```
import React, { useEffect, useState, useCallback, memo } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { RootState } from '../store';
import { fetchTreeById, updateTreeNodes } from '../features/trees/treeSlice';
import ReactFlow, {
  Background,
  Controls,
  Node,
  Edge,
  Connection,
  addEdge,
  useNodesState,
  useEdgesState,
  Panel,
  NodeMouseHandler,
  ReactFlowInstance,
  ConnectionMode,
  Handle,
  Position
} from 'reactflow';
import 'reactflow/dist/style.css';
import './TreeEdit.css';
import YoutubeSearch from '../components/YoutubeSearch';
```

```
interface NodeData {
  label: string;
  videoTitle?: string;
  videoUrl?: string;
  description?: string;
  likes?: number;
  comments?: Array<{
    id: string;
    text: string;
    author: string;
    createdAt: string;
  }>;
}

const initialNodes: Node[] = [];

const getYoutubeID = (url: string) => {
  const regExp =
/^.*((youtu.be₩/)|(v₩/)|(₩/u₩/₩w₩/)|(embed₩/)|(watch₩?))₩??v?=?([^#&?]*).*/;
  const match = url.match(regExp);
  return (match && match[7].length === 11) ? match[7] : null;
};

const nodeDefaults = {
  style: {
```

```
    borderRadius: '8px',
    border: '1px solid #ddd',
    padding: '10px',
    cursor: 'pointer',
    background: 'white',
    boxShadow: '0 2px 5px rgba(0, 0, 0, 0.1)',
  }
};

const CustomNode = memo(({ data, id, selected }: { data: NodeData; id: string; selected: boolean }) => {
  const videoId = data.videoUrl ? getYoutubeID(data.videoUrl) : null;

  // 디버깅 로그 추가
  useEffect(() => {
    console.log(`[CustomNode ${id}] 렌더링:`, { data, videoId, dataVideoUrl: data.videoUrl }); // data.videoUrl 추가
  }, [data, videoId, id]);

  return (
    <div style={{
      ...nodeDefaults.style,
      border: selected ? '2px solid #3b82f6' : '1px solid #ddd',
      boxShadow: selected ? '0 0 0 2px rgba(59, 130, 246, 0.5)' : nodeDefaults.style.boxShadow,
      position: 'relative',
```

```
    padding: '15px'
  }}>
    <div className="flex items-center justify-between">
      <div className="font-medium">{data.videoTitle || data.label}</div>
      {data.likes !== undefined && (
        <div className="text-sm text-gray-500">
          ❤️ {data.likes}
        </div>
      )}
    </div>
    {videoId && (
      <div className="mt-2 rounded overflow-hidden">
        <img
          src={`https://img.youtube.com/vi/${videoId}/mqdefault.jpg`}
          alt="YouTube Thumbnail"
          className="w-full h-auto"
        />
      </div>
    )}
    <Handle
      type="source"
      position={Position.Right}
      id={`right-${id}`}
      isConnectable={true}
      style={{ background: '#555', width: '15px', height: '15px', right: '-8px' }}
```

```
        />
        <Handle
          type="target"
          position={Position.Left}
          id={`left-${id}`}
          isConnectable={true}
          style={{ background: '#555', width: '15px', height: '15px', left: '-8px' }}
        />
        <Handle
          type="source"
          position={Position.Top}
          id={`top-${id}`}
          isConnectable={true}
          style={{ background: '#555', width: '15px', height: '15px', top: '-8px' }}
        />
        <Handle
          type="target"
          position={Position.Bottom}
          id={`bottom-${id}`}
          isConnectable={true}
          style={{ background: '#555', width: '15px', height: '15px', bottom: '-8px' }}
        />
      </div>
    );
});
```

```
interface NodeDetailModalProps {
  node: Node<NodeData>;
  onClose: () => void;
  onSave: (data: NodeData) => void;
  isEdit: boolean;
  onToggleEdit: () => void;
}

// 유튜브 메타데이터 가져오기 함수
const fetchYoutubeMetadata = async (videoId: string) => {
  try {
    // API 키가 없는 경우, 클라이언트 측에서만 썸네일 URL을 생성합니다
    return {
      title: '', // API 연동 시 response.data.items[0].snippet.title
      description: '', // API 연동 시 response.data.items[0].snippet.description
      thumbnailUrl: `https://img.youtube.com/vi/${videoId}/mqdefault.jpg`
    };
  } catch (error) {
    console.error('Failed to fetch YouTube metadata:', error);
    return null;
  }
};

const NodeDetailModal: React.FC<NodeDetailModalProps> = ({
```

```
  node,
  onClose,
  onSave,
  isEdit,
  onToggleEdit,
}) => {
  const [formData, setFormData] = useState<NodeData>({
    ...node.data,
  });
  const [videoId, setVideoId] = useState<string | null>(
    node.data.videoUrl ? getYoutubeID(node.data.videoUrl) : null
  );
  const [newComment, setNewComment] = useState('');
  const [showYoutubeSearch, setShowYoutubeSearch] = useState(false);
  const [youtubeUrl, setYoutubeUrl] = useState('');
  const [isDragging, setIsDragging] = useState(false);

  const handleDragOver = (e: React.DragEvent) => {
    e.preventDefault();
    setIsDragging(true);
  };

  const handleDragLeave = (e: React.DragEvent) => {
