1.项目介绍

智慧园区，简单来说，就是应用了现代信息技术来管理和优化园区运作的高科技园区。这种园区通过集成物联网、大数据、人工智能等技术，提高了管理效率，节约能源，增强安全保障，改善环境质量，从而为居住、工作的人提供了一个更加舒适、智能和便捷的空间。

想象一下你进入一个园区，所有的设施——如照明、安全监控、能源使用等——都是自动控制的。比如，园区内的路灯能根据天气和周围亮度自动调节亮度；园区的能源系统会根据实际用电情况自动调整，以节约电能和降低成本；安全系统能通过智能相机监控园区安全，甚至能自动识别可疑行为并及时报警。

这些只是一些例子，智慧园区还可以通过手机应用、智能终端等方式，实现对车辆停放、会议室预定、景观灌溉等多种服务的智能管理。总之，智慧园区通过科技的力量，使得园区的运营更加智能和高效，提升了居住和工作的品质，并对环境友好。

这是一个智慧园区项目，通过这个平台对园区内

2.技术框架

> **React18 + redux(RTK) + react-router v6.4 + TypeScript + antd + webpack+ echarts + create-react-app**

3.项目环境搭建

1.使用creaet-react-app的ts模板创建项目

```
npx create-react-app my-app --template [template-name]
```

```
npx create-react-app my-app --template typescript
```

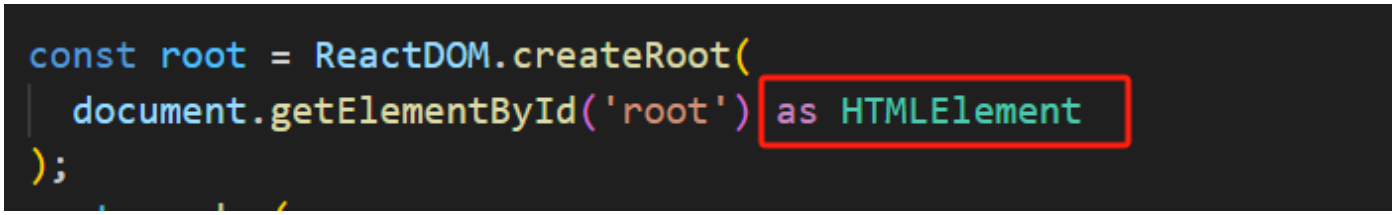
2.安装react-router

```
npm install react-router-dom
```

3.安装 antd

```
npm install antd --save
```

关于为什么要添加 as HTMLElement



document.getElementById 返回 HTMLElement | null 类型。null 是当没有元素与指定的 ID 匹配时的返回值。但在大多数情况下，开发人员确信特定 ID 的元素肯定存在（例如，在 index.html 中定义的根元素）。因此，为了防止 TypeScript 报告可能的 null 引用错误，使用 as HTMLElement 可以断言返回的值不是 null，确保它总是 HTMLElement 类型。这样可以直接使用该元素而不需要额外的 null 检查，使代码更为简洁。

> 如何自己封装一个类似antd中的组件

```
function Xubutton(props:any){
    const cssStyle={padding:"10px 20px"}
    return <button style=
{{...cssStyle,backgroundColor:props.type=="primary"?"blue":""}}>{props.children}
</button>
}
```

> 关于React.FC

React.FC 是 React.FunctionComponent 的缩写

```
interface XubuttonProps {
    type: string;
    children:React.ReactNode
}

const Xubutton: React.FC<XubuttonProps> = (props) => {
    const cssStyle = { padding: "10px 20px" }
    return (
        <button style={{ ...cssStyle, backgroundColor: props.type === "primary" ?
"blue" : "" }}>
            {props.children}
        </button>
    );
};
```

4.基础路由配置

注意：路由表文件后缀必须为.tsx，否则会报错

如果eslint报错，可以直接卸载掉eslint

```
npm uninstall eslint --save
```

src-router-index.tsx

```
import React from "react";
import { createBrowserRouter } from "react-router-dom";
const Home=React.lazy(()=>import("../page/home"));
const Login=React.lazy(()=>import("../page/Login"));
const NotFound=React.lazy(()=>import("../page/404"));
const router=createBrowserRouter([
    {
        path:"/",
        element:<Home/>
    },
    {
        path:"/login",
        element:<Login/>
    },
    {
        path:"*",
        element:<NotFound/>
    }
])
export default router
```

