```
/home/project/
├── src/                    # Source code directory
│   ├── components/         # React components
│   │   ├── FlowCanvas.tsx  # Main canvas for the flow editor
│   │   ├── Navigation.tsx  # Top navigation bar
│   │   ├── NodePanel.tsx   # Side panel showing available nodes
│   │   ├── Toolbar.tsx     # Top toolbar with actions
│   │   └── nodes/          # Node components
│   │       ├── BaseNode.tsx # Base node component
│   │       ├── ConditionNode.tsx # Condition node implementation
│   │       └── index.tsx   # Node type definitions
│   ├── store/              # State management
│   │   ├── flowStore.ts    # Flow editor state
│   │   └── themeStore.ts   # Theme state (dark/light)
│   ├── types/              # TypeScript type definitions
│   │   └── flow.ts         # Flow-related types
│   ├── utils/              # Utility functions
│   │   ├── nodeExecutors.ts # Node execution logic
│   │   └── pipelineExecutor.ts # Pipeline execution
│   ├── App.tsx             # Root component
│   └── main.tsx            # Entry point
├── public/                 # Static assets
├── index.html              # HTML template
├── package.json            # Project dependencies
├── tsconfig.json           # TypeScript configuration
└── vite.config.ts          # Vite configuration
```

FlowCanvas.tsx

```
import React, { useCallback, useRef, useLayoutEffect } from 'react';
import ReactFlow, {
  Background,
  Controls,
  Panel,
  Connection,
  Edge,
  ReactFlowProvider,
  Node,
} from 'reactflow';
import 'reactflow/dist/style.css';
import { useFlowStore } from '../store/flowStore';
import { Save } from 'lucide-react';
import { NodeType } from '../types/flow';
import { nodeTypes } from './nodes';
```

```
export const FlowCanvas: React.FC = () => {
  const {
    nodes,
    edges,
    onNodesChange,
    onEdgesChange,
    onConnect,
    setSelectedNode,
  } = useFlowStore();

  const reactFlowWrapper = useRef<HTMLDivElement>(null);

  useLayoutEffect(() => {
    const resizeObserver = new ResizeObserver(() => {
      window.dispatchEvent(new Event('resize'));
    });

    if (reactFlowWrapper.current) {
      resizeObserver.observe(reactFlowWrapper.current);
    }

    return () => {
      if (reactFlowWrapper.current) {
        resizeObserver.unobserve(reactFlowWrapper.current);
      }
    };
  }, []);

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

  const onDrop = useCallback(
    (event: React.DragEvent) => {
      event.preventDefault();

      const type = event.dataTransfer.getData('application/reactflow') as NodeType;
      if (!type || !reactFlowWrapper.current) return;

      const bounds = reactFlowWrapper.current.getBoundingClientRect();
      const position = {
        x: event.clientX - bounds.left,
        y: event.clientY - bounds.top,
      };

      const newNode: Node = {
        id: `${type}-${Date.now()}`,
        type,
        position,
        data: { label: `${type} node`, type },
      };

      useFlowStore.setState((state) => ({
        nodes: [...state.nodes, newNode],
      }));
    },
    []
  );

  const onNodeClick = useCallback((event: React.MouseEvent, node: Node) => {
    setSelectedNode(node);
  }, [setSelectedNode]);

  return (
    <div ref={reactFlowWrapper} className="flex-1 h-full">
      <ReactFlowProvider>
        <ReactFlow
          nodes={nodes}
          edges={edges}
          onNodesChange={onNodesChange}
          onEdgesChange={onEdgesChange}
          onConnect={onConnect}
          onNodeClick={onNodeClick}
          onDragOver={onDragOver}
          onDrop={onDrop}
          nodeTypes={nodeTypes}
          fitView
          deleteKeyCode="Delete"
          className="bg-gray-50"
        >
          <Background />
          <Controls />
          <Panel position="top-right">
            <button
              className="flex items-center gap-2 px-4 py-2 bg-blue-600 text-white rounded-md
hover:bg-blue-700 transition-colors"
              onClick={() => {
                console.log('Saving workflow...');
              }}
            >
              <Save className="w-4 h-4" />
              Save Workflow
            </button>
          </Panel>
        </ReactFlow>
      </ReactFlowProvider>
    </div>
  );
};
```

Navigation.tsx

```tsx
import React from 'react';
import { Moon, Sun, Search } from 'lucide-react';
import { NodeCategory } from '../types/flow';
import { useTheme } from '../store/themeStore';

interface NavigationProps {
  activeCategory: NodeCategory;
  onCategoryChange: (id: NodeCategory) => void;
  onSearch: (query: string) => void;
}

const categories: { id: NodeCategory; label: string }[] = [
  { id: 'general', label: 'Core Settings' },
  { id: 'llms', label: 'AI Models' },
  { id: 'knowledge-base', label: 'Smart Database' },
  { id: 'integrations', label: 'Connected Apps' },
  { id: 'data-loaders', label: 'Data Import' },
  { id: 'multi-modal', label: 'Mixed Modal' },
  { id: 'logic', label: 'Workflow Rules' },
  { id: 'ai-tools', label: 'AI Tools & SparkLayer' }
];

export const Navigation: React.FC<NavigationProps> = ({
  activeCategory,
  onCategoryChange,
  onSearch,
}) => {
  const { theme, toggleTheme } = useTheme();

  return (
    <div className={`border-b border-gray-200 ${theme === 'dark' ?
'bg-gray-800' : 'bg-white'}`}>
      <div className="px-4 py-2">
        <div className="flex items-center justify-between mb-4">
          <h1 className={`text-xl font-bold ${theme === 'dark' ?
'text-white' : 'text-gray-900'}`}>
            FlowMind AI
          </h1>
          <div className="flex items-center space-x-4">
            <button
              onClick={toggleTheme}
              className={`p-2 rounded-md ${
                theme === 'dark'
                  ? 'bg-gray-700 text-gray-200 hover:bg-gray-600'
                  : 'bg-gray-100 text-gray-600 hover:bg-gray-200'
              }`}
            >
              {theme === 'dark' ? (
                <Sun className="w-5 h-5" />
              ) : (
                <Moon className="w-5 h-5" />
              )}
            </button>
          </div>
        </div>
        <div className="flex items-center space-x-4">
          <div className="relative flex-1 max-w-xs">
            <Search className={`absolute left-3 top-1/2 transform
-translate-y-1/2 w-4 h-4 ${
              theme === 'dark' ? 'text-gray-400' : 'text-gray-500'
            }`} />
            <input
              type="text"
              placeholder="Search nodes..."
              onChange={(e) => onSearch(e.target.value)}
              className={`w-full pl-9 pr-4 py-1.5 text-sm rounded-md
focus:outline-none focus:ring-2 focus:ring-blue-500 ${
                theme === 'dark'
                  ? 'bg-gray-700 border-gray-600 text-white
placeholder-gray-400'
                  : 'border border-gray-300 text-gray-900
placeholder-gray-500'
              }`}
            />
          </div>
          <nav className="flex space-x-1 overflow-x-auto">
            {categories.map(({ id, label }) => (
              <button
                key={id}
                onClick={() => onCategoryChange(id)}
                className={`flex items-center px-3 py-2 text-sm
font-medium rounded-md transition-colors whitespace-nowrap ${
                  activeCategory === id
                    ? theme === 'dark'
                      ? 'bg-blue-900 text-blue-200'
                      : 'bg-blue-50 text-blue-600'
                    : theme === 'dark'
                    ? 'text-gray-300 hover:text-white hover:bg-gray-700'
                    : 'text-gray-600 hover:text-gray-900 hover:bg-gray-50'
                }`}
              >
                {label}
              </button>
            ))}
          </nav>
        </div>
      </div>
    </div>
  );
};
```

