1. 环境/技术选型

React+bootstrap，node.js，mysql

1. 数据库设计（需等待）

用户表（是否需要用户注册，考虑安全性问题？）

素材库表

成品库表

1. 后端开发

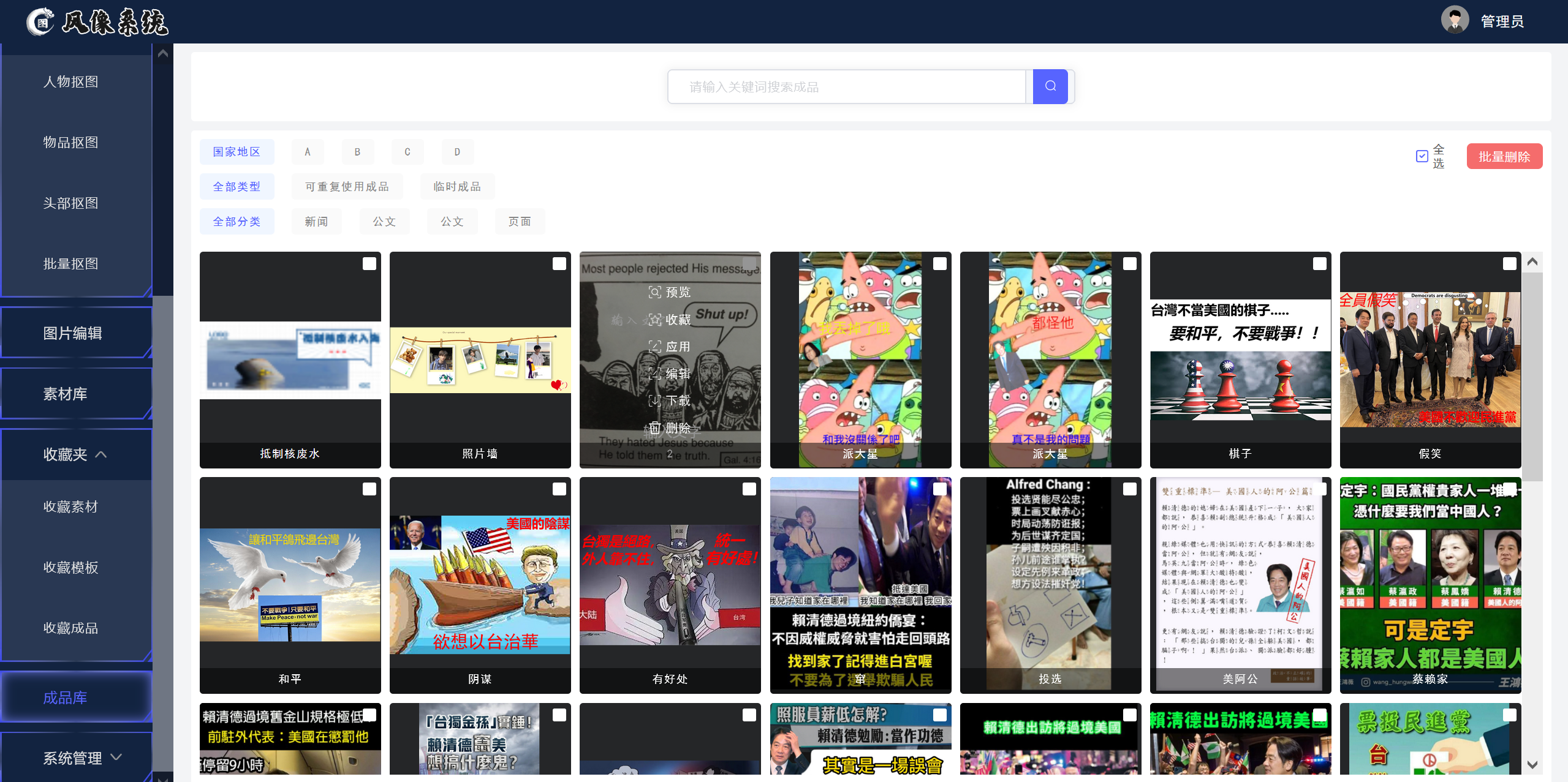
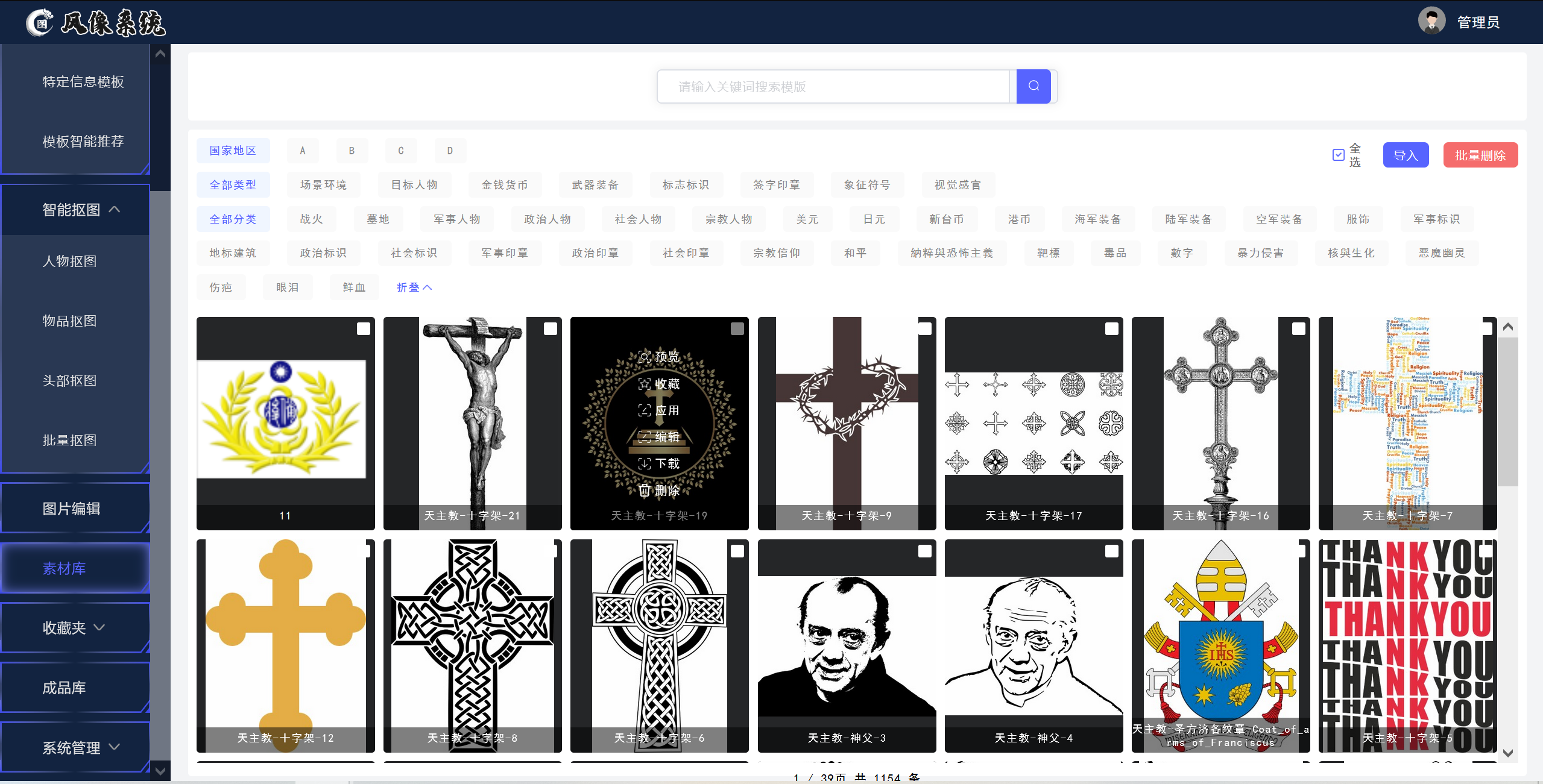
用户