    e.preventDefault();
    setIsDragging(false);
```

```
};

const handleDrop = async (e: React.DragEvent) => {
  e.preventDefault();
  setIsDragging(false);

  let url = e.dataTransfer.getData('text');
  console.log('Dropped URL:', url); // 디버깅용

  if (url) {
    const videoId = getYoutubeID(url);
    if (videoId) {
      const fullUrl = `https://www.youtube.com/watch?v=${videoId}`;
      console.log('Video ID:', videoId); // 디버깅용
      setVideoId(videoId);
      setFormData(prev => ({
        ...prev,
        videoUrl: fullUrl,
      }));

      // 메타데이터 가져오기
      const metadata = await fetchYoutubeMetadata(videoId);
      if (metadata) {
        setFormData(prev => ({
          ...prev,
```

```
            videoTitle: prev.videoTitle || metadata.title,
            description: prev.description || metadata.description
          }));
        }
      }
    }
  };

  const handleUrlChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
    const url = e.target.value;
    const newVideoId = getYoutubeID(url);
    setVideoId(newVideoId);
    setFormData(prev => ({ ...prev, videoUrl: url }));

    // 새 비디오 ID가 있고, 제목과 설명이 비어있는 경우에만 메타데이터 가져오기
    if (newVideoId && (!formData.videoTitle || !formData.description)) {
      const metadata = await fetchYoutubeMetadata(newVideoId);
      if (metadata) {
        setFormData(prev => ({
          ...prev,
          videoTitle: prev.videoTitle || metadata.title,
          description: prev.description || metadata.description
        }));
      }
    }
```

```
};

const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    onSave(formData);
    onToggleEdit();
};

const handleAddComment = () => {
    if (!newComment.trim()) return;

    const comment = {
        id: Date.now().toString(),
        text: newComment,
        author: '사용자',
        createdAt: new Date().toLocaleString()
    };

    setFormData(prev => ({
        ...prev,
        comments: [...(prev.comments || []), comment]
    }));
    setNewComment('');
};
```

```
  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
      <div className={`flex w-full h-full p-4 ${showYoutubeSearch ? 'justify-between' : 'justify-center'}`}>
        {/* 선택창 */}
        <div
          className={`bg-white rounded-lg p-6 overflow-y-auto transition-all duration-300 ${
            showYoutubeSearch ? 'w-1/2 max-h-[40vh]' : 'max-w-2xl w-full max-h-[40vh]'
          }`}
        >
          <div className="flex justify-between items-center mb-4">
            <h2 className="text-xl font-bold">노드 {isEdit ? '수정' : '상세정보'}</h2>
            <button onClick={onClose} className="text-gray-500 hover:text-gray-700">
              ×
            </button>
          </div>

          {isEdit ? (
            <form onSubmit={handleSubmit}>
              <div className="space-y-4">
                <div>
                  <label className="block text-sm font-medium text-gray-700 mb-1">
                    제목
                  </label>
```

```
                    <input
                      type="text"
                      value={formData.videoTitle || formData.label}
                      onChange={(e) =>
                        setFormData((prev) => ({
                          ...prev,
                          videoTitle: e.target.value,
                          label: e.target.value,
                        }))
                      }
                      className="w-full p-2 border rounded"
                    />
                  </div>

                  <div
                    className={`border-2 border-dashed p-4 rounded-lg transition-
colors ${
                      isDragging ? 'border-blue-500 bg-blue-50' : 'border-gray-300'
                    }`}
                    onDragOver={handleDragOver}
                    onDragLeave={handleDragLeave}
                    onDrop={handleDrop}
                  >
                    <label className="block text-sm font-medium text-gray-700 mb-1">
                      유튜브 URL
```

```
                    </label>
                    <div className="flex gap-2">
                      <input
                        type="text"
                        value={formData.videoUrl || ''}
                        onChange={handleUrlChange}
                        placeholder="유튜브 URL을 입력하거나 썸네일을 이곳에 끌어다 놓으세요"
                        className="flex-1 p-2 border rounded"
                        id="youtube-url-input"
                      />
                      <button
                        type="button"
                        onClick={() => setShowYoutubeSearch(true)}
                        className="px-4 py-2 bg-red-500 text-white rounded hover:bg-red-600"
                      >
                        유튜브 검색
                      </button>
