src\ui.jsx

```jsx
// ui.js
// Displays the drag-and-drop UI
// -----------------------------------------------

import { useState, useRef, useCallback } from 'react';
import ReactFlow, { Controls, Background, MiniMap } from 'reactflow';
import { useStore } from './store';
import { shallow } from 'zustand/shallow';
import { InputNode } from './nodes/inputNode';
import { LLMNode } from './nodes/llmNode';
import { OutputNode } from './nodes/outputNode';
import { TextNode } from './nodes/textNode';

import 'reactflow/dist/style.css';

const gridSize = 20;
const proOptions = { hideAttribution: true };
const nodeTypes = {
  customInput: InputNode,
  llm: LLMNode,
  customOutput: OutputNode,
  text: TextNode,
};

const selector = (state) => ({
  nodes: state.nodes,
  edges: state.edges,
  getNodeID: state.getNodeID,
  addNode: state.addNode,
  onNodesChange: state.onNodesChange,
  onEdgesChange: state.onEdgesChange,
  onConnect: state.onConnect,
});

export const PipelineUI = () => {
    const reactFlowWrapper = useRef(null);
    const [reactFlowInstance, setReactFlowInstance] = useState(null);
    const {
      nodes,
      edges,
      getNodeID,
      addNode,
      onNodesChange,
      onEdgesChange,
      onConnect
    } = useStore(selector, shallow);

    const getInitNodeData = (nodeID, type) => {
      let nodeData = { id: nodeID, nodeType: `${type}` };
      return nodeData;
    }

    const onDrop = useCallback(
        (event) => {
```

```
            event.preventDefault();

            const reactFlowBounds = reactFlowWrapper.current.getBoundingClientRect();
            if (event?.dataTransfer?.getData('application/reactflow')) {
              const appData = JSON.parse(event.dataTransfer.getData('application/reactflow'));
              const type = appData?.nodeType;

              // check if the dropped element is valid
              if (typeof type === 'undefined' || !type) {
                return;
              }

              const position = reactFlowInstance.project({
                x: event.clientX - reactFlowBounds.left,
                y: event.clientY - reactFlowBounds.top,
              });

              const nodeID = getNodeID(type);
              const newNode = {
                id: nodeID,
                type,
                position,
                data: getInitNodeData(nodeID, type),
              };

              addNode(newNode);
            }
        },
        [reactFlowInstance]
    );

    const onDragOver = useCallback((event) => {
        event.preventDefault();
        event.dataTransfer.dropEffect = 'move';
    }, []);

    return (
        <>
        <div ref={reactFlowWrapper} style={{width: '100wv', height: '70vh'}}>
            <ReactFlow
                nodes={nodes}
                edges={edges}
                onNodesChange={onNodesChange}
                onEdgesChange={onEdgesChange}
                onConnect={onConnect}
                onDrop={onDrop}
                onDragOver={onDragOver}
                onInit={setReactFlowInstance}
                nodeTypes={nodeTypes}
                proOptions={proOptions}
                snapGrid={[gridSize, gridSize]}
                connectionLineType='smoothstep'
            >
                <Background color="#aaa" gap={gridSize} />
                <Controls />
                <MiniMap />
```

```
            </ReactFlow>
        </div>
        </>
    )
}

```