**What is mui/material in React?**

Material UI is an open-source React component library that implements Google's Material Design. It includes a comprehensive collection of prebuilt components that are ready for use in production right out of the box.

**What is mui/icon/material in react?**

@mui/icons-material includes the 2,100+ official Material Icons converted to SvgIcon components. It depends on @mui/material , which requires Emotion packages. Use the following command to install it: npm install @mui/icons-material @mui/material @emotion/styled @emotion/react.

**What is emotion/styled?**

Emotion is a library designed for writing css styles with JavaScript. It provides powerful and predictable style composition in addition to a great developer experience with features such as source maps, labels, and testing utilities. Both string and object styles are supported.

**What is React-router-Dom?**

React Router DOM is an npm package that enables you to implement dynamic routing in a web app. It allows you to display pages and allow users to navigate them. It is a fully-featured client and server-side routing library for React.

**What is uuid in react?**

Fortunately, create-react-app ships with an industry-standard option: a JS library called UUID. The UUID library is designed to quickly generate UUIDs. UUID stands for "universally unique identifier". They're also sometimes referred to as GUID, or "globally unique identifiers".

**What are web-vitals in react?**

Web Vitals are a set of useful metrics that aim to capture the user experience of a web page. In Create React App, a third-party library is used to measure these metrics (web-vitals). To understand more about the object returned to the function when a metric value is calculated, refer to the documentation.

**What is @hello-pangea/dnd in react?**

@hello-pangea/dnd is a higher level abstraction specifically built for lists (vertical, horizontal, movement between lists, nested lists and so on). Within that subset of functionality @hello-pangea/dnd offers a powerful, natural and beautiful drag and drop experience.

**Explain this piece of code.**

const archiveNote = (note) => {

        const updatedNotes = notes.filter(data => data.id !== note.id);

        setNotes(updatedNotes);

        setArchiveNotes(prevArr => [note, ...prevArr]);

    }

This code defines a function called archiveNote that takes a note object as a parameter. Let's break down what the code does step by step:

1. const updatedNotes = notes.filter(data => data.id !== note.id);

This line creates a new array called updatedNotes by filtering the existing notes array. It removes any element from notes whose id property does not match the id of the note parameter. In other words, it removes the note object from the notes array.

1. setNotes(updatedNotes);

This line updates the notes state variable with the updatedNotes array. The setNotes function is usually provided by a state management library like React's useState, and it is used to update the state value and trigger a re-render of the component.

1. setArchiveNotes(prevArr => [note, ...prevArr]);

This line updates the archiveNotes state variable by creating a new array. It uses the functional form of the state updater, where the previous state (prevArr) is passed as an argument to a function. The function returns a new array that includes the note object followed by the elements of the previous state array. This effectively adds the note object to the beginning of the archiveNotes array.

Overall, the archiveNote function removes the specified note object from the notes array and adds it to the beginning of the archiveNotes array.

**Explain this Piece of code.**

const StyledCard = styled(Card)`

    width: 240px;

    margin: 8px;

    box-shadow: none;

    border: 1px solid #e0e0e0;

    border-radius: 8px

`

This code defines a styled component called StyledCard using a library like styled-components. Let's break down what each line of the code does:

const StyledCard = styled(Card):

This line creates a new styled component called StyledCard by invoking the styled function on the Card component. The styled function is provided by the styled-components library and allows you to create styled components based on existing components. In this case, Card is the base component that we want to style.

``:

The backtick ( ) signifies that the following code is a template literal, allowing you to write CSS within the template.

Overall, this code creates a new styled component called StyledCard that applies specific CSS styles to the Card component. The resulting StyledCard component will have a fixed width of 240px, a margin of 8px, no box-shadow, a solid border with a color value of #e0e0e0, and rounded corners with a radius of 8px.

**Explain this piece of code.**

const reorder = (list, startIndex, endIndex) => {

    const result = Array.from(list);

    const [removed] = result.splice(startIndex, 1);

    result.splice(endIndex, 0, removed);

    return result;

  };

This code defines a function named reorder that takes in three parameters: list, startIndex, and endIndex. Here's an explanation of what the code does:

The first line of the function initializes a variable named result with a new array created using Array.from(list). This creates a copy of the original list array so that any modifications made to result won't affect the original array.

The second line of the function uses array destructuring to assign the first element of the result array (which is removed from the array) to the variable named removed. The splice method is used to remove one element from the result array starting at the index specified by the startIndex parameter.