                    </div>
                    {isDragging && (
                      <div className="mt-2 text-blue-500 text-sm">
                        여기에 썸네일을 놓으세요
                      </div>
                    )}
                    {videoId && (
```

```
                    <div className="mt-2">
                      <img
                        src={`https://img.youtube.com/vi/${videoId}/mqdefault.jpg`}
                        alt="YouTube Thumbnail"
                        className="rounded w-full h-auto"
                      />
                    </div>
                  )}
                </div>

                <div>
                  <label className="block text-sm font-medium text-gray-700">설명
</label>
                  <textarea
                    value={formData.description || ''}
                    onChange={(e) => setFormData(prev => ({ ...prev, description:
e.target.value }))}
                    className="mt-1 block w-full rounded-md border-gray-300
shadow-sm focus:border-blue-500 focus:ring-blue-500"
                    rows={3}
                  />
                </div>

                <div className="flex justify-end gap-2 mt-4">
                  <button
                    type="button"
```

```
                onClick={onClose}
                className="px-4 py-2 bg-gray-100 text-gray-700 rounded hover:bg-gray-200"
              >
                취소
              </button>
              <button
                type="submit"
                className="px-4 py-2 bg-blue-500 text-white rounded hover:bg-blue-600"
              >
                저장
              </button>
            </div>
          </div>
        </form>
      ) : (
        <div className="space-y-4">
          <div>
            <h3 className="text-lg font-medium">{node.data.videoTitle || node.data.label}</h3>
          </div>
          {videoId && (
            <div className="aspect-video w-full">
              <iframe
                width="100%"
```

```
                    height="100%"
                    src={`https://www.youtube.com/embed/${videoId}`}
                    title="YouTube video player"
                    frameBorder="0"
                    allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture"
                    allowFullScreen
                  ></iframe>
                </div>
              )}
              {node.data.description && (
                <p className="text-gray-600 whitespace-pre-wrap">{node.data.description}</p>
              )}
              <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                  <button className="flex items-center gap-1 text-gray-600 hover:text-red-500">
                    <span>❤️</span>
                    <span>{node.data.likes || 0}</span>
                  </button>
                  <button className="text-gray-600 hover:text-blue-500">
                    💬 {node.data.comments?.length || 0}
                  </button>
                </div>
                <div className="flex gap-2">
```

```
                    <button
                      onClick={onToggleEdit}
                      className="px-4 py-2 bg-blue-500 text-white rounded hover:bg-
blue-600"
                    >
                      수정
                    </button>
                    <button
                      onClick={onClose}
                      className="px-4 py-2 bg-gray-100 text-gray-700 rounded
hover:bg-gray-200"
                    >
                      닫기
                    </button>
                  </div>
                </div>

                {node.data.comments && node.data.comments.length > 0 && (
                  <div className="border-t pt-4 mt-4">
                    <h4 className="font-medium mb-2">댓글</h4>
                    <div className="space-y-2">
                      {node.data.comments.map(comment => (
                        <div key={comment.id} className="bg-gray-50 p-2 rounded">
                          <div className="flex justify-between text-sm">
                            <span className="font-medium">{comment.author}</span>
                            <span className="text-gray-
```

```
500">{comment.createdAt}</span>
                                </div>
                                <p className="text-sm mt-1">{comment.text}</p>
                              </div>
                            ))}
                          </div>
                        </div>
                      )}

                      <div className="border-t pt-4 mt-4">
                        <h4 className="font-medium mb-2">댓글 작성</h4>
                        <div className="space-y-2">
                          <textarea
                            placeholder="댓글을 입력하세요..."