5.安装scss

sass和scss是同一个东西，老版本后缀为sass，新版本后缀为scss

npm install sass即可

6.登录页面静态结构编写

> 关于图片的引入问题

注意图片不可以直接通过相对路径引入，会出错

比如在JSX中使用，这里的路径是相对于网页文件（通常是index.html）的位置，并没有考虑到Webpack或其他构建工具将项目打包后的结构。打包过程中，Webpack等工具会将所有JS文件、CSS文件及图片等资源进行处理，然后输出到build或dist目录中。在这个过程中，原始的文件结构可能被改变

为什么js文件之间就可以互相引入，不会出错

对于JavaScript或CSS文件互相引用的相对路径，由于Webpack和其他模块打包工具是基于这类文件之间的相对路径来解析和建立依赖关系的，这些路径在开发过程中是保持不变的。打包工具能够根据这些路径，找到相关文件，并把它们打包到一起

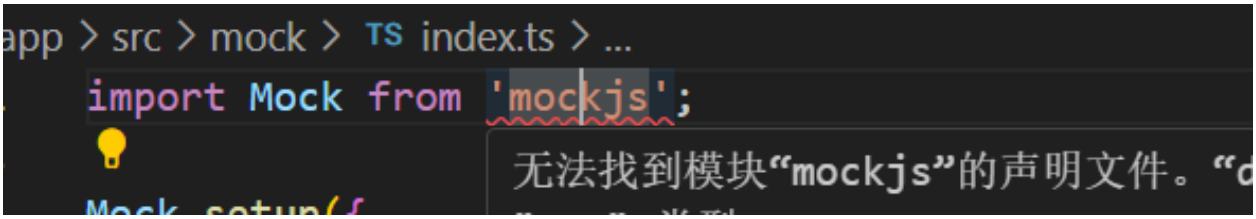
如果是在css文件中引入图片应该怎么办

将图片放在公共的 public 目录中，例如 public/a.jpg。然后在 CSS 文件中直接使用 /a.jpg。这种方法不会通过 Webpack 处理

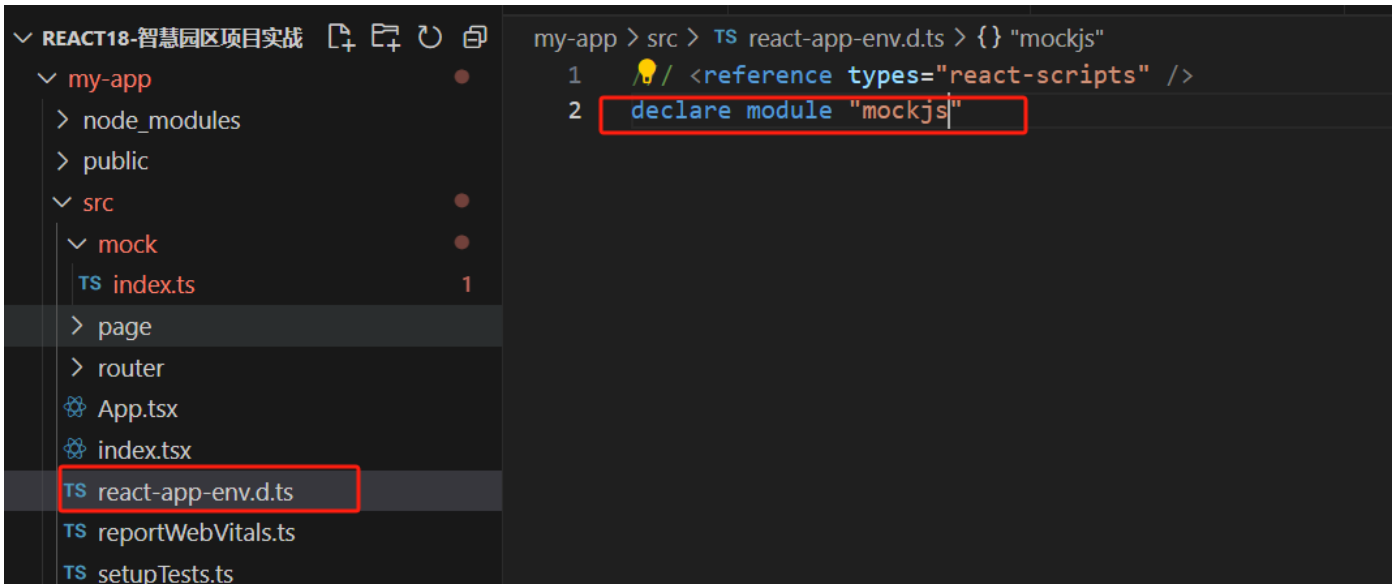
> 关于Form组件的ts类型

- 这表示Form.Item组件被设计为可接受一个泛型参数。这里的FieldType可以是任何你定义的类型，它用于指定与这个Form.Item关联的表单控件值的类型。这种做法有助于确保当你处理表单项的数据时，例如在表单提交的处理函数中，这些数据将拥有正确的类型

