists, categories, and filtering based on user interactions.
- *Solution:* Implemented conditional rendering using isDropdownOpen, filteredData, and category-based filtering. React states were used to toggle visibility and filter the displayed content.
- **Form Validation & Error Handling:**
- *Challenge:* Ensuring form validation for required fields and managing user feedback.

- *Solution:* Integrated HTML5 form validation with additional state-based success/error messages (e.g., after submitting the contact form).
- **Handling User Interactions (Cart, Category Selection, Price Sorting):**
- *Challenge:* Managing cart actions such as quantity changes, item removal, and price sorting across the product listing page.
- *Solution:* Used event handlers like handleRemove, handleQuantityChange, handlePriceSort, and handleCategorySelect to manage cart items and product filtering.

Best practices followed during development:

Separation of Concerns (Component Modularity):

- **Best Practice:** Code is divided into smaller, reusable components such as Topnav, Cartpage, CategoryList, etc., which helps in improving maintainability and scalability.
- **Benefit:** This makes it easier to manage, debug, and extend the application since each component has a single responsibility.

State Management with React Hooks:

- **Best Practice:** Utilized useState and useEffect for managing component-level states and side effects (such as fetching data, handling form submission, etc.).
- **Benefit:** Helps in efficiently managing dynamic data and side effects within the application, making it predictable and easier to test.

Form Validation & User Feedback:

- **Best Practice:** Used HTML5 form validation for required fields and custom validation for form submission. Additionally, provided clear feedback (success/error messages) after form submission.
- **Benefit:** Enhances user experience by preventing invalid form submissions and guiding the user to success/error messages.

Asynchronous Data Fetching with Error Handling:

- **Best Practice:** Used async/await for fetching data from APIs (e.g., Formspree, Sanity), ensuring smooth asynchronous operations. Also included error handling to provide meaningful feedback if data fetching fails.
- **Benefit:** Ensures data is fetched asynchronously without blocking the UI and allows graceful handling of errors (e.g., showing a fallback message).

Dynamic Rendering with Conditional Logic:

- **Best Practice:** Applied conditional rendering techniques to show and hide UI elements dynamically based on the component state (e.g., category dropdown, product filters, cart updates).
- **Benefit:** Improves performance and user experience by only rendering necessary elements, preventing unnecessary re-renders.

Optimizing Performance:

- **Best Practice:** Avoided direct manipulation of the DOM and utilized React's state to trigger updates. Additionally, data fetching is done asynchronously on mount, reducing initial page load time.

- **Benefit:** Enhances performance by ensuring that only necessary UI updates occur, reducing unnecessary re-renders.

Reusability and DRY Principle (Don't Repeat Yourself):

- **Best Practice:** Components like CategoryList, Topnav, and cart item handling logic are reusable, reducing code duplication.
- **Benefit:** Increases code maintainability and reduces bugs due to repetitive code. Reusable components can be easily extended and reused in different parts of the application.

Error Handling and Validation:

- **Best Practice:** Integrated error handling in asynchronous actions (e.g., try/catch in form submission and data fetching). Also, implemented proper form validation to ensure data integrity.
- **Benefit:** Ensures a better user experience and helps in identifying potential issues early in the development process.

Optimized Image Handling:

- **Best Practice:** Used image optimization techniques like the next/image component to serve responsive and optimized images, ensuring quicker loading times.
- **Benefit:** This reduces page load times, improves the app's performance, and provides a better user experience, especially on slower networks.

Consistent UI/UX Design:

- **Best Practice:** Followed consistent styling guidelines using Tailwind CSS to ensure a clean and user-friendly interface.
- **Benefit:** Ensures visual consistency across the app and delivers a cohesive user experience.

Focus on Accessibility:

- **Best Practice:** Focused on providing better accessibility by using semantic HTML, adding focus rings, and ensuring form inputs are properly labeled.
- **Benefit:** Ensures the application is usable for all users, including those with disabilities, improving inclusivity.

Code Readability and Documentation:

- **Best Practice:** Code is clean, modular, and well-documented with appropriate comments explaining complex logic.
- **Benefit:** Enhances code maintainability, making it easier for other developers (or future you) to understand and build upon.