                            className="w-full border rounded p-2"
                            rows={2}
                            value={newComment}
                            onChange={(e) => setNewComment(e.target.value)}
                          />
                          <button
                            onClick={handleAddComment}
                            className="px-3 py-1 bg-blue-500 text-white rounded hover:bg-
blue-600"
                          >
                            댓글 작성
```

```
                    </button>
                  </div>
                </div>
              </div>
            )}
          </div>

          {/* 유튜브 검색 모달 */}
          {showYoutubeSearch && (
            <YoutubeSearch
              onSelect={(url) => {
                setFormData(prev => ({ ...prev, videoUrl: url }));
                const id = getYoutubeID(url);
                setVideoId(id);
                setShowYoutubeSearch(false);
              }}
              onClose={() => setShowYoutubeSearch(false)}
            />
          )}
        </div>
      </div>
    );
};

const nodeTypes = {
```

```
  default: CustomNode,
};

const TreeEdit = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { currentTree, loading, error } = useSelector((state: RootState) => state.trees);

  const [nodes, setNodes, onNodesChange] = useNodesState(initialNodes);
  const [edges, setEdges, onEdgesChange] = useEdgesState([]);
  const [isSaving, setIsSaving] = useState(false);
  const [selectedNode, setSelectedNode] = useState<Node | null>(null);
  const [selectedEdge, setSelectedEdge] = useState<Edge | null>(null);
  const [isDragging, setIsDragging] = useState(false);
  const [reactFlowInstance, setReactFlowInstance] = useState<ReactFlowInstance |
null>(null);
  const [showNodeDetail, setShowNodeDetail] = useState(false);
  const [isEditMode, setIsEditMode] = useState(false);
  const [lastClickTime, setLastClickTime] = useState(0);
  const [autoSaveStatus, setAutoSaveStatus] = useState<string | null>(null);
  const [isEditing, setIsEditing] = useState(false);
  const [saveTimeoutId, setSaveTimeoutId] = useState<NodeJS.Timeout | null>(null);
  const [autoSaveEnabled, setAutoSaveEnabled] = useState<boolean>(false);
  const [showYoutubeSearch, setShowYoutubeSearch] = useState(false);
```

```
const [youtubeUrl, setYoutubeUrl] = useState('');
const [formData, setFormData] = useState({
  title: '',
  description: '',
  videoUrl: '',
  videoTitle: ''
});

useEffect(() => {
  if (id) {
    dispatch(fetchTreeById(id) as any);
  }
}, [dispatch, id]);

useEffect(() => {
  if (currentTree?.nodes && currentTree?.edges) {
    const nodesWithDefaults = currentTree.nodes.map((node: any) => ({
      ...node,
      type: 'default'
    }));
    setNodes(nodesWithDefaults);
    setEdges(currentTree.edges);
  }
}, [currentTree, setNodes, setEdges]);
```

```
useEffect(() => {
  if (selectedNode) {
    setFormData({
      title: selectedNode.data.title || '',
      description: selectedNode.data.description || '',
      videoUrl: selectedNode.data.videoUrl || '',
      videoTitle: selectedNode.data.videoTitle || ''
    });
  }
}, [selectedNode]);

// 자동 저장 토글 함수
const toggleAutoSave = useCallback(() => {
  setAutoSaveEnabled(prev => !prev);
  if (!autoSaveEnabled) {
    setAutoSaveStatus('자동 저장이 활성화되었습니다');
    setTimeout(() => setAutoSaveStatus(null), 3000);
  } else {
    setAutoSaveStatus('자동 저장이 비활성화되었습니다');
    setTimeout(() => setAutoSaveStatus(null), 3000);
  }
}, [autoSaveEnabled]);

// 자동 저장 기능 (조건부 활성화)
useEffect(() => {
```

```
    if (autoSaveEnabled && id && nodes.length > 0 && !isEditing) {
      // 기존 타이머가 있으면 취소
      if (saveTimeoutId) {
        clearTimeout(saveTimeoutId);
      }

      // 새 타이머 설정 (30초 지연)
      const timerId = setTimeout(() => {