NodePanel.tsx

```
import React from 'react';
import { NodeCategory, NodeType } from '../types/flow';
import { Database, Github, Table2, FileText as NotionIcon, Building2,
Mail, MessageSquare, FolderOpen, FileEdit, Webhook, Slack as SlackIcon,
MessageSquare as TeamsIcon, FileText, Send, Globe, Youtube, FileSearch,
Search, Download, Upload, Save, StickyNote, Tractor, Zap, FileInput,
GitBranch, Merge, Brain, BookOpen, Upload as KBLoader, Search as KBSearch,
RefreshCw, File, Apple as Api, FileSpreadsheet, Link, BookOpen as Wiki,
Newspaper, Music, Camera, MessageCircle, Database as DataCollector,
FileText as FileReader, Bell, Users, FileJson, FileCode, Cog, MailCheck,
MailWarning, Clock, Share2 } from 'lucide-react';

interface NodeInfo {
  type: NodeType;
  label: string;
  icon: React.FC<any>;
  description: string;
}

const nodeCategories: Record<NodeCategory, NodeInfo[]> = {
  general: [
    { type: 'input', label: 'Input', icon: FileInput, description:
'User-provided text input' },
  //  { type: 'web-extractor', label: 'Web Extractor', icon: Globe,
description: 'Extracts structured/unstructured text from web pages' },

    { type: 'document-to-text', label: 'Document to Text', icon:
FileSearch, description: 'Converts PDFs, Word files, and scanned images
into text' },
   // { type: 'google-search', label: 'Google Search', icon: Search,
description: 'Retrieves search results dynamically' },
    //{ type: 'http-get', label: 'HTTP GET', icon: Download, description:
'Fetches data from APIs and URLs' },
    //{ type: 'file-upload', label: 'File ', icon: Upload, description:
'Uploads files for processing' },
    { type: 'output', label: ' Output', icon: Send, description: 'Displays
processed text' },
    { type: 'file-save', label: 'File Save', icon: Save, description:
'Stores output in specified formats (PDF, CSV, JSON, DOCX, TXT)' },
    { type: 'note', label: 'Notes', icon: StickyNote, description: 'Saves
output as a note for reference' },
    { type: 'transform', label: 'Transform', icon: Tractor, description:
'Modify, convert, and enrich data before output' },
   // { type: 'api', label: 'API', icon: Api, description: 'Make an API
request to a given URL' },
   { type: 'pipeline', label: 'Workflow', icon: Zap, description: 'Create
and manage workflows' }, // Add Pipeline node
    { type: 'share', label: 'Share', icon: Share2, description: 'Share
workflows with others' }, // Add Share node
  ],
  llms: [
    { type: 'openai', label: 'OpenAI', icon: Zap, description: 'OpenAI
language model integration' },
    { type: 'anthropic', label: 'Anthropic', icon: Brain, description:
'Anthropic language model integration' },
    { type: 'gemini', label: 'Gemini', icon: Zap, description: 'Google
Gemini model integration' },
    { type: 'cohere', label: 'Cohere', icon: Brain, description: 'Cohere
language model integration' },
    { type: 'perplexity', label: 'Perplexity', icon: Brain, description:
'Perplexity language model integration' },
    { type: 'custom', label: 'Custom', icon: Brain, description: 'Custom
language model integration' },
  ],
  'knowledge-base': [
    { type: 'kb-reader', label: 'Query Knowledge', icon: BookOpen,
description: 'Query and retrieve information from knowledge base' },
    { type: 'kb-loader', label: 'Upload Knowledge', icon: KBLoader,
description: 'Load data into an existing knowledge base' },
    { type: 'kb-search', label: 'Smart Search', icon: KBSearch,
description: 'Semantic search across knowledge base' },
    { type: 'kb-sync', label: 'Sync Knowledge', icon: RefreshCw,
description: 'Synchronize knowledge base data' },
  ],
  integrations: [
    { type: 'mysql', label: 'MySQL', icon: Database, description: 'Execute
SQL queries and interact with MySQL databases' },
    { type: 'mongodb', label: 'MongoDB', icon: Database, description: 'Query and manage MongoDB databases' },
    { type: 'github', label: 'GitHub', icon: Github, description: 'Manage GitHub repositories and pull requests' },
    { type: 'airtable', label: 'Airtable', icon: Table2, description: 'Interact with Airtable databases' },
    { type: 'notion', label: 'Notion', icon: NotionIcon, description: 'Manage Notion pages and databases' },
    { type: 'hubspot', label: 'HubSpot', icon: Building2, description: 'Access HubSpot CRM data' },
    { type: 'gmail', label: 'Gmail', icon: Mail, description: 'Send and manage Gmail emails' },
    { type: 'outlook', label: 'Outlook', icon: Mail, description: 'Send and manage Outlook emails' },
    { type: 'discord', label: 'Discord', icon: MessageSquare, description: 'Send messages to Discord channels' },
    { type: 'google-drive', label: 'Google Drive', icon: FolderOpen, description: 'Manage files on Google Drive' },
    { type: 'onedrive', label: 'OneDrive', icon: FolderOpen, description: 'Manage files on OneDrive' },
    { type: 'google-docs', label: 'Google Docs', icon: FileEdit, description: 'Read and write Google Docs' },
    { type: 'slack', label: 'Slack', icon: SlackIcon, description: 'Send and read Slack messages' },
    { type: 'teams', label: 'Microsoft Teams', icon: TeamsIcon, description: 'Send messages to Teams channels' },
    { type: 'make-webhook', label: 'Make Webhook', icon: Webhook, description: 'Send data to Make.com webhooks' },
    { type: 'zapier-webhook', label: 'Zapier Webhook', icon: Webhook, description: 'Send data to Zapier webhooks' },
  ],
  'data-loaders': [
    { type: 'file-loader', label: 'File Loader', icon: File, description: 'Load data from local files' },
    { type: 'api-loader', label: 'API', icon: Api, description: 'Load data from REST APIs' },
    { type: 'csv-loader', label: 'CSV', icon: FileSpreadsheet, description: 'Load and parse CSV files' },
    { type: 'url-loader', label: 'URL', icon: Link, description: 'Load data from URLs' },
    { type: 'wiki-loader', label: 'Wiki', icon: Wiki, description: 'Load
data from Wikipedia' },
    { type: 'youtube-loader', label: 'YouTube', icon: Youtube,
description: 'Load data from YouTube' },
    { type: 'arxiv-loader', label: 'Arxiv', icon: FileText, description:
'Load papers from Arxiv' },
    { type: 'rss-loader', label: 'RSS', icon: Newspaper, description:
'Load data from RSS feeds' },
  ],
  'multi-modal': [
    { type: 'audio-processor', label: 'Audio', icon: Music, description:
'Process and analyze audio files' },
    { type: 'image-processor', label: 'Image', icon: Camera, description:
'Process and analyze images' },
  ],
  logic: [
    { type: 'condition', label: 'Condition', icon: GitBranch, description:
'Branch workflow based on conditions' },
    { type: 'merge', label: 'Merge', icon: Merge, description: 'Combine
multiple inputs' },
    { type: 'time', label: 'Time', icon: Clock, description: 'Trigger
based on time' }, // Added Time node
    { type: 'ttsql', label: 'TTSQL', icon: Database, description: 'SQL
transformation' }, // Added TTSQL node
  ],
  'ai-tools': [
    { type: 'chat-memory', label: 'Chat Memory', icon: MessageCircle,
description: 'Store chat history and context' },
    { type: 'data-collector', label: 'Data Collector', icon:
DataCollector, description: 'Collect and store chat data' },
    { type: 'chat-file-reader', label: 'Chat File Reader', icon:
FileReader, description: 'Read and process chat files' },
    { type: 'outlook-trigger', label: 'Outlook Trigger', icon: MailCheck,
description: 'Trigger workflows from Outlook events' },
    { type: 'gmail-trigger', label: 'Gmail Trigger', icon: MailWarning,
description: 'Trigger workflows from Gmail events' },
    { type: 'text-processor', label: 'Text Processor', icon: FileText,
description: 'Process and analyze text' },
    { type: 'json-handler', label: 'JSON Handler', icon: FileJson,
description: 'Handle JSON data operations' },
    { type: 'file-transformer', label: 'File Transformer', icon: FileCode,
description: 'Transform file formats' },
    { type: 'ai-task-executor', label: 'AI Task Executor', icon: Cog,
description: 'Execute AI-powered tasks' },
    { type: 'notification-node', label: 'Notification Node', icon: Bell,
description: 'Send notifications' },
    { type: 'crm-enricher', label: 'CRM Enricher', icon: Users,
description: 'Enrich CRM data' },
  ],
};

interface NodePanelProps {
  category: NodeCategory;
  searchQuery: string;
  onDragStart: (event: React.DragEvent, nodeType: string) => void;
}

export const NodePanel: React.FC<NodePanelProps> = ({
  category,
  searchQuery,
  onDragStart,
}) => {
  const nodes = nodeCategories[category] || [];
  const filteredNodes = nodes.filter(
    (node) =>
      node.label.toLowerCase().includes(searchQuery.toLowerCase()) ||
      node.description.toLowerCase().includes(searchQuery.toLowerCase())
  );

  return (
    <div className="p-4">
      <h2 className="text-lg font-semibold text-gray-900 mb-4">
        {category === 'ai-tools' ? 'AI Tools & SparkLayer' :
category.replace('-', ' ').replace(/\b\w/g, l => l.toUpperCase())}
      </h2>
      <div className="grid grid-cols-1 gap-3">
        {filteredNodes.map(({ type, label, icon: Icon, description }) => (
          <div
            key={type}
            draggable
            onDragStart={(e) => onDragStart(e, type)}
            className="flex items-start p-3 bg-white border
border-gray-200 rounded-lg cursor-move hover:border-blue-500
hover:shadow-md transition-all group"
          >
            <div className="flex-shrink-0 p-2 bg-blue-50 rounded-md
group-hover:bg-blue-100 transition-colors">
              <Icon className="w-5 h-5 text-blue-600" />
            </div>
            <div className="ml-3">
              <h3 className="text-sm font-medium
text-gray-900">{label}</h3>
              <p className="text-xs text-gray-500 mt-1">{description}</p>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};
```

