Fakestore API App

**Principles & Decisions Applied**

I used redux state management library along with few internal state properties in React components to accomplish the development of this application. Along with that Redux-Thunk middleware is also used for handling async action creators.   
The reason for choosing Redux state management library was to have a single source of truth and centralize the main application data at one place. This would provide the following benefits:

1. The app components could refer to the same state data thus making the code look more modularized with Redux’s actions, reducers. For example: Fetching cart details required to fetch the product details to get the product price, title and image details for updating the cart menu. Rather than passing these properties specifically it is better to have one central store from where the component can access all this data.
2. It would avoid the need to send down props at different parts of the application which would help in improving the code reusability.
3. Redux thunk library is helpful in making sure async actions can be completed to improve the performance of the application. Async actions makes the operations non-blocking thus making them better than synchronous calls.
4. Action to filter products at this stage supports filtering by 1 category only as the API given filters on one argument. Ideally, this can be done on multiple filters but assuming the app needs to demonstrate the usage of Fake store API I decided to keep the filtering limited to 1 category only.
5. Action to sort products does not use the Fake store API. It fetches all the products and makes use of a comparator function to sort by any product property. The reason to do this was because the sort API only sorts the products on their product Ids. This won’t accomplish the task of sorting by price.

**Tool Selection Motivation**

The motivation to use Redux state management library was to have the data centralized at one place to allow multiple components to refer to one single source of truth. This makes the code more modularized and avoids the use of passing too many props here and there and to components which may be deeply nested.

**Further Improvements**

I would like to implement the following functionalities in the future to update the application:

1. Add checkout feature for shopping cart with connection to a payment API like Stripe, etc.
2. Learn more about lazy loading and implement lazy loading to load products on scrolling.
3. Include a map-based component to place marker on user profile screen of the user address. Preferably, Google maps
4. Understand and implement the functionality for scrolling to a particular point in page on back/ forward navigation. Some ideas would be to use state-based properties and call scroll into view with arguments applied to achieve this functionality.
5. Experiment with different designs to improve my understanding on UX and build the app using different design wireframes.

**Please Note** -> While working on the app, I found that the PUT call wasn’t working for updating the user even if I tried the payload mentioned in the fakestore api docs. I think the Put action creator is taking the correct user object payload as part of the request in the application code. Please do let me know in feedback if the same issues were found on review.