        handleSave();
      }, 30000); // 30초로 늘림

      setSaveTimeoutId(timerId);

      return () => {
        if (timerId) clearTimeout(timerId);
      };
    }
  }, [nodes, edges, id, isEditing, autoSaveEnabled]);

  // 노드 변경 이벤트 처리 수정
  const handleNodesChange = useCallback((changes: any) => {
    setIsEditing(true); // 편집 시작
    onNodesChange(changes);

    // 편집 완료 타이머 설정
```

```
    setTimeout(() => {
      setIsEditing(false);
    }, 1000); // 편집 후 1초 동안은 저장하지 않음
  }, [onNodesChange]);

  // 엣지 변경 이벤트 처리 수정
  const handleEdgesChange = useCallback((changes: any) => {
    setIsEditing(true); // 편집 시작
    onEdgesChange(changes);

    // 편집 완료 타이머 설정
    setTimeout(() => {
      setIsEditing(false);
    }, 1000); // 편집 후 1초 동안은 저장하지 않음
  }, [onEdgesChange]);

  // 연결 이벤트 처리 수정
  const onConnect = useCallback((params: Connection) => {
    setIsEditing(true);

    // 연결선 스타일 개선
    setEdges((eds) => addEdge({
      ...params,
      type: 'smoothstep',
      animated: false,
```

```
        style: {
          strokeWidth: 3,
          stroke: '#666',
        },
        markerEnd: undefined
      }, eds));

      setTimeout(() => {
        setIsEditing(false);
      }, 1000);
    }, [setEdges]);

    // 저장 함수 수정
    const handleSave = async () => {
      if (!id || isEditing) return; // 편집 중이면 저장하지 않음

      setIsSaving(true);
      setAutoSaveStatus('저장 중...');
      try {
        await dispatch(updateTreeNodes({ treeId: id, nodes, edges }) as any);
        setAutoSaveStatus('저장 완료');
        // 3초 후 상태 메시지 제거
        setTimeout(() => setAutoSaveStatus(null), 3000);
      } catch (err) {
        console.error('트리 저장 실패:', err);
```

```
      setAutoSaveStatus('저장 실패');
    } finally {
      setIsSaving(false);
    }
  };

  const onNodeClick: NodeMouseHandler = useCallback((event, node) => {
    event.preventDefault();
    const currentTime = new Date().getTime();
    const timeDiff = currentTime - lastClickTime;

    if (timeDiff < 300) { // 더블 클릭 감지
      setSelectedNode(node);
      setShowNodeDetail(true);
      // 노드에 videoUrl이 없는 경우에만 편집 모드로 시작
      setIsEditMode(!node.data.videoUrl);
    } else {
      setSelectedNode(node);
      setSelectedEdge(null);
    }

    setLastClickTime(currentTime);
  }, [lastClickTime]);

  const onEdgeClick = useCallback((event: React.MouseEvent, edge: Edge) => {
```

```
    event.preventDefault();
    setSelectedEdge(edge);
    setSelectedNode(null);
  }, []);

  const onPaneClick = useCallback((event: React.MouseEvent) => {
    // 모달이 열려있고, 실제로 pane을 클릭했을 때만 모달 닫기
    if (showNodeDetail && (event.target as HTMLElement).classList.contains('react-flow__pane')) {
      setSelectedNode(null);
      setSelectedEdge(null);
      setShowNodeDetail(false);
      setIsEditMode(false);
    }
  }, [showNodeDetail]);

  const addNewNode = useCallback((position: { x: number, y: number }) => {
    const newNode: Node<NodeData> = {
      id: `node-${Date.now()}`,
      position,
      data: { label: `노드 ${nodes.length + 1}` },
      type: 'default'
    };
    setNodes((nds) => [...nds, newNode]);
  }, [nodes.length, setNodes]);
```

```
  const onPaneMouseDown = useCallback(() => {
    setIsDragging(true);
    document.body.classList.add('react-flow-grabbing');
  }, []);

  const onPaneMouseUp = useCallback(() => {
    setIsDragging(false);
    document.body.classList.remove('react-flow-grabbing');
  }, []);

  const onDoubleClick = useCallback((event: React.MouseEvent) => {
    // 더블클릭이 빈 공간에서 발생했는지 확인
    if ((event.target as HTMLElement).classList.contains('react-flow__pane')) {
      if (!reactFlowInstance) return;

      // ReactFlow 요소의 경계 정보 가져오기
      const reactFlowBounds = document.querySelector('.react-
flow')?.getBoundingClientRect();
      if (!reactFlowBounds) return;