The third line of the function uses the splice method again to insert the removed element back into the result array at the index specified by the endIndex parameter. The second argument 0 means no elements are removed before the insertion.

Finally, the function returns the modified result array, which now reflects the reordering of the element from startIndex to endIndex within the array.

**Explain this piece of code.**

const onDragEnd = (result) => {

        if (!result.destination) {

            return;

        }

        const items = reorder(notes, result.source.index, result.destination.index);

        setNotes(items);

    }

This code defines a function called onDragEnd that takes a single parameter called result. Here's a breakdown of what the code does:

1. The function first checks if result.destination is defined. If it is not, it means that the dragged item was not dropped on a valid destination, so the function simply returns without performing any further actions.
2. If result.destination is defined, it means that the dragged item was dropped on a valid destination. In this case, the code proceeds to the next steps.
3. The code calls a function called reorder and passes three arguments: notes, result.source.index, and result.destination.index. This suggests that there is an array of notes and the reorder function is responsible for reordering the items in the array. The result.source.index and result.destination.index are the indices of the item being dragged and the destination index where it was dropped, respectively.
4. The reorder function is expected to return a new array with the items reordered according to the drag and drop action. The code assigns this new array to a variable called items.
5. Finally, the code calls a function called setNotes and passes items as an argument. This indicates that setNotes is responsible for updating the state or value of the notes variable to reflect the reordered items.

**Explain this piece of code.**

const navList = [

        { id: 1, name: 'Notes', icon: <LightbulbOutlinedIcon />, route: '/' },

        { id: 2, name: 'Archive', icon: <ArchiveOutlinedIcon />, route: '/archive' },

        { id: 3, name: 'Trash', icon: <DeleteOutlineOutlinedIcon />, route: '/delete' }

    ]

The given code is defining an array called navList that contains objects representing navigation items. Each object represents a specific item in a navigation list and has the following properties:

1. id: An identifier for the item. It is a numeric value (1, 2, 3 in this case) and serves as a unique identifier for each navigation item.
2. name: The name or label of the navigation item. It is a string value ('Notes', 'Archive', 'Trash' in this case) and represents the display text for each item in the navigation list.
3. icon: An icon associated with the navigation item. It appears that the code is using custom icon components from a library. In the provided code snippet, <LightbulbOutlinedIcon />, <ArchiveOutlinedIcon />, and <DeleteOutlineOutlinedIcon /> are JSX components representing icons for 'Notes', 'Archive', and 'Trash' respectively. These components are likely imported from an external library or defined elsewhere in the code.
4. route: The route or URL associated with the navigation item. It is a string value ('/', '/archive', '/delete' in this case) and represents the destination URL or path that the user will be directed to when they click on the corresponding navigation item.

Overall, the navList array is a collection of navigation items with their respective names, icons, and routes. It can be used to render a navigation menu or list in a user interface, where each item represents a clickable link to a specific page or section of the application.

**What is DragDropContext, Droppable, Draggable in @hello-pangea/dnd?**

In the context of the @hello-pangea/dnd library, DragDropContext, Droppable, and Draggable are components that facilitate drag and drop functionality in a user interface. These components are typically used in combination to create draggable and droppable elements within an application.

1. DragDropContext: This is a higher-order component provided by the @hello-pangea/dnd library. It serves as a wrapper component that enables drag and drop functionality within its children components. DragDropContext is responsible for managing the state and interactions related to drag and drop operations. It handles the initialization of the drag and drop system and provides callbacks for handling events such as when an item is dragged or dropped. In order for the drag and drop functionality to work, the DragDropContext component needs to be placed at the root of the component tree.
2. Droppable: This component represents an area or container within which draggable elements can be dropped. It defines the boundaries and behavior of a drop target. The Droppable component needs to be nested within a DragDropContext component. It accepts a unique identifier (droppableId) as a prop to identify itself. The Droppable component can wrap one or more Draggable components, creating a draggable area.
3. Draggable: This component represents an item or element that can be dragged within a Droppable area. It defines the draggable content and its behavior. The Draggable component needs to be nested within a Droppable component. It accepts a unique identifier (draggableId) as a prop to identify itself. Additionally, it can receive a dragHandleProps prop to specify a specific area within the draggable element that serves as the drag handle. This allows for more granular control over which part of the element triggers the drag operation.

By using these components together, developers can create interactive interfaces that allow users to drag and drop elements, rearrange items, or perform other actions based on drag and drop interactions. The @hello-pangea/dnd library provides the necessary abstractions and utilities to handle the complexities of managing and updating the state during drag and drop operations.