7.安装mockjs和axios



注意，mockjs会提示找不到他的类型，所以在类型声明文件中加一句话就行了



8.二次封装axios

```
import axios,{AxiosInstance,InternalAxiosRequestConfig,AxiosResponse} from "axios";
import { message } from "antd";
import { store } from "../../store";
const http:AxiosInstance=axios.create({
    baseURL:"https://www.demo.com",
    timeout:5000
})

//请求拦截器
http.interceptors.request.use((config:InternalAxiosRequestConfig)=>{
    const {token}=store.getState().authSlice
    if(token){
        //Authorization专门用来携带认证信息
        //Bearer表示的是一种认证类型，表示后面携带的是一个令牌
        config.headers['Authorization']=`Bearer ${token}`
    }
    return config
})


//响应拦截器
http.interceptors.response.use((response:AxiosResponse)=>{
    console.log("response",response);
    const res=response.data
    if(res.code!=200){
        message.error(res.code+":"+ res.message);
        return Promise.reject(new Error(res.message))
    }
    return response.data
})

export default http
```

在 TypeScript 中，Promise 是对 ES6 Promises 的类型封装，用于表示一个异步操作的最终完成 (或失败) 及其结果值。这里的 T 代表 Promise 将会最终解析成的值的类型

```
import http from "./http"
interface ApiResponse{
    code:number;
    message:string,
    data:any
}
export function get(url:string,params?:any):Promise<ApiResponse> {
    return http.get(url,{params})
}

export function post(url:string,data?:any):Promise<ApiResponse>{
    return http.post(url,data)
}
```

9.配置登录功能

1.token的处理

> token为什么需要同时存储到redux和本地存储里

虽然将token存储在本地存储中可以持久保存用户的登录状态，但每次需要使用token时，应用都需要从本地存储中读取，相比于内存存取来说，这是一个比较慢的操作。

使用Redux存储token可以快速访问状态，避免了频繁的IO操作（访问本地存储通常需要更多时间）。而将token同时存放在本地存储中则可以在页面加载时快速初始化应用状态，避免了额外的API调用来获取新的token。

> redux中处理token

```
import { createSlice } from "@reduxjs/toolkit";

export const authSlice=createSlice({
    name:"auth",
    initialState:{
        token:sessionStorage.getItem("token")||null
    },
    reducers:{
        setToken:(state,action)=>{
            state.token=action.payload;//把token存储到redux里
            sessionStorage.setItem("token",action.payload)
        },
        clearToken:state=>{
            state.token=null;
```

```
            sessionStorage.removeItem("token")
        }
    }
})

export const{setToken,clearToken}=authSlice.actions;
export default authSlice.reducer
```

> 拦截器里token的处理

config.headers['Authorization'] = `Bearer ${token}`;

- `'Authorization'`是HTTP标准中一个特定的头字段，用于携带认证信息。
- `Bearer`是一种认证类型，表示后面携带的是一个访问令牌（token）。

`Bearer`令牌是OAuth 2.0协议中定义的一种令牌使用方式。使用`Bearer`关键字的目的在于告诉服务器，客户端随请求发送的是一个`Bearer Token`。这种令牌通常是不透明的，服务器接收到令牌后会进行验证，并决定是否授权请求的访问。