E:\refer code\FlowMindAi-main\src\components\Toolbar.tsx

```
import React, { useState } from 'react';
import { Play, Save, Share2, Download, Upload, Plus, Settings, Loader2,
AlertCircle } from 'lucide-react';
import { useFlowStore } from '../store/flowStore';
import { PipelineExecutor } from '../utils/pipelineExecutor';

export const Toolbar: React.FC = () => {
  const [workflowName, setWorkflowName] = useState('Untitled Pipeline');
  const { nodes, edges, clearWorkflow, updateNodeResults } =
useFlowStore();
  const [saveStatus, setSaveStatus] = useState<'saved' | 'saving' |
'unsaved'>('saved');
  const [isRunning, setIsRunning] = useState(false);
  const [executionError, setExecutionError] = useState<string |
null>(null);

  const handleSave = async () => {
    setSaveStatus('saving');
    // Implement save logic here
    setTimeout(() => setSaveStatus('saved'), 1000);
  };

  const handleDeploy = async () => {
    // Implement deployment logic
    console.log('Deploying workflow...');
  };

  const handleExport = () => {
    const workflow = {
      name: workflowName,
      nodes,
      edges,
      exportedAt: new Date().toISOString(),
    };
    const blob = new Blob([JSON.stringify(workflow, null, 2)], { type:
'application/json' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `${workflowName.toLowerCase().replace(/\s+/g,
'-')}.json`;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    URL.revokeObjectURL(url);
  };

  const handleRunPipeline = async () => {
    setIsRunning(true);
    setExecutionError(null);

    try {
      const executor = new PipelineExecutor(nodes, edges);
      const result = await executor.execute();

      if (result.error) {
        setExecutionError(result.error);
      } else {
        // Update nodes with their execution results
        result.nodeResults.forEach((output, nodeId) => {
          updateNodeResults(nodeId, output);
        });
      }
    } catch (error) {
      setExecutionError(error instanceof Error ? error.message : 'Pipeline
execution failed');
    } finally {
      setIsRunning(false);
    }
  };

  return (
    <div className="flex items-center justify-between px-4 py-2 border-b
border-gray-200 bg-white">
      <div className="flex items-center space-x-4">
        <input
          type="text"
          value={workflowName}
          onChange={(e) => setWorkflowName(e.target.value)}
          className="text-lg font-semibold text-gray-900 bg-transparent
border-none focus:outline-none focus:ring-2 focus:ring-blue-500
focus:ring-offset-2 rounded px-2 py-1"
        />
        <span className="text-sm text-gray-500">
          {saveStatus === 'saved' && 'All changes saved'}
          {saveStatus === 'saving' && 'Saving...'}
          {saveStatus === 'unsaved' && 'Unsaved changes'}
        </span>
        {executionError && (
          <span className="text-sm text-red-500 flex items-center">
            <AlertCircle className="w-4 h-4 mr-1" />
            {executionError}
          </span>
        )}
      </div>
      <div className="flex items-center space-x-2">
        <button
          onClick={() => clearWorkflow()}
          className="p-2 text-gray-600 hover:text-gray-900
hover:bg-gray-100 rounded-md"
          title="New Workflow"
        >
          <Plus className="w-4 h-4" />
        </button>
        <button
          onClick={handleSave}
          className="p-2 text-gray-600 hover:text-gray-900
hover:bg-gray-100 rounded-md"
          title="Save Workflow"
        >
          <Save className="w-4 h-4" />
        </button>
        <button
          onClick={handleExport}
          className="p-2 text-gray-600 hover:text-gray-900
hover:bg-gray-100 rounded-md"
          title="Export Workflow"
        >
          <Download className="w-4 h-4" />
        </button>
        <button
          className="p-2 text-gray-600 hover:text-gray-900
hover:bg-gray-100 rounded-md"
          title="Share Workflow"
        >
          <Share2 className="w-4 h-4" />
        </button>
        <button
          className="p-2 text-gray-600 hover:text-gray-900
hover:bg-gray-100 rounded-md"
          title="Settings"
        >
          <Settings className="w-4 h-4" />
        </button>
        <button
          onClick={handleRunPipeline}
          disabled={isRunning || nodes.length === 0}
          className={`px-4 py-1.5 bg-green-600 text-white text-sm
font-medium rounded-md hover:bg-green-700 flex items-center space-x-1.5 ${
            (isRunning || nodes.length === 0) ? 'opacity-75
cursor-not-allowed' : ''
          }`}
        >
          {isRunning ? (
            <Loader2 className="w-4 h-4 animate-spin" />
          ) : (
            <Play className="w-4 h-4" />
          )}
          <span>{isRunning ? 'Running...' : 'Run Pipeline'}</span>
        </button>
        <button
          onClick={handleDeploy}
          className="px-4 py-1.5 bg-blue-600 text-white text-sm
font-medium rounded-md hover:bg-blue-700 flex items-center space-x-1.5"
        >
          <Play className="w-4 h-4" />
          <span>Deploy Changes</span>
        </button>
      </div>
    </div>
  );
};
```