      // 뷰포트 정보 가져오기
      const { zoom } = reactFlowInstance.getViewport();

      // 실제 마우스 클릭 위치 계산
```

```
      const position = reactFlowInstance.screenToFlowPosition({
        x: event.clientX,
        y: event.clientY
      });

      // 새 노드 생성
      const newNode = {
        id: `node-${Date.now()}`,
        position,
        data: { label: `노드 ${nodes.length + 1}` },
        type: 'default'
      };

      setNodes((nds) => [...nds, newNode]);
    }
  }, [reactFlowInstance, nodes.length, setNodes]);

  const handleNodeDataSave = useCallback((data: NodeData) => {
    if (!selectedNode) return;

    setNodes(nds => nds.map(node => {
      if (node.id === selectedNode.id) {
        return {
          ...node,
          data: {
```

```
                ...data,
                label: data.videoTitle || node.data.label
              }
            };
          }
          return node;
        }));
    }, [selectedNode, setNodes]);

    const deleteSelectedNode = useCallback(() => {
      if (selectedNode) {
        setNodes(nodes.filter(n => n.id !== selectedNode.id));
        setEdges(edges.filter(e => e.source !== selectedNode.id && e.target !== selectedNode.id));
        setSelectedNode(null);
      }
    }, [selectedNode, nodes, edges, setNodes, setEdges]);

    const deleteSelectedEdge = useCallback(() => {
      if (selectedEdge) {
        setEdges(edges.filter(e => e.id !== selectedEdge.id));
        setSelectedEdge(null);
      }
    }, [selectedEdge, edges, setEdges]);
```

```
  const addNodeAtCenter = useCallback(() => {
    if (!reactFlowInstance) return;

    const { x, y, zoom } = reactFlowInstance.getViewport();
    addNewNode({ x: x + window.innerWidth / (2 * zoom), y: y + window.innerHeight / (2 * zoom) });
  }, [reactFlowInstance, addNewNode]);

  useEffect(() => {
    const handleKeyDown = (event: KeyboardEvent) => {
      if ((event.key === 'Delete' || event.key === 'Backspace') && selectedNode) {
        event.preventDefault();
        setNodes(nodes => nodes.filter(n => n.id !== selectedNode.id));
        setEdges(edges => edges.filter(e => e.source !== selectedNode.id && e.target !== selectedNode.id));
        setSelectedNode(null);
      }
    };

    window.addEventListener('keydown', handleKeyDown);
    return () => window.removeEventListener('keydown', handleKeyDown);
  }, [selectedNode, setNodes, setEdges]);

  // 문서 레벨 드롭 이벤트 핸들러
  useEffect(() => {
    const handleDocumentDrop = (e: DragEvent) => {
```

```
        e.preventDefault();

        // 입력 필드가 포커스되어 있는지 확인
        const activeElement = document.activeElement;
        const isInputFocused = activeElement && activeElement.id === 'youtube-url-
input';

        const reactFlowEl = document.querySelector('.react-flow');
        if (!reactFlowEl || !reactFlowInstance) return;

        const reactFlowBounds = reactFlowEl.getBoundingClientRect();
        const isWithinReactFlow =
          e.clientX >= reactFlowBounds.left &&
          e.clientX <= reactFlowBounds.right &&
          e.clientY >= reactFlowBounds.top &&
          e.clientY <= reactFlowBounds.bottom;

        if (isWithinReactFlow) {
          const position = reactFlowInstance.screenToFlowPosition({
            x: e.clientX,
            y: e.clientY
          });

          const url = e.dataTransfer?.getData('text');
          if (url) {
```

```
            const videoId = getYoutubeID(url);
            if (videoId) {
              // URL을 입력 필드에 설정
              if (isInputFocused && selectedNode) {
                // 노드 수정 모달에서 input이 포커스된 경우
                // NodeDetailModal 컴포넌트의 formData와 videoId를 직접 업데이트할 수 없으므로
                // 대신 노드 데이터를 업데이트하여 모달이 리렌더링되도록 함
                const fullUrl = `https://www.youtube.com/watch?v=${videoId}`;
                setNodes(nds => nds.map(n => {