2.登录权限控制

未登录不允许跳转到首页

已登录不允许跳转到登录页

登录成功后不能跳转到登录页，未登录不能跳转到首页

```
import React from 'react';
import { useNavigate } from 'react-router-dom';

interface Iprops{
    allowed:boolean,
    redirectTo:string,
    children:React.ReactNode
}
//如果 allowed 为 true（意味着访问当前路由需要用户已登录），且 isLoggedIn() 也返回 true（用户已登录），则表达式的结果为 true，用户可以访问该路由。
//如果 allowed 为 false（意味着访问当前路由不需要用户登录），但 isLoggedIn() 返回 true 或 false，则会根据具体的检查决定是否允许访问或重定向到其他页面。
function RequireAuth({ allowed, redirectTo,children }:Iprops) {
    const isLoggedIn = sessionStorage.getItem('token')?true:false ; // 假设登录状态保存在localStorage中
    const navigate = useNavigate();
    React.useEffect(() => {
        if (allowed !== isLoggedIn) {
            navigate(redirectTo);
        }
```

```
    }, [allowed, isLoggedIn, navigate, redirectTo]);

    return allowed === isLoggedIn ? <>{children}</> : null;
}

export default RequireAuth
```

```
const router=createBrowserRouter([
    {
        path:"/",
        element: <RequireAuth allowed={true} redirectTo="/login"> <Home/>
</RequireAuth>
    },
    {
        path:"/login",
        element:<RequireAuth allowed={false} redirectTo="/"> <Login/> </RequireAuth>
    },
    {
        path:"*",
        element:<NotFound/>
    }

])
```

10.配置主页界面

11.menu组件的开发

> **关于menu组件的ts类型**
>
> type MenuItem = Required['items'][number];

1. **`MenuProps`**：这是一个类型定义，通常在 Ant Design 的类型定义中表示 `Menu` 组件的所有属性的类型。它包括了该组件所有支持的属性，例如 `items`、`onClick`、`onSelect` 等。
2. **`Required<T>`**：这是 TypeScript 中的一个工具类型，它将传入类型 `T` 的所有属性变成必需的。即，如果原来类型中的属性是可选的（用 `?` 标记），使用 `Required<T>` 后，这些属性将不再是可选的，变为必须提供的。
3. **`MenuProps['items']`**：这表示访问 `MenuProps` 类型中 `items` 属性的类型。在 `Menu` 组件中，`items` 属性通常用于定义菜单项数据，这是一种比手动构建 `Menu.Item` 更声明式的项目定义方式。
4. **`Required<MenuProps>['items']`**：组合上面提到的 `Required` 和属性访问，这段代码获取 `MenuProps` 所有属性为必需的情形下的 `items` 属性类型。这确保 `items` 属性已经被定义在类型中，即使在原始的 `MenuProps` 类型定义中 `items` 是可选的。
5. **`Required<MenuProps>['items'][number]`**：最后这部分是访问上文得到的 `items` 数组类型中的单个元素的类型。在 TypeScript 中，使用 `[number]` 来访问数组中元素的类型。

1.根据权限生成不同的菜单

1.定义不同权限的mock数据

2.定义请求菜单的接口

3.根据后端返回的图标字符串，对应图标对照表，生成图标组件

4.重点：组件递归处理，生成符合我们要求的菜单数据

```
   async function configMenu(){
        const {data}=await getMenu();
        const mappedMenuItems:MenuItem[]=mapMenuItems(data);
        setMenuData(mappedMenuItems)
    }
    //将返回的菜单数据转换成我们需要的格式
    function mapMenuItems(items:MenuItemFromData[]):any{
        return items.map((item:MenuItemFromData)=>({
            key:item.key,
            label:item.label,
            icon:icons[item.icon],
            children:item.children ? mapMenuItems(item.children) : null //递归操作
        }))
    }
```

2.关于useEffect

```
//我们知道 effect function 应该返回一个销毁函数（effect：是指return返回的cleanup函数），如果
useEffect 第一个参数传入 async，返回值则变成了 Promise，会导致 react 在调用销毁函数的时候报错。
    useEffect(async ()=>{
     const res=await getMenu()
    },[])
```

12头部组件的开发

```
import React from 'react';
import { UserOutlined , PoweroffOutlined ,DownOutlined } from '@ant-design/icons';
import type { MenuProps } from 'antd';
import { Dropdown, Space } from 'antd';
import { clearToken } from '../../store/login/authSlice';
import { useDispatch } from 'react-redux';
const items: MenuProps['items'] = [
    {
      key: '1',
```

```
      label: (
        <a target="_blank" >
            个人中心
        </a>
      ),
      icon: <UserOutlined />,
    },
    {
      key: '2',
      label: (
        <a target="_blank" >
         退出登录
        </a>
      ),
      icon: <PoweroffOutlined />,
    },

  ];
function MyHeader() {
    const dispatch=useDispatch()
    const onClick:MenuProps['onClick']=({key})=>{
       if(key=="1"){
            //跳转到个人中心
       }else{
            dispatch(clearToken());
            sessionStorage.removeItem("username")
       }

    }
    return <div>
        <Dropdown menu={{ items,onClick}}>
            <a onClick={(e) => e.preventDefault()}>
                <Space>
                   欢迎您,{sessionStorage.getItem("username")}
                    <DownOutlined />
                </Space>
            </a>
        </Dropdown>
    </div>
}

export default MyHeader
```