E:\refer code\FlowMindAi-main\src\components\Sidebar.tsx

```
import React, { useState } from 'react';
import { Brain, FileInput, FileOutput, GitFork, Search, MessageSquare,
Mail, Slack, FileText } from 'lucide-react';

const nodeTypes = [
  { type: 'input', label: 'Input Node', icon: FileInput },
  { type: 'ai-model', label: 'AI Model', icon: Brain },
  { type: 'output', label: 'Output Node', icon: FileOutput },
  { type: 'condition', label: 'Condition', icon: GitFork }
];

const templates = [
  {
    id: 'search-assistant',
    name: 'AI Search Assistant',
    description: 'Create an AI-powered search assistant that combines
multiple data sources',
    icon: Search,
    nodes: [
      { type: 'input', label: 'Query Input' },
      { type: 'ai-model', label: 'Search Processing' },
      { type: 'output', label: 'Search Results' }
    ]
  },
  {
    id: 'chatbot',
    name: 'Custom Chatbot',
    description: 'Build an interactive chatbot with customizable
responses',
    icon: MessageSquare,
    nodes: [
      { type: 'input', label: 'User Message' },
      { type: 'ai-model', label: 'Chat Processing' },
      { type: 'output', label: 'Bot Response' }
    ]
  },
  {
    id: 'email-assistant',
    name: 'Email Assistant',
    description: 'Create an AI email assistant for drafting and analyzing
emails',
    icon: Mail,
    nodes: [
      { type: 'input', label: 'Email Content' },
      { type: 'ai-model', label: 'Email Analysis' },
      { type: 'output', label: 'Suggestions' }
    ]
  }
];

export const Sidebar: React.FC = () => {
  const [activeTab, setActiveTab] = useState<'nodes' |
'templates'>('templates');

  const onDragStart = (event: React.DragEvent, nodeType: string) => {
    event.dataTransfer.setData('application/reactflow', nodeType);
    event.dataTransfer.effectAllowed = 'move';
  };

  return (
    <aside className="w-64 bg-white border-r border-gray-200">
      <div className="border-b border-gray-200">
        <nav className="flex -mb-px" aria-label="Tabs">
          <button
            onClick={() => setActiveTab('templates')}
            className={`w-1/2 py-4 px-1 text-center border-b-2 font-medium
text-sm ${
              activeTab === 'templates'
                ? 'border-blue-500 text-blue-600'
                : 'border-transparent text-gray-500 hover:text-gray-700
hover:border-gray-300'
            }`}
          >
            Templates
          </button>
          <button
            onClick={() => setActiveTab('nodes')}
            className={`w-1/2 py-4 px-1 text-center border-b-2 font-medium
text-sm ${
              activeTab === 'nodes'
                ? 'border-blue-500 text-blue-600'
                : 'border-transparent text-gray-500 hover:text-gray-700
hover:border-gray-300'
            }`}
          >
            Nodes
          </button>
        </nav>
      </div>

      <div className="p-4">
        {activeTab === 'templates' ? (
          <div className="space-y-4">
            <h2 className="text-lg font-semibold
text-gray-900">Templates</h2>
            <div className="space-y-3">
              {templates.map((template) => {
                const Icon = template.icon;
                return (
                  <div
                    key={template.id}
                    className="p-4 bg-white border border-gray-200
rounded-lg hover:border-blue-500 transition-colors cursor-pointer"
                  >
                    <div className="flex items-center space-x-3">
                      <div className="flex-shrink-0">
                        <Icon className="w-6 h-6 text-blue-600" />
                      </div>
                      <div>
                        <h3 className="text-sm font-medium
text-gray-900">{template.name}</h3>
                        <p className="text-xs text-gray-500
mt-1">{template.description}</p>
                      </div>
                    </div>
                  </div>
                );
              })}
            </div>
          </div>
        ) : (
          <div className="space-y-4">
            <h2 className="text-lg font-semibold text-gray-900">Node
Types</h2>
            <div className="space-y-2">
              {nodeTypes.map(({ type, label, icon: Icon }) => (
                <div
```

```
                  key={type}
                  className="flex items-center p-3 bg-gray-50 rounded-lg
cursor-move hover:bg-gray-100 transition-colors"
                  onDragStart={(e) => onDragStart(e, type)}
                  draggable
                >
                  <Icon className="w-5 h-5 mr-2 text-blue-600" />
                  <span className="text-sm text-gray-700">{label}</span>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
    </aside>
  );
};
```

E:\refer code\FlowMindAi-main\src\components\CustomEdge.tsx

```
import React from 'react';
import { EdgeProps, getBezierPath } from 'reactflow';
import { Trash2 } from 'lucide-react'; // Changed from react-feather to
lucide-react

const CustomEdge: React.FC<EdgeProps> = ({
  id,
  sourceX,
  sourceY,
  targetX,
  targetY,
  sourcePosition,
  targetPosition,
  style = {},
  markerEnd,
  data,
}) => {
  const [edgePath, labelX, labelY] = getBezierPath({
    sourceX,
    sourceY,
    sourcePosition,
    targetX,
    targetY,
    targetPosition,
  });

  const handleDelete = () => {
    if (data?.onDelete) {
      data.onDelete(id); // Call the delete function passed via `data`
    }
  };

  return (
    <>
      <path
        id={id}
        style={style}
        className="react-flow__edge-path"
        d={edgePath}
        markerEnd={markerEnd}
      />
      <foreignObject
        width={20}
        height={20}
        x={labelX - 10}
        y={labelY - 10}
        className="edge-button"
      >
        <div
          onClick={handleDelete}
          className="w-5 h-5 bg-white border border-gray-300 rounded-full
flex items-center justify-center cursor-pointer hover:bg-red-500
hover:text-white"
        >
          <Trash2 className="w-3 h-3" />
        </div>
      </foreignObject>
    </>
  );
};

export default CustomEdge;
```

