```
import React, { useEffect, useRef, useState } from 'react';
import { TreeView } from 'devextreme-react/tree-view';
import { getCaFictifsByFifthLevelServiceId } from 'your-api-service'; // Replace with actual import
import { administrationConstants } from 'your-constants';
import styles from './CAFictifTreeview.module.scss';
import { CAFictifEntities, CAFictifTreeviewData } from './types';

interface CAFictifTreeviewProps {
  fifthLevelServices: CAFictifEntities[];
  treeviewid: string;
  selectedCaFictif: CAFictifTreeviewData | null;
  visible: boolean;
  addBaseNode: boolean;
  onTreeItemclick: Function;
  onTreeLoaded?: (entities: CAFictifEntities[]) => void;
}

interface CAFictifTreeviewState {
  caFictifTreeData: CAFictifTreeviewData[] | null;
  selectedCaFictifStruct: CAFictifEntities | null;
}

const caFictifEntityCache = new Map<string, CAFictifEntities[]>();

export const CAFictifTreeview = (props: CAFictifTreeviewProps) => {
  const {
    fifthLevelServices,
    treeviewid,
    visible,
    addBaseNode,
    selectedCaFictif,
    onTreeItemclick,
    onTreeLoaded
  } = props;

  const [caFictifTreeviewState, setCaFictifTreeviewState] = useState<CAFictifTreeviewState>({
```

```
  caFictifTreeData: null,
  selectedCaFictifStruct: null
});

const treeviewRef = useRef<TreeView>(null);

useEffect(() => {
  constructCAFictifTreeviewData(administrationConstants.fictiveCa);
}, []);

const constructCAFictifTreeviewData = async (serviceId: string) => {
  const fifthLevelServiceData = await getCached(serviceId);
  const filteredServices = filterFifthLevelServices(fifthLevelServiceData);
  const baseNodeId = '0';

  const nodes = filteredServices.map((entity, index) => ({
    id: `${baseNodeId}_${index + 1}`,
    text: `${entity.serviceId} ${entity.entity}`,
    serviceLevel: entity.level,
    serviceId: entity.serviceId,
    service: entity.entity,
    endDate: entity.endDate,
    startDate: entity.startDate,
    color: entity.color,
    denomination: entity.denomination,
    items: [] // lazy-loaded
  }));

  const resultTree = addBaseNode
    ? [{
        id: baseNodeId,
        text: administrationConstants.caFictifBaseNodeText,
        items: nodes
      }]
    : nodes;
```

```
  setCaFictifTreeviewState(prev => ({
    ...prev,
    caFictifTreeData: resultTree
  }));

  onTreeLoaded?.(filteredServices);
};

const getCached = async (serviceId: string): Promise<CAFictifEntities[]> => {
  if (caFictifEntityCache.has(serviceId)) {
    return caFictifEntityCache.get(serviceId)!;
  }
  const data = await getCaFictifsByFifthLevelServiceId(serviceId);
  caFictifEntityCache.set(serviceId, data);
  return data;
};

const filterFifthLevelServices = (services: CAFictifEntities[]) => {
  return services.filter(service =>
    service.serviceId !== administrationConstants.nonAffectableCa &&
    service.entity.toUpperCase() !== administrationConstants.affecter
  );
};

const handleItemExpanded = async (e: any) => {
  const item = e.itemData as CAFictifTreeviewData;

  if (item.items && item.items.length > 0) return;

  const allEntities = await getCached(item.serviceId);

  const childEntities = allEntities.filter(entity => {
    const currentLevel = parseInt(item.serviceLevel || '05');
    const nextLevel = (currentLevel - 1).toString().padStart(2, '0');

    return entity.level === nextLevel && matchesParent(entity, item, currentLevel);
```

```
    });

    const children = childEntities.map((entity, index) => ({
      id: `${item.id}_${index + 1}`,
      text: `${entity.serviceId} ${entity.entity}`,
      serviceLevel: entity.level,
      serviceId: entity.serviceId,
      service: entity.entity,
      endDate: entity.endDate,
      startDate: entity.startDate,
      color: entity.color,
      denomination: entity.denomination,
      items: []
    }));

    item.items = children;

    setCaFictifTreeviewState(prev => ({
      ...prev,
      caFictifTreeData: [...(prev.caFictifTreeData || [])]
    }));
  };

  const matchesParent = (entity: CAFictifEntities, parent: CAFictifTreeviewData, currentLevel:
number) => {
    switch (currentLevel) {
      case 5:
        return entity.fifthLevelServiceId === parent.serviceId;
      case 4:
        return entity.fourthLevelServiceId === parent.serviceId;
      case 3:
        return entity.thirdLevelServiceId === parent.serviceId;
      case 2:
        return entity.secondLevelServiceId === parent.serviceId;
      case 1:
        return entity.firstLevelServiceId === parent.serviceId;
```

```
    default:
      return false;
  }
};

const renderTreeItem = (item: CAFictifTreeviewData) => {
  const isBaseNode = item.id === '0';
  const isSelected = selectedCaFictif?.id === item.id;
  const fontColor = isBaseNode ? 'black' : item.endDate ? 'red' : 'black';
  const finalColor = isSelected ? 'blue' : fontColor;

  return (
    <div className={styles.treeItem} style={{ color: finalColor }}>
      {item.text}
    </div>
  );
};

if (!caFictifTreeviewState.caFictifTreeData) {
  return <div>Loading Tree...</div>;
}

return (
  <div className={styles.leftContainer}>
    <TreeView
      ref={treeviewRef}
      id={treeviewid}
      dataSource={caFictifTreeviewState.caFictifTreeData}
      focusStateEnabled={false}
      itemRender={renderTreeItem}
      onItemClick={(e: any) => onTreeItemclick(e.itemData)}
      onItemExpanded={handleItemExpanded}
      visible={visible}
    />
  </div>
);
```

```
};
```