13.动态路由的创建

1.根组件中通过getMenu接口获取到该权限所拥有的菜单项

```
useEffect(() => {
    async function loadData() {
      const { data } = await getMenu();
  },[token])
```

2.获取到的菜单项存到redux中，因为以后面包屑组件也需要用到这个数据

> **关于菜单项数据为什么不选择存储到本地存储里?**
>
> 存储到本地存储里有安全性问题，用户可以直接更改，改之后用户就可以访问他没有权限访问的页面了

```
dispatch(setMenu(data))
```

3.将返回的菜单项修改成我们路由需要的格式即 path、element、children的形式

generateRoutes.tsx

```
import { RouteObject } from "react-router-dom";
import { componentMap } from "../router/routerMap";
interface MenuType{
    icon:string;
    key:string;
    label:string;
    children?:MenuType[]
}
export function generateRoutes(menu:MenuType[]):RouteObject[]{
    return menu.map((item:MenuType)=>{
        const hasChildren=item.children
        let routerObj:RouteObject={
            path:item.key,
            element:hasChildren?null:<>{componentMap[item.key]}</>
        };
        if(item.children){
            routerObj.children=generateRoutes(item.children)
        }
        return routerObj
    })

}
```

4.将修改后生成的路由表加入到我们首页路由的子路由中，并且第一个子集的index设置为true，即默认子路由

```
 useEffect(() => {
    async function loadData() {
      const { data } = await getMenu();
      if (data.length) {
        dispatch(setMenu(data))
```

```
        const routers = generateRoutes(data)//动态创建的路由表
        const myRoutes = [...routes];
        myRoutes[0].children = routers;
        myRoutes[0].children[0].index = true;
        const router = createBrowserRouter(myRoutes)
        setRouter(router);
      }else{
        const router = createBrowserRouter(routes)
        setRouter(router);
      }
    }
    loadData()
  },[token])
```

14.封装面包屑导航组件

> 核心思想：

根据当前路由的路径，去循环菜单项数据，判断我这个路由路径到底是以菜单项中的哪一个打头的，把以他打头的那几项筛选出来，找出他对应的中文名称

```
import { Breadcrumb } from "antd";
import { useLocation } from "react-router-dom";
import { useSelector } from "react-redux";
interface MenuItem {
    key: string;
    label: string;
    children?: MenuItem[];
}
interface Breadcrumb {
    path: string;
    label: string;
}
function findBreadcrumbPath(path: string, menuItems: MenuItem[]): Breadcrumb[] {
    const pathSegments: Breadcrumb[] = [];
    function findPath(currentPath: string, items: MenuItem[]): Breadcrumb[] {
        for (let item of items) {
            console.log(222,item.key,currentPath)
            if (currentPath.startsWith(item.key)) {
                console.log(9,item.key)
                pathSegments.push({ path: item.key, label: item.label });
                if (item.children) {
                    findPath(currentPath, item.children);
                }
                break;
            }
        }
```

```
        console.log(787,pathSegments)
        return pathSegments;
    }
    return findPath(path, menuItems);
}

function MyBreadCrumb(){
    const location=useLocation();
    const {menuList}=useSelector((state:any)=>state.authSlice)
    const breadList=findBreadcrumbPath(location.pathname,menuList).map(item=>
({title:item.label}))
    return <Breadcrumb style={{margin:"16px 0"}}
    items={breadList}
/>
}
export default MyBreadCrumb
```

15.Dashboard页面开发

关于echarts的使用

首先，你需要安装 echarts 和 echarts-for-react。echarts 是核心图表库，而 echarts-for-react 是一个为React应用封装的组件，可以更方便地在React环境中使用ECharts。