E:\refer code\FlowMindAi-main\src\store\flowStore.ts

```
import { create } from 'zustand';
import {
  Connection,
  addEdge,
  applyNodeChanges,
  applyEdgeChanges,
  MarkerType,
} from 'reactflow';
import { FlowNode, FlowEdge, Template, NodeType } from '../types/flow';

interface FlowState {
  nodes: FlowNode[];
  edges: FlowEdge[];
  selectedNode: FlowNode | null;
  selectedTemplate: Template | null;
  onNodesChange: (changes: any) => void;
  onEdgesChange: (changes: any) => void;
  onConnect: (connection: Connection) => void;
  updateNodeParams: (nodeId: string, params: any) => void;
  updateNodeResults: (nodeId: string, results: any) => void;
  loadTemplate: (template: Template) => void;
  addNode: (type: NodeType, position: { x: number; y: number }) => void;
  removeNode: (nodeId: string) => void;
  setSelectedNode: (node: FlowNode | null) => void;
  clearWorkflow: () => void;
  setEdges: (updater: (edges: FlowEdge[]) => FlowEdge[]) => void; // Add
setEdges
  updateNodeData: (id: string, params: any) => void; // Add updateNodeData
}
```

```
export const useFlowStore = create<FlowState>((set, get) => ({
  nodes: [],
  edges: [],
  selectedNode: null,
  selectedTemplate: null,

  onNodesChange: (changes) => {
    set({
      nodes: applyNodeChanges(changes, get().nodes),
    });
  },

  onEdgesChange: (changes) => {
    set({
      edges: applyEdgeChanges(changes, get().edges),
    });
  },

  onConnect: (connection) => {
    const newEdge = {
      ...connection,
      type: 'smoothstep',
      markerEnd: { type: MarkerType.ArrowClosed },
      animated: true,
    };
    set({
      edges: addEdge(newEdge, get().edges),
    });
  },

  updateNodeParams: (nodeId, params) => {
    set({
      nodes: get().nodes.map((node) => {
        if (node.id === nodeId) {
          return {
            ...node,
            data: {
              ...node.data,
              params: {
```

```
              ...node.data.params,
              ...params,
            },
          },
        };
      }
      return node;
    }),
  });
},

updateNodeResults: (nodeId, results) => {
  set({
    nodes: get().nodes.map((node) => {
      if (node.id === nodeId) {
        return {
          ...node,
          data: {
            ...node.data,
            results,
          },
        };
      }
      return node;
    }),
  });
},

loadTemplate: (template) => {
  set({
    nodes: template.nodes,
    edges: template.edges,
    selectedTemplate: template,
  });
},

addNode: (type, position) => {
  const newNode: FlowNode = {
    id: `${type}-${Date.now()}`,
    type,
      position,
      data: {
        label: `${type} node`,
        type,
        params: {
          fieldName: 'input_1',
          type: 'Text'
        }
      },
    };
    set({
      nodes: [...get().nodes, newNode],
    });
  },

  removeNode: (nodeId) => {
    set({
      nodes: get().nodes.filter((node) => node.id !== nodeId),
      edges: get().edges.filter(
        (edge) => edge.source !== nodeId && edge.target !== nodeId
      ),
      selectedNode: get().selectedNode?.id === nodeId ? null :
get().selectedNode,
    });
  },

  setSelectedNode: (node) => {
    set({ selectedNode: node });
  },

  clearWorkflow: () => {
    set({
      nodes: [],
      edges: [],
      selectedNode: null,
      selectedTemplate: null,
    });
  },

  setEdges: (updater) => {
    set({
      edges: updater(get().edges),
    });
  },

  updateNodeData: (id, params) => {
    set((state) => ({
      nodes: state.nodes.map((node) =>
        node.id === id
          ? { ...node, data: { ...node.data, params: {
...node.data.params, ...params } } }
          : node
      ),
    }));
  },
}));
```

E:\refer code\FlowMindAi-main\src\store\themeStore.ts

```
import { create } from 'zustand';
import { persist } from 'zustand/middleware';

interface ThemeState {
  theme: 'light' | 'dark';
  toggleTheme: () => void;
}

export const useTheme = create<ThemeState>()(
  persist(
    (set) => ({
      theme: 'light',
      toggleTheme: () => set((state) => ({
        theme: state.theme === 'light' ? 'dark' : 'light'
      })),
    }),
    {
      name: 'theme-storage',
    }
  )
);
```

E:\refer code\FlowMindAi-main\src\types\flow.ts

```
import { Node, Edge } from 'reactflow';

export interface FlowNode extends Node {
  data: {
    label: string;
    type: string;
    params?: Record<string, any>;
    icon?: string;
    results?: any;
    paths?: Array<{
      id: string;
      clauses: Array<{
        id: string;
        inputField: string;
        operator: string;
        value: string;
      }>;
      logicalOperator: 'AND' | 'OR';
    }>;
  };
}

export type FlowEdge = Edge & {
  animated?: boolean;
};

export type NodeCategory =
  | 'general'
  | 'llms'
  | 'knowledge-base'
  | 'integrations'
  | 'data-loaders'
  | 'multi-modal'
  | 'logic'
  | 'ai-tools';

export type NodeType =
  | 'input'
  | 'web-extractor'
  | 'document-to-text'
  | 'google-search'
  | 'http-get'
  | 'file-upload'
  | 'output'
  | 'file-save'
  | 'note'
  | 'transform'
  | 'pipeline'
  | 'share'
  | 'api' // Added 'api' to NodeType
  | 'openai'
  | 'anthropic'
  | 'gemini'
  | 'cohere'
  | 'perplexity'
  | 'custom'
  | 'kb-reader'
  | 'kb-loader'
  | 'kb-search'
  | 'kb-sync'
  | 'mysql'
  | 'mongodb'
  | 'github'
  | 'airtable'
  | 'notion'
  | 'hubspot'
  | 'gmail'
  | 'outlook'
  | 'discord'
  | 'google-drive'
  | 'onedrive'
  | 'google-docs'
```

```typescript
  | 'slack'
  | 'teams'
  | 'make-webhook'
  | 'zapier-webhook'
  | 'file-loader'
  | 'api-loader'
  | 'csv-loader'
  | 'url-loader'
  | 'wiki-loader'
  | 'youtube-loader'
  | 'arxiv-loader'
  | 'rss-loader'
  | 'audio-processor'
  | 'image-processor'
  | 'condition'
  | 'merge'
  | 'time'
  | 'ttsql'
  | 'chat-memory'
  | 'data-collector'
  | 'chat-file-reader'
  | 'outlook-trigger'
  | 'gmail-trigger'
  | 'text-processor'
  | 'json-handler'
  | 'file-transformer'
  | 'ai-task-executor'
  | 'notification-node'
  | 'crm-enricher';

export type DatabaseAction =
  | 'natural-language-query'
  | 'raw-sql-query'
  | 'natural-language-agent'
  | 'mongodb-find'
  | 'mongodb-find-one'
  | 'mongodb-aggregate'
  | 'nl-query'
  | 'nl-aggregation'
  | 'read-file'
  | 'create-pull-request'
  | 'update-pull-request';

export type ProductivityAction =
  | 'read-table'
  | 'add-new-record'
  | 'column-list-writer'
  | 'update-records'
  | 'write-to-database'
  | 'create-new-page'
  | 'create-new-block'
  | 'database-reader'
  | 'database-updater'
  | 'fetch-contacts'
  | 'fetch-companies'
  | 'fetch-deals'
  | 'fetch-tickets'
  | 'fetch-notes'
  | 'fetch-calls'
  | 'create-email-draft'
  | 'send-email'
  | 'draft-reply'
  | 'send-reply'
  | 'send-message'
  | 'read-file'
  | 'save-file'
  | 'add-file';
```