                  if (n.id === selectedNode.id) {
                    return {
                      ...n,
                      data: {
                        ...n.data,
                        videoUrl: fullUrl
                      }
                    };
                  }
                  return n;
                }));

                // 모달이 업데이트되도록 선택된 노드도 업데이트
                setSelectedNode(prev => {
                  if (!prev) return prev;
```

```
        return {
          ...prev,
          data: {
            ...prev.data,
            videoUrl: fullUrl
          }
        };
      });
    } else {
      // 새 노드 생성
      const newNode = {
        id: `node-${Date.now()}`,
        position,
        data: {
          label: `Video ${nodes.length + 1}`,
          videoUrl: `https://www.youtube.com/watch?v=${videoId}`
        },
        type: 'default'
      };

      setNodes((nds) => [...nds, newNode]);

      fetchYoutubeMetadata(videoId).then(metadata => {
        if (metadata) {
          setNodes(nds => nds.map(n => {
```

```
                        if (n.id === newNode.id) {
                            return {
                                ...n,
                                data: {
                                    ...n.data,
                                    videoTitle: metadata.title,
                                    description: metadata.description
                                }
                            };
                        }
                        return n;
                    }));
                }
            });
        }
    }
  }
};

document.addEventListener('drop', handleDocumentDrop);
document.addEventListener('dragover', e => e.preventDefault());

return () => {
  document.removeEventListener('drop', handleDocumentDrop);
```

```
      document.removeEventListener('dragover', e => e.preventDefault());
    };
  }, [reactFlowInstance, nodes.length, setNodes, selectedNode, setSelectedNode]);

  if (loading === 'pending') {
    return (
      <div className="h-screen flex justify-center items-center">
        <div className="text-gray-600">로딩 중...</div>
      </div>
    );
  }

  if (error) {
    return (
      <div className="h-screen flex justify-center items-center">
        <div className="text-red-600">에러: {error}</div>
      </div>
    );
  }

  return (
    <div className="h-screen flex flex-col">
      <div className="p-4 bg-white border-b flex justify-between items-center">
        <div>
          <h1 className="text-xl font-bold">{currentTree?.title || '트리 편집'}</h1>
```

```
          </div>
          <div className="flex gap-2 items-center">
            <button
              onClick={toggleAutoSave}
              className={`px-2 py-2 rounded-full flex items-center ${
                autoSaveEnabled ? 'bg-green-100 text-green-600' : 'bg-gray-100 text-gray-500'
              }`}
              title={autoSaveEnabled ? "자동 저장 활성화됨" : "자동 저장 비활성화됨"}
            >
              <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20" fill="currentColor">
                {autoSaveEnabled ? (
                  // 열린 자물쇠 아이콘
                  <path d="M10 2a5 5 0 00-5 5v2a2 2 0 00-2 2v5a2 2 0 002 2h10a2 2 0 002-2v-5a2 2 0 00-2-2H7V7a3 3 0 015.905-.75 1 1 0 001.937-.5A5.002 5.002 0 0010 2z" />
                ) : (
                  // 닫힌 자물쇠 아이콘
                  <path fillRule="evenodd" d="M5 9V7a5 5 0 0110 0v2a2 2 0 012 2v5a2 2 0 01-2 2H5a2 2 0 01-2-2v-5a2 2 0 012-2zm8-2v2H7V7a3 3 0 016 0z" clipRule="evenodd" />
                )}
              </svg>
            </button>

            {autoSaveEnabled && (
```

```
            <span className="text-xs text-gray-500 mr-2">자동 저장 중</span>
          )}

          <button
            onClick={handleSave}