> 基本使用方式：

```
    const initialOption = {
        title: {
            text: '当日能源消耗'
        },
        tooltip: {
            trigger: 'axis'
        },
        legend: {
            data: []
        },
        grid: {
            left: '3%',
            right: '4%',
            bottom: '3%',
            containLabel: true
        },
        toolbox: {
            feature: {
```

```
                saveAsImage: {}
            }
        },
        xAxis: {
            type: 'category',
            boundaryGap: false,
            data: ['0: 00', '4: 00', '8: 00', '12: 00', '16: 00', '20: 00', '24: 00']
        },
        yAxis: {
            type: 'value'
        },
        series: []
    };
    const [data, setData] = useState(initialOption);

    useEffect(() => {
        const loadData = async () => {
            const { data: apiData } = await getEnergyData();
            const dataList = apiData.map((item:any) => ({
                name: item.name,
                data: item.data,
                type: 'line',
                stack: 'Total'
            }));

            // Create a new option object to trigger re-render
            const updatedOption = {
                ...data,
                legend: {
                    data: dataList.map((item:any) => item.name)
                },
                series: dataList
            };

            setData(updatedOption);  // Update state with new option
        };
        loadData();
    }, []);
```

16.租户列表的开发

1.基本表格的开发

> 表格定义的两个重点

1.columns数据列

2.dataSource数据源

dataSource中数据的属性名应该对应数据列中的dataIndex属性，这样可以保证把数据渲染到对应的位置上

> 如果数据要经过处理再渲染

columns里可以写一个render函数，render函数返回的jsx即是表格列要渲染的内容

> 关于分页

注意每次切换页码和切换每页多少条，本质上都是向后端发送请求

```
<Pagination
    className="fr mt"
    total={total}
    current={page}
    pageSize={pageSize}
    showSizeChanger
    showQuickJumper
    showTotal={(total) => `共 ${total} 条`}
    onChange={onChange}
 />
```

> 关于子组件的优化

因为子组件传入属性，每次父组件数据修改数据子组件都会重新渲染，这是没有必要的，所有我们采用React.memo把子组件缓存起来

但是，因为子组件传入的属性中有一个是函数，而父组件的函数每次都会重新创建，所以父组件应该把函数缓存起来

缓存组件

```
const MyUserForm=React.memo(UserForm)
```

缓存函数

```
    const hideModal=useCallback(()=>{
        setIsModalOpen(false)
    },[])
```

17.楼宇管理页面开发

> 表头

```
const columns:TableProps<DataType>['columns']=[
    {
        title:"No.",
        key:"index",
        render:(value,record,index)=>index+1
    },
    {
        title:"楼宇名称",
        dataIndex:"name",
        key:"name"
    },
    {
        title:"负责人",
        dataIndex:"person",
        key:"person"
    },
    {
        title:"负责人电话",
        dataIndex:"tel",
        key:"tel"
    },
    {
        title:"使用状态",
        dataIndex:"status",
        key:"status",
        render:(value)=>{
            if(value==1){
                return <Tag color="#f50">建设中</Tag>
            }else if(value==2){
                return <Tag color="#2db7f5">已竣工</Tag>
            }else{
                return <Tag color="#87d068">使用中</Tag>
            }
        }
    },
    {
        title:"空置率",
        dataIndex:"vacancyRate",
        key:"vacancyRate",
        render(value){
            return <Progress percent={value} status="active" />
        }
    },
    {
        title:"物业费率",
```

```
        dataIndex:"propertyFee",
        key:"propertyFee",
        render(value){
            return <Badge color="green" text={value}></Badge>
        }
    },
    {
        title:"操作",
        key:"operate",
        render(value){
            return <>
                <Button type="primary" className="mr">编辑</Button>
                <Button type="primary" danger>删除</Button>
            </>
        }
    },
]
```