E:\refer code\FlowMindAi-main\src\utils\nodeExecutors.ts

```
import { FlowNode, FlowEdge } from '../types/flow';

interface NodeInput {
  [key: string]: any;
}

interface NodeOutput {
  [key: string]: any;
```

```
}

interface ExecutionContext {
  nodes: FlowNode[];
  edges: FlowEdge[];
  nodeOutputs: Map<string, NodeOutput>;
}

async function processMultiInput(input: string, resume: string, yy:
string): Promise<string> {
  await new Promise(resolve => setTimeout(resolve, 1000));
  return `Processed inputs:
- Input: ${input}
- Resume: ${resume}
- YY: ${yy}`;
}

export async function executeNode(
  node: FlowNode,
  inputs: NodeInput,
  context: ExecutionContext
): Promise<NodeOutput> {
  switch (node.type) {
    case 'input':
      return {
        input_1: inputs.input || '',
        resume: inputs.resume || '',
        yy: inputs.yy || ''
      };

    case 'output':
      return {
        output: inputs.input || 'No output',
        type: node.data.params?.type || 'Text'
      };

    case 'text':
      if (inputs.input) {
        return { output: inputs.input };
      }
      return { output: node.data.params?.text || 'Sample text' };

    case 'transform':
      return {
        output: transformData(inputs.input || inputs.output,
node.data.params?.transformation)
      };

    case 'pipeline':
      const pipelineResult = await processPipelineInputs(inputs,
node.data.params?.batchMode);
      return { output: pipelineResult };

    case 'file-save':
      return {
        name: node.data.params?.fileName || 'output.txt',
        files: inputs.files || inputs.input || 'No content'
      };

    case 'note':
      return { note: node.data.params?.note || '' };

    // LLM Nodes
    case 'openai':
    case 'google':
    case 'anthropic':
    case 'azure':
    case 'aws':
    case 'perplexity':
      return {
        response: await simulateLLMResponse(
          node.type,
          inputs.system || '',
          inputs.prompt || ''
        )
      };

    default:
      throw new Error(`Unsupported node type: ${node.type}`);
  }
```

```
}

async function simulateLLMResponse(provider: string, system: string,
prompt: string): Promise<string> {
  await new Promise(resolve => setTimeout(resolve, 1000));
  return `${provider.toUpperCase()} response to:
System: ${system}
Prompt: ${prompt}`;
}

async function processPipelineInputs(inputs: NodeInput, batchMode?:
boolean): Promise<string> {
  await new Promise(resolve => setTimeout(resolve, 500));
  return `Pipeline processed${batchMode ? ' (Batch Mode)' : ''}:
${JSON.stringify(inputs)}`;
}

function transformData(data: any, transformation?: string): any {
  if (!data) return data;
  if (!transformation) return data;

  try {
    // Simple transformation examples
    switch (transformation) {
      case 'uppercase':
        return typeof data === 'string' ? data.toUpperCase() : data;
      case 'lowercase':
        return typeof data === 'string' ? data.toLowerCase() : data;
      case 'number':
        return Number(data);
      case 'string':
        return String(data);
      case 'json':
        return typeof data === 'string' ? JSON.parse(data) :
JSON.stringify(data);
      default:
        return data;
    }
  } catch (error) {
    console.error('Transformation error:', error);
    return data;
  }
}
```

E:\refer code\FlowMindAi-main\src\utils\pipelineExecutor.ts

```
import { FlowNode, FlowEdge } from '../types/flow';
import { executeNode } from './nodeExecutors';

interface ExecutionResult {
  nodeResults: Map<string, any>;
  error?: string;
}

export class PipelineExecutor {
  private nodes: FlowNode[];
  private edges: FlowEdge[];
  private nodeOutputs: Map<string, any>;
  private executionOrder: FlowNode[];

  constructor(nodes: FlowNode[], edges: FlowEdge[]) {
    this.nodes = nodes;
    this.edges = edges;
    this.nodeOutputs = new Map();
    this.executionOrder = this.calculateExecutionOrder();
  }

  private calculateExecutionOrder(): FlowNode[] {
    const visited = new Set<string>();
    const order: FlowNode[] = [];
    const inProgress = new Set<string>();

    const visit = (nodeId: string) => {
      if (inProgress.has(nodeId)) {
        throw new Error('Circular dependency detected in pipeline');
      }
      if (visited.has(nodeId)) return;
      inProgress.add(nodeId);

      // Get all outgoing edges from this node
      const outgoingEdges = this.edges.filter(e => e.source === nodeId);
      for (const edge of outgoingEdges) {
        visit(edge.target);
      }

      inProgress.delete(nodeId);
      visited.add(nodeId);
      const node = this.nodes.find(n => n.id === nodeId);
      if (node) order.unshift(node);
    };

    // Start with input nodes (nodes with no incoming edges)
    const inputNodes = this.nodes.filter(node =>
      !this.edges.some(edge => edge.target === node.id)
    );

    for (const node of inputNodes) {
      visit(node.id);
    }

    return order;
  }

  private getNodeInputs(nodeId: string): any {
    const inputs: any = {};
    const incomingEdges = this.edges.filter(e => e.target === nodeId);

    for (const edge of incomingEdges) {
      const sourceOutput = this.nodeOutputs.get(edge.source);
      if (sourceOutput) {
        inputs[edge.targetHandle || 'input'] =
sourceOutput[edge.sourceHandle || 'output'];
      }
    }

    return inputs;
  }
  async execute(): Promise<ExecutionResult> {
    this.nodeOutputs.clear();
    const context = {
      nodes: this.nodes,
      edges: this.edges,
      nodeOutputs: this.nodeOutputs
    };

    try {
      for (const node of this.executionOrder) {
        const inputs = this.getNodeInputs(node.id);
        const output = await executeNode(node, inputs, context);
        this.nodeOutputs.set(node.id, output);
      }

      return {
        nodeResults: this.nodeOutputs
      };
    } catch (error) {
      return {
        nodeResults: this.nodeOutputs,
        error: error instanceof Error ? error.message : 'Unknown error
occurred'
      };
    }
  }
}
```

E:\refer code\FlowMindAi-main\src\App.tsx

```
import React, { useState } from 'react';
import { Navigation } from './components/Navigation';
import { NodePanel } from './components/NodePanel';
import { Toolbar } from './components/Toolbar';
import { FlowCanvas } from './components/FlowCanvas';
import { NodeCategory } from './types/flow';
import { useTheme } from './store/themeStore';

function App() {
  const [activeCategory, setActiveCategory] =
useState<NodeCategory>('general');
  const [searchQuery, setSearchQuery] = useState('');
  const { theme } = useTheme();

  return (
    <div className={`flex flex-col h-screen ${
      theme === 'dark' ? 'bg-gray-900' : 'bg-gray-50'
    }`}>
      <Navigation
        activeCategory={activeCategory}
        onCategoryChange={setActiveCategory}
        onSearch={setSearchQuery}
      />
      <Toolbar />
      <main className="flex flex-1 overflow-hidden">
        <aside className={`w-64 border-r overflow-y-auto ${
          theme === 'dark'
            ? 'border-gray-700 bg-gray-800'
            : 'border-gray-200 bg-white'
        }`}>
          <NodePanel
            category={activeCategory}
            searchQuery={searchQuery}
            onDragStart={(event: React.DragEvent, nodeType: string) => {
              event.dataTransfer.setData('application/reactflow',
nodeType);
              event.dataTransfer.effectAllowed = 'move';
            }}
          />
        </aside>
        <FlowCanvas />
      </main>
    </div>
  );
}
```

```
export default App;
```

E:\refer code\FlowMindAi-main\src\index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Custom styles for the edge button */
.edge-button {
  pointer-events: all;
}

.edge-button div {
  display: flex;
  align-items: center;
  justify-content: center;
  background-color: white;
  border-radius: 50%;
  border: 1px solid #ccc;
  width: 20px;
  height: 20px;
  cursor: pointer;
  transition: all 0.2s ease;
}

.edge-button div:hover {
  background-color: #f87171; /* Red background on hover */
  color: white;
  border-color: #f87171;
}
```

E:\refer code\FlowMindAi-main\src\main.tsx

```

```

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import App from './App.tsx';
import './index.css';

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

E:\refer code\FlowMindAi-main\src\components\nodes\BaseNode.tsx

