Alright, let's dive into the code and address your queries about props, hooks (useState, useEffect, useContext), and their functionalities with real-life examples.

Understanding the Code: StepCoreInfo.jsx

The code you provided is a React component named StepCoreInfo. It's likely part of a larger application, possibly a form or a step-by-step wizard, where users input information about a "Core" entity. Let's break down the key parts and concepts.

1. Props (Properties)

\* What are Props?

\* Props are how you pass data from a parent component to a child component in React. They are like function arguments, but for components. They are read-only within the child component.

\* Real-Life Example:

\* Imagine you have a ProfileCard component. You might pass the user's name, profile picture URL, and job title as props from a UserList component:

\* <ProfileCard name="Alice" imageUrl="alice.jpg" jobTitle="Software Engineer" />

\* In the StepCoreInfo component you provided, we see props being used like this:

\* function StepCoreInfo(props) { ... }

\* And then props are used like this props.setFeedDraft, props.nextStep

\* This means that the parent component is passing down functions to this child component.

2. useState Hook

\* What is useState?

\* useState is a React Hook that lets you add state variables to functional components. State is data that can change over time, triggering re-renders of the component when it changes.

\* Real-Life Example:

\* A light switch: The state is either "on" or "off." When you click the switch, the state changes, and the light turns on or off.

\* In the code you provided, you see useState used like this:

\* const [currProjObj, setCurrProjObj] = useState({});

\* const [validAndSave, setValidAndSave] = useState(false);

\* const [feedNameError, setFeedNameError] = useState('');

\* const [errorCaption, setErrorCaption] = useState('Match the pattern, hover on information icon to see details with examples');

\* const [errorDescription, setErrorDescription] = useState('');

\* These lines create state variables (e.g., currProjObj, validAndSave) and functions to update them (e.g., setCurrProjObj, setValidAndSave).

3. useEffect Hook

\* What is useEffect?

\* useEffect is a React Hook that lets you perform side effects in functional components. Side effects are actions that interact with the outside world, such as fetching data, setting up subscriptions, or directly manipulating the DOM.

\* Real-Life Example:

\* Fetching weather data when a user enters a city. The side effect is the data fetching, and it happens when the city input changes.

\* In the code you provided, you see useEffect used like this:

\* useEffect(() => {

const project = getProjectName();

setCurrProjObj(project);

if (feedDraft?.core?.feedUniqueName) {

props.setFeedDraft((current) => ({

...current,

core: {

...current?.core,

feedUniqueName: project?.name ? `${project?.name} -` : '',

},

}));

}

}, [userData, currProjObj]);

\* This useEffect hook runs when userData or currProjObj changes. It calls getProjectName() and updates state.

\* useEffect(() => {

if (data?.errorMessage && data?.data?.feedMaster?.feedSequence) {

sideNavigationCtx.count++;

setValidAndSave(false);

props.setFeedDraft((current) => ({

...current,

core: {

...current?.core,

feedSequence: data.data.feedMaster.feedSequence,

},

}));

props.setCompleted(0);

props.setActive(1);

} else {

setValidAndSave(false);

}

}, [data]);

\* This useEffect hook runs when data changes. It updates the state, and also calls functions that where passed in by props.

4. useContext Hook

\* What is useContext?

\* useContext is a React Hook that lets you access context values in functional components. Context provides a way to share values between components without explicitly passing props through every level of the component tree.

\* Real-Life Example:

\* A theme (light or dark) in an application. You can use context to make the theme available to all components without passing it as a prop to each one.

\* In the code you provided, you see useContext used like this:

\* const sideNavigationCtx = useContext(SideNavigationContext);

\* This line gets the context value from SideNavigationContext. This context likely provides data or functions related to side navigation.

Code Breakdown

\* getUser(): This function retrieves user profile information from a cookie.

\* canEditCoreInfo: This variable determines if the user has permission to edit core information.

\* State Variables: The component uses useState to manage various pieces of state, such as form data, validation errors, and loading states.

\* handleNameChange(): This function updates the feed name state and performs validation.

\* validateTransferMode(): This function handles the selection of data transfer modes.

\* setFeedDescription(): This function updates the feed description state and performs validation.

\* saveData(): This function saves the form data.

\* getProjectName(): This function retrieves the project name.

\* isSavedBtnDisabled(): This function determines if the save button should be disabled.

\* updateStep(): This function updates the step in the form wizard.

\* useEffect(): The component uses useEffect to perform side effects, such as fetching data and updating state based on changes in props or state.

\* JSX: The component renders a form with input fields, buttons, and error messages. It uses Material-UI components for styling.

In Summary

The StepCoreInfo component is a complex form component that uses props to receive data and functions from its parent, state to manage its own internal data, and effects to perform side effects. It also utilizes context to access shared data.

Let me know if you have any more questions or want to dive deeper into any specific aspect of the code!