提供上传、删除、另存为/下载、详情页签、编辑页签（？）和检索的api

1. 前端开发

是否需要响应式布局（为移动端重新编写布局样式）





1. 部署

是否需要部署在云服务器上

搜索框的实现：  
const express = require('express');

const router = express.Router();

const mysql = require('mysql');

// 创建MySQL连接

const connection = mysql.createConnection({

host: 'localhost',

user: 'yourUsername',

password: 'yourPassword',

database: 'yourDatabase'

});

// 连接到数据库

connection.connect();

// 搜索API

router.get('/search', (req, res) => {

const searchQuery = req.query.q; // 从请求中获取搜索关键词

const searchType = req.query.type; // 可选，用于指定搜索的类型（素材库或成品库）

// 构建查询语句，这里使用了LIKE进行模糊匹配

let query = 'SELECT \* FROM your\_table WHERE title LIKE ?';

if (searchType) {

query += ' AND type = ?'; // 如果指定了类型，添加到查询条件中

}

// 执行查询

connection.query(query, [`%${searchQuery}%`, searchType], (error, results) => {

if (error) {

return res.status(500).send(error);

}

res.json(results);

});

});

module.exports = router;

import React, { useState } from 'react';

function SearchComponent() {

const [searchTerm, setSearchTerm] = useState('');

const [results, setResults] = useState([]);

const handleSearch = async () => {

try {

const response = await fetch(`/search?q=${encodeURIComponent(searchTerm)}`);

const data = await response.json();

setResults(data);

} catch (error) {

console.error('Error fetching data: ', error);

}

};

return (

<div>

<input

type="text"

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

placeholder="Search..."