```
import React, { memo } from 'react';
import { Handle, Position, NodeProps } from 'reactflow';
import { Trash2 } from 'lucide-react';
import { useFlowStore } from '../../store/flowStore';
// Change the import path to the correct location
import RenderNodeContent from './types/RenderNodeContent';


interface BaseNodeProps extends NodeProps {
  icon?: React.ReactNode;
  inputs?: string[];
  outputs?: string[];
}

export const BaseNode = memo(({ data, id, selected, icon: Icon, inputs =
[], outputs = [] }: BaseNodeProps) => {
  const removeNode = useFlowStore((state) => state.removeNode);
  const updateNodeData = useFlowStore((state) => state.updateNodeParams);

  const renderInputHandles = () => {
    return inputs.map((input, index) => (
      <Handle
        key={`input-${index}`}
        type="target"
        position={Position.Left}
        id={input}
        className="w-3 h-3 bg-gray-400 border-2 border-white rounded-full"
        style={{ left: -6, top: `${50 + index * 20}%`, transform:
'translateY(-50%)' }}
      />
    ));
  };

  const renderOutputHandles = () => (
    <Handle
      type="source"
      position={Position.Right}
      id="output"
      className="w-3 h-3 bg-blue-500 border-2 border-white rounded-full"
      style={{ right: -6, top: '50%' }}
    />
  );

  return (
    <div
      className={`relative bg-white rounded-lg shadow-md border-2 ${
        selected ? 'border-blue-500' : 'border-gray-200'
      }`}
    >
      <div className="p-3">
        <div className="flex items-center justify-between mb-2">
          <div className="flex items-center space-x-2">
            {Icon && <div className="text-blue-600">{Icon}</div>}
            <div className="text-sm font-medium
text-gray-900">{data.label}</div>
          </div>
          <button
            onClick={() => removeNode(id)}
            className="text-gray-400 hover:text-red-500 transition-colors"
          >
            <Trash2 className="w-4 h-4" />
          </button>
        </div>
        {/* Render Node Content */}
        <RenderNodeContent
          type={data.type}
          data={data}
          id={id}
          updateNodeData={updateNodeData}
          removeNode={removeNode}
        />
      </div>

      {renderInputHandles()}
      {renderOutputHandles()}
    </div>
  );
});

BaseNode.displayName = 'BaseNode';

// Removed duplicate React import

interface LocalRenderNodeContentProps {
  type: string;
  data: any;
  id: string;
  updateNodeData: (id: string, data: any) => void;
  removeNode: (id: string) => void;
}

const LocalRenderNodeContent: React.FC<LocalRenderNodeContentProps> = ({
type, data, id, updateNodeData, removeNode }) => {
  return (
    <div>
      <p>Type: {type}</p>
      <p>Data: {JSON.stringify(data)}</p>
    </div>
  );
};

interface LLMNodeProps {
  id: string;
  data: {
    type: string;
    params?: {
      system?: string;
      prompt?: string;
      model?: string;
      usePersonalKey?: boolean;
      apiKey?: string;
      finetunedModel?: string;
      showSettings?: boolean;
    };
  };
  updateNodeData?: (id: string, params: any) => void;
  removeNode?: (id: string) => void;
}

export default LocalRenderNodeContent;
```

E:\refer code\FlowMindAi-main\src\components\nodes\index.tsx

```
import React from 'react';
import { NodeTypes } from 'reactflow';
import { BaseNode } from './BaseNode';
import { ConditionNode } from './ConditionNode';
import PipelineNode from './types/PipelineNode'; // Update the import path
import ShareNode from './Share';
import FileSaveNode from './FileSave';
import TransformNode from './types/TransformNode';
import InputNode from './types/InputNode';
import OutputNode from './types/OutputNode';
import DocumentToTextNode from './types/DocumentToTextNode';
import LLMNode from './types/LLMNode';
import TextNode from './types/TextNode';  // Add this import
import {
  FileText,
  Send,
  Zap,
  FileInput,
  Save,
  StickyNote,
  Database,
  Github,
  Table2,
  FileText as NotionIcon,
  Building2,
  Mail,
  MessageSquare,
  FolderOpen,
  FileEdit,
  Webhook,
  Slack as SlackIcon,
  MessageSquare as TeamsIcon,
  Globe,
  Youtube,
  FileSearch,
  Search,
  Download,
  Upload,
  Cloud,
  BrainCircuit,
  Bot,
  Cpu,
  Settings,
  BookOpen,
  RefreshCw,
  Baseline as PipelineIcon,
  Tractor as Transform,
  GitBranch,
  Clock,
  Globe as Api,
  FileSpreadsheet,
  Link,
  // Wiki, // Removed as it is not exported by 'lucide-react'
  Newspaper,
  Music,
  Camera,
  MessageCircle,
  FileJson,
  FileCode,
  Cog,
  Bell,
  Users,
  MailCheck,
  MailWarning,
  File,
  Share2

} from 'lucide-react';

// Add after imports
interface NodeProps {
  id: string;
  data: {
    params?: {
      [key: string]: any;
    };
  };
  updateNodeData?: (id: string, params: any) => void; // Optional property
  removeNode?: (id: string) => void;
}

const createNode = (icon: React.ReactNode, inputs?: string[], outputs?:
string[]) => {
  return (props: any) => (
    <BaseNode {...props} icon={icon} inputs={inputs} outputs={outputs} />
  );
};

const nodeTypeMap: Record<string, React.FC<any>> = {
  // ...existing nodes...
  'transform': TransformNode,       // Changed from 'transform-node'
  'input': InputNode,               // Changed from 'InputNode'
  'output': OutputNode,             // Changed from 'OutputNode'
  'pipeline': PipelineNode,         // Changed from 'PipelineNode'
  // ...rest of the nodes...
};
```

