```
import React, { useState, useRef } from 'react';

const Card = ({ id, text, onDragStart, onDragEnd, position }) => {
  return (
    <div draggable onDragStart={(e) =>
      onDragStart(e, id)} onDragEnd={(e) =>
      onDragEnd(e, id)}
      style={{
        position: 'absolute', top:
        position.y, left:
        position.x, padding:
        '10px', border: '1px
        solid black', cursor:
        'grab',
      }}
    >
      {text}
    </div>
  );
};

const Canvas = () => {
  const [cards, setCards] = useState({
    card1: { text: 'Card 1', position: { x: 50, y: 50 } },
    card2: { text: 'Card 2', position: { x: 150, y: 150 } },
  });

  const handleDragStart = (e, id) => {
    const card = cards[id];
    e.dataTransfer.setData('cardId', id);
    e.dataTransfer.setData('offsetX', e.clientX - card.position.x);
    e.dataTransfer.setData('offsetY', e.clientY - card.position.y);
  };

  const handleDragEnd = (e, id) => {
    const offsetX = parseInt(e.dataTransfer.getData('offsetX'), 10);
    const offsetY = parseInt(e.dataTransfer.getData('offsetY'), 10);

    const newCards = {
      ...cards,
```

```
      [id]: {
        ...cards[id],
        position: {
          x: e.clientX - offsetX,
          y: e.clientY - offsetY,
        },
      },
    };

    setCards(newCards);
  };

  return (
    <div style={{ width: '100%', height: '400px', position: 'relative', border: '2px solid blue' }}>
      {Object.entries(cards).map(([id, card]) => (
        <Card
          key={id} id={id} text={card.text}
          position={card.position}
          onDragStart={handleDragStart
          }
          onDragEnd={handleDragEnd}
        />
      ))}
    </div>
  );
};

const App = () => {
  return (
    <div>
      <h2>Drag-and-Drop UI</h2>
      <Canvas />
    </div>
  );
};

export default App;
```

Thought Process for Designing a Drag-and-Drop Feature:

When designing a Drag-and-Drop feature, the following considerations were made:

User Interaction: The user should be able to interact seamlessly with the cards on the canvas. This includes dragging, dropping, and connecting cards with arrows.

Component Structure: Each card, arrow, and popup is a React component. The canvas serves as the container where all these components interact.

State Management: The state of the cards (position, size, connections) is managed using React's useState or a state management library like Redux. This ensures the UI is responsive and updates in real-time.

Reusability and Extensibility: The components are designed to be modular so that they can be easily reused and extended. For example, adding a new type of card or a different type of connection (e.g., dotted arrows) can be done with minimal changes to the existing code.

Performance Optimization: As users interact with the UI, especially with multiple cards, performance can degrade. Techniques such as memoization with React.memo and optimizing rendering by only updating affected components are considered.

Accessibility: Ensuring that the drag-and-drop interface is accessible to all users, including those using keyboards or screen readers. This involves adding ARIA labels and keyboard shortcuts.