            disabled={isSaving}
            className="px-4 py-2 bg-blue-500 text-white rounded hover:bg-blue-600 disabled:opacity-50"
          >
            {isSaving ? '저장 중...' : '변경사항 저장'}
          </button>

          <button
            onClick={() => navigate(`/trees/${id}`)}
            className="px-4 py-2 bg-gray-100 text-gray-700 rounded hover:bg-gray-200"
          >
            취소
          </button>
        </div>
      </div>
      <div className="flex-1 relative">
        <ReactFlow
          nodes={nodes}
          edges={edges}
          onNodesChange={handleNodesChange}
```

```
onEdgesChange={handleEdgesChange}
onConnect={onConnect}
onNodeClick={onNodeClick}
onEdgeClick={onEdgeClick}
onPaneClick={onPaneClick}
onDoubleClick={onDoubleClick}
onMouseDown={onPaneMouseDown}
onMouseUp={onPaneMouseUp}
onInit={setReactFlowInstance}
nodeTypes={nodeTypes}
fitView
connectionMode={ConnectionMode.Loose}
connectionLineStyle={{
   stroke: '#666',
   strokeWidth: 3,
   strokeDasharray: 'none'
}}
defaultEdgeOptions={{
   type: 'smoothstep',
   animated: false,
   style: {
      stroke: '#666',
      strokeWidth: 3,
      strokeDasharray: 'none'
   }
```

```
        }}
        zoomOnDoubleClick={false}
        panOnDrag={true}
        className="react-flow"
        style={{ cursor: isDragging ? 'grabbing' : 'default' }}
      >
        <Background />
        <Controls />
        <Panel position="top-left" style={{ background: 'white', padding: '10px', borderRadius: '4px', boxShadow: '0 2px 4px rgba(0,0,0,0.1)' }}>
          <div className="text-sm space-y-2">
            <div className="font-medium mb-2">도구</div>
            <div className="space-y-1">
              <button
                onClick={addNodeAtCenter}
                className="w-full px-3 py-1 text-left hover:bg-gray-100 rounded flex items-center gap-2"
              >
                <span className="text-blue-500">+</span> 새 노드 추가
              </button>
              {selectedNode && (
                <button
                  onClick={deleteSelectedNode}
                  className="w-full px-3 py-1 text-left hover:bg-gray-100 rounded flex items-center gap-2 text-red-500"
                >
```

```
                <span>×</span> 선택한 노드 삭제
              </button>
            )}
            {selectedEdge && (
              <button
                onClick={deleteSelectedEdge}
                className="w-full px-3 py-1 text-left hover:bg-gray-100 rounded flex items-center gap-2 text-red-500"
              >
                <span>×</span> 선택한 연결선 삭제
              </button>
            )}
          </div>
          <div className="border-t pt-2 mt-2">
            <div className="text-xs text-gray-500 mb-1">조작 방법</div>
            <p>• 더블 클릭: 새 노드 추가</p>
            <p>• 클릭: 노드/연결선 선택</p>
            <p>• 노드 더블 클릭: 내용 보기/편집</p>
            <p>• Delete: 선택한 항목 삭제</p>
            <p>• 드래그: 화면 이동</p>
            <p>• 노드 드래그: 노드 이동</p>
            <p>• 노드 연결: 드래그 앤 드롭</p>
          </div>
        </div>
      </Panel>
```

```
            </ReactFlow>
            {autoSaveStatus && (
              <div className="fixed bottom-4 right-4 bg-gray-800 text-white px-4 py-2 rounded shadow-lg opacity-75">
                {autoSaveStatus}
              </div>
            )}
            {showNodeDetail && selectedNode && (
              <NodeDetailModal
                node={selectedNode}
                onClose={() => {
                  setShowNodeDetail(false);
                  setIsEditMode(false);
                }}
                onSave={handleNodeDataSave}
                isEdit={isEditMode}
                onToggleEdit={() => setIsEditMode(!isEditMode)}
              />
            )}
          </div>
        </div>
    );
};

export default TreeEdit;
```