```
export const nodeTypes: NodeTypes = {
  // Core Settings nodes
  input: (props) => <InputNode {...props as NodeProps & { updateNodeData:
(data: any) => void; removeNode: () => void }}
updateNodeData={props.updateNodeData} removeNode={props.removeNode} />,
  output: (props) => <OutputNode {...props as NodeProps & {
updateNodeData: (id: string, params: any) => void; removeNode: (id:
string) => void }} updateNodeData={props.updateNodeData}
removeNode={props.removeNode} />,
  text: (props) => (
    <TextNode
      {...(props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      })}
      updateNodeData={props.updateNodeData}
      removeNode={props.removeNode ?? (() => {})}
    />
  ),
  //'web-extractor': createNode(<Globe className="w-4 h-4" />, ['url'],
['content']),
  //'youtube-transcript': createNode(<Youtube className="w-4 h-4" />,
['url'], ['transcript']),
  'document-to-text': (props) => (
    <DocumentToTextNode
      {...props as NodeProps & {
        updateNodeData?: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      }}
      updateNodeData={props.updateNodeData ?? (() => {})}
      removeNode={props.removeNode}
    />
  ),
  //'google-search': createNode(<Search className="w-4 h-4" />, ['query'],
['results']),
 // 'http-get': createNode(<Download className="w-4 h-4" />, ['url'],
['response']),
 // 'file-upload': createNode(<Upload className="w-4 h-4" />, [],
['file']),
  'file-save': (props) => <FileSaveNode {...props}
removeNode={props.removeNode} />, // Pass removeNode
  note: createNode(<StickyNote className="w-4 h-4" />),
  transform: (props) => <TransformNode {...props} />,
  condition: ConditionNode,
 // Use PipelineNode
  share: (props) => <ShareNode {...props} removeNode={props.removeNode}
/>, // Pass removeNode
  // LLM nodes
  openai: (props) => (
    <LLMNode
      {...props}
      id={props.id}
      data={props.data}
      updateNodeData={(id, params) => {
        if (typeof props.updateNodeData === 'function') {
          props.updateNodeData(id, params);
        }
      }}
      removeNode={props.removeNode}
    />
  ),
  anthropic: (props) => (
    <LLMNode
      {...props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      }}
      updateNodeData={props.updateNodeData}
      removeNode={props.removeNode}
    />
  ),
  gemini: (props) => (
    <LLMNode
      {...props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      }}
      updateNodeData={props.updateNodeData}
      removeNode={props.removeNode}
    />
  ),
  cohere: (props) => (
    <LLMNode
      {...props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      }}
      updateNodeData={props.updateNodeData}
      removeNode={props.removeNode}
    />
  ),
  perplexity: (props) => (
    <LLMNode
      {...props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      }}
      updateNodeData={props.updateNodeData}
      removeNode={props.removeNode}
    />
  ),
  custom: (props) => (
    <LLMNode
      {...props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      }}
      updateNodeData={props.updateNodeData}
      removeNode={props.removeNode}
    />
  ),
  llm: (props) => (
    <LLMNode
      {...props}
      updateNodeData={props.updateNodeData} // Ensure this is passed
      removeNode={props.removeNode} // Ensure this is passed
    />
  ),
  // Knowledge Base nodes
  'kb-reader': createNode(<BookOpen className="w-4 h-4" />, ['query'],
['results']),
  'kb-loader': createNode(<Upload className="w-4 h-4" />, ['documents'],
['status']),
  'kb-search': createNode(<Search className="w-4 h-4" />, ['query',
'documents'], ['result']),
  'kb-sync': createNode(<RefreshCw className="w-4 h-4" />, ['source'],
['status']),

  // Database nodes
  mysql: createNode(<Database className="w-4 h-4" />, ['query'],
['output', 'complete']),
  mongodb: createNode(<Database className="w-4 h-4" />, ['query'],
['output', 'complete']),
  github: createNode(<Github className="w-4 h-4" />, ['input'],
['complete']),

  // Productivity tool nodes
  airtable: createNode(<Table2 className="w-4 h-4" />, ['input'],
['complete']),
  notion: createNode(<NotionIcon className="w-4 h-4" />, ['input'],
['complete']),
  hubspot: createNode(<Building2 className="w-4 h-4" />, ['input'],
['complete']),
  gmail: createNode(<Mail className="w-4 h-4" />, ['input'],
['complete']),
  outlook: createNode(<Mail className="w-4 h-4" />, ['input'],
['complete']),
  discord: createNode(<MessageSquare className="w-4 h-4" />, ['message'],
['complete']),
  'google-drive': createNode(<FolderOpen className="w-4 h-4" />, ['file'],
['complete']),
  onedrive: createNode(<FolderOpen className="w-4 h-4" />, ['file'],
['complete']),
  'google-docs': createNode(<FileEdit className="w-4 h-4" />, ['input'],
['complete']),

  // Communication nodes
  slack: createNode(<SlackIcon className="w-4 h-4" />, ['message'],
['complete']),
  teams: createNode(<TeamsIcon className="w-4 h-4" />, ['message'],
['complete']),

  // Webhook nodes
  'make-webhook': createNode(<Webhook className="w-4 h-4" />, ['url',
'payload'], ['complete']),
  'zapier-webhook': createNode(<Webhook className="w-4 h-4" />, ['url',
'payload'], ['complete']),

  // Logic nodes
  time: createNode(<Clock className="w-4 h-4" />, ['input'], ['output']),
  ttsql: createNode(<Database className="w-4 h-4" />, ['query'],
['output']),

  // New nodes
  api: createNode(<Api className="w-4 h-4" />, ['url'], ['response']),
  csv: createNode(<FileSpreadsheet className="w-4 h-4" />, ['file'],
['data']),
  url: createNode(<Link className="w-4 h-4" />, ['url'], ['content']),
 // wiki: createNode(<Wiki className="w-4 h-4" />, ['query'],
['content']), // Removed as 'Wiki' is not available
  youtube: createNode(<Youtube className="w-4 h-4" />, ['url'],
['transcript']),
  arxiv: createNode(<FileText className="w-4 h-4" />, ['query'],
['papers']),
  rss: createNode(<Newspaper className="w-4 h-4" />, ['url'], ['feed']),
  'web-search': createNode(<Search className="w-4 h-4" />, ['query'],
['results']),
  'chat-memory': createNode(<MessageCircle className="w-4 h-4" />,
['input'], ['memory']),
  'data-collector': createNode(<Database className="w-4 h-4" />,
['input'], ['data']),
  'chat-file-reader': createNode(<FileText className="w-4 h-4" />,
['file'], ['content']),
  'outlook-trigger': createNode(<MailCheck className="w-4 h-4" />,
['event'], ['output']),
  'gmail-trigger': createNode(<MailWarning className="w-4 h-4" />,
['event'], ['output']),
  'text-processor': createNode(<FileText className="w-4 h-4" />,
['input'], ['output']),
  'json-handler': createNode(<FileJson className="w-4 h-4" />, ['input'],
['output']),
  'file-transformer': createNode(<FileCode className="w-4 h-4" />,
['file'], ['transformed']),
  'ai-task-executor': createNode(<Cog className="w-4 h-4" />, ['task'],
['result']),
  'notification-node': createNode(<Bell className="w-4 h-4" />, ['input'],
['notification']),
  'crm-enricher': createNode(<Users className="w-4 h-4" />, ['data'],
['enriched']),
  audio: createNode(<Music className="w-4 h-4" />, ['file'], ['audio']),
  image: createNode(<Camera className="w-4 h-4" />, ['file'], ['image']),
  'merge': createNode(<GitBranch className="w-4 h-4" />, ['input'],
['output']),

  // Multi-modal nodes
  'audio-processor': createNode(<Music className="w-4 h-4" />, ['file'],
['audio']),
  'image-processor': createNode(<Camera className="w-4 h-4" />, ['file'],
['image']),
  'file-loader': createNode(<File className="w-4 h-4" />, ['file'],
['output']),
  'api-loader': createNode(<Api className="w-4 h-4" />, ['url'],
['response']),
  'csv-loader': createNode(<FileSpreadsheet className="w-4 h-4" />,
['file'], ['data']),
  'url-loader': createNode(<Link className="w-4 h-4" />, ['url'],
['content']),
 // 'wiki-loader': createNode(<Wiki className="w-4 h-4" />, ['query'],
['content']),
  'youtube-loader': createNode(<Youtube className="w-4 h-4" />, ['url'],
['transcript']),
  'arxiv-loader': createNode(<FileText className="w-4 h-4" />, ['query'],
['papers']),
  'rss-loader': createNode(<Newspaper className="w-4 h-4" />, ['url'],
['feed']),
  // 'wiki-loader': createNode(<Wiki className="w-4 h-4" />, ['query'],
['content']),
  pipeline: (props) => (
    <PipelineNode
      {...(props as NodeProps & {
        updateNodeData: (id: string, params: any) => void;
        removeNode: (id: string) => void;
      })}
      updateNodeData={props.updateNodeData ?? (() =>
console.error('updateNodeData is not defined'))}
      removeNode={props.removeNode ?? (() => console.error('removeNode is
not defined'))}
    />
  ),
};
```