/>

<button onClick={handleSearch}>Search</button>

<div>

{results.map((result, index) => (

<div key={index}>{result.title}</div> // 显示搜索结果

))}

</div>

</div>

);

}

export default SearchComponent;

批量选择功能：

**前端实现：**

1. **批量选择框**：在每个素材选项旁边添加一个复选框，允许用户选择多个素材。
2. **批量操作按钮**：提供一个或多个按钮，如“批量删除”或“批量下载”，用户在选择了多个素材后可以点击这些按钮来执行操作。
3. **状态管理**：管理哪些项被选中，通常可以通过在每个素材对象上添加一个selected属性来实现，或者使用一个数组来跟踪所有选中的素材ID。
4. **发送请求**：当用户点击批量操作按钮时，前端需要收集所有选中的素材ID，并将它们作为请求的一部分发送到后端。

**后端实现：**

1. **接收批量请求**：后端需要能够接收包含多个ID的请求。
2. **执行批量操作**：根据前端发送的ID列表，执行相应的数据库操作，如批量删除或批量下载。
3. **返回结果**：操作完成后，后端需要返回操作结果，告知前端操作是否成功。

import React, { useState } from 'react';

function MaterialList({ materials, onBulkDelete }) {

const [selectedMaterials, setSelectedMaterials] = useState([]);

const handleSelectAll = (materialId) => {

setSelectedMaterials(prevSelectedMaterials =>

prevSelectedMaterials.includes(materialId)

? prevSelectedMaterials.filter(id => id !== materialId)

: [...prevSelectedMaterials, materialId]

);

};

const handleBulkDelete = () => {

onBulkDelete(selectedMaterials);

setSelectedMaterials([]); // 清空选择

};

return (

<div>

{materials.map(material => (

<div key={material.id}>

<input

type="checkbox"

checked={selectedMaterials.includes(material.id)}

onChange={() => handleSelectAll(material.id)}

/>

{material.title}

</div>

))}

<button onClick={handleBulkDelete}>批量删除</button>

</div>

);

}  
const express = require('express');

const router = express.Router();

// 假设你有一个函数来处理实际的删除操作

const deleteMaterials = (materialIds) => {

// 删除数据库中的记录

};

router.post('/delete-bulk', (req, res) => {

const materialIds = req.body.materialIds;

try {

deleteMaterials(materialIds);

res.send({ success: true });

} catch (error) {

res.status(500).send({ success: false, message: error.message });

}

});

module.exports = router;

import React, { useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import axios from 'axios';

function HomePage() {

    const [user, setUser] = useState({ name: 'user', isLoggedIn: false, userType: '' });

    const navigate = useNavigate();

*//处理登录事件*

    const handleLogin = async (*event*) => {

*event*.preventDefault(); *// 阻止表单默认提交行为*

        try {

*// 使用 axios 发送 POST 请求到你的登录 API*

            const response = await axios.post('/api/login', user);

*// 假设登录成功后，服务器会返回一些用户信息*

            setUser(response.data);

            if (response.data.isLoggedIn) {

                navigate('/material'); *// 重定向到仪表盘或其他页面*

            }

        } catch (error) {

            console.error('登录失败:', error);

*// 这里可以添加错误处理逻辑，比如显示一个错误消息*

        }

    };

    return (

        <div *className*="home-container">

            <h1>产品名称</h1>

            <div *className*="login-card">

                <h5 *className*="card-title">Login to get started</h5>

                <form *onSubmit*={handleLogin}>

                    <div *className*="form-group">

                        <label *htmlFor*="userId">ID</label>

                        <input

*type*="text"

*className*="form-control"

*id*="userId"

*placeholder*="输入ID"

*value*={user.name}

*onChange*={(*e*) => setUser({ ...user, name: *e*.target.value })}

                        />

                    </div>

                    <div *className*="form-group">

                        <label *htmlFor*="password">密码</label>

                        <input

*type*="password"

*className*="form-control"

*id*="password"

*placeholder*="输入密码"

*value*={user.password}

*onChange*={(*e*) => setUser({ ...user, password: *e*.target.value })}

                        />

                    </div>

                    <button *type*="submit" *className*="btn btn-primary btn-block">

                        登录

                    </button>

                </form>

            </div>

        </div>

    );

}

export default HomePage;