> 数据

```
const data: DataType[] = [
    {
        key: '1',
        name: 'A1幢写字楼',
        person: "王达",
        tel: '16654789654',
        status: "建设中",
        vacancyRate: 60,
        propertyFee:"3.5%",
    },
    {
        key: '2',
        name: 'A2幢写字楼',
        person: "苏乐凯",
        tel: '13698756669',
        status: "已竣工",
        vacancyRate: 40,
        propertyFee:"3.8%",
    },
    {
        key: '3',
        name: 'B1幢写字楼',
        person: "莉亚",
        tel: '15587966698',
        status: "使用中",
        vacancyRate: 20,
        propertyFee:"3.1%",
```

```
    },
    {
        key: '4',
        name: 'B2幢写字楼',
        person: "常可",
        tel: '13698756324',
        status: "使用中",
        vacancyRate: 30,
        propertyFee:"4.0%",
    },
    {
        key: '5',
        name: 'C1幢写字楼',
        person: "刘伟",
        tel: '19878965444',
        status: "使用中",
        vacancyRate: 50,
        propertyFee:"3.5%",
    },
    {
        key: '6',
        name: 'C2幢写字楼',
        person: "孙强浩",
        tel: '13369888562',
        status: "使用中",
        vacancyRate: 10,
        propertyFee:"2.9%",
    },
    {
        key: '7',
        name: '天汇国际大厦A座',
        person: "马浩瀚",
        tel: '13578549687',
        status: "使用中",
        vacancyRate: 25,
        propertyFee:"3.7%",
    },
    {
        key: '8',
        name: '时代金融广场',
        person: "杨柳",
        tel: '18745889874',
        status: "使用中",
        vacancyRate: 15,
        propertyFee:"3.3%",
    },

];
```

18.房间管理页面开发

19.车辆管理页面开发

```
import {Card,Row,Col,Table,Input,Button,Tabs,Image } from "antd"
import type { TabsProps,TableProps } from 'antd';
import come from "../../assets/come.jpg"
interface DataType {
    key: string;
    orderNo: string;
    date: string;
    carNo: string;
    type: string;
    startDate: string;
    time: string;
    count: string;
    cost: string;
}
interface DataType2 {
    key: string;
    carNo: string;
    name: string;
    tel: string;
    type: string;
    rest: string;
    time: string;
    pic: string;
}
const columns: TableProps<DataType>['columns'] = [
    {
        title: "No.",
        key: "index",
        render: (text, record, index) => index + 1,
    },
    {
        title: '订单编号',
        dataIndex: 'orderNo',
        key: 'orderNo',

    },
    {
        title: '订单日期',
        dataIndex: 'date',
        key: 'date',
    },
    {
        title: '车牌号码',
        dataIndex: 'carNo',
```

```
        key: 'carNo',
    },
    {
        title: '车辆类型',
        dataIndex: 'type',
        key: 'type',

    },
    {
        title: '充电开始时间',
        dataIndex: 'startDate',
        key: 'startDate',
    },
    {
        title: '充电时长',
        dataIndex: 'time',
        key: 'time',
    },
    {
        title: '充电量',
        dataIndex: 'count',
        key: 'count',
    },
    {
        title: '充电费用',
        dataIndex: 'cost',
        key: 'cost',
    },
    {
        title: '操作',
        dataIndex: 'operate',
        key: 'operate',
        render: (text, record) => {
            return <>
                <Button type="primary" size="small">查看</Button>
            </>
        }
    },

];

const data: DataType[] = [
    {
        key: '1',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
```

```
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
        key: '2',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
        key: '3',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
        key: '4',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
        key: '5',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
```

```
        key: '6',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
        key: '7',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },
    {
        key: '8',
        orderNo: 'CD9872380',
        date: "2024-02-13",
        carNo: '京A88888',
        type: "自有车辆",
        startDate: "2024-02-13 15:33:12",
        time: "2小时25分钟",
        count: "30kw",
        cost: "¥40.50"
    },

];

const columns2: TableProps<DataType2>['columns'] = [
    {
        title: "No.",
        key: "index",
        render: (text, record, index) => index + 1,
    },
    {
        title: '车牌号',
        dataIndex: 'carNo',
        key: 'carNo',

    },
    {
        title: '车主姓名',
```

```
        dataIndex: 'name',
        key: 'name',
    },
    {
        title: '车主电话',
        dataIndex: 'tel',
        key: 'tel',
    },
    {
        title: '租赁类型',
        dataIndex: 'type',
        key: 'type',

    },
    {
        title: '租期剩余',
        dataIndex: 'rest',
        key: 'rest',
    },
    {
        title: '超期天数',
        dataIndex: 'time',
        key: 'time',
    },
    {
        title: '入场照片',
        dataIndex: 'pic',
        key: 'pic',
        render: (text) => <Image
            src={come}
            width={50}
            placeholder={
                <Image
                    preview={false}
                    src={come}
                    width={150}
                />
            }
        />
    },

    {
        title: '操作',
        dataIndex: 'operate',
        key: 'operate',
        render: (text, record) => {
            return <>
                <Button type="primary" size="small" className='mr'>编辑</Button>
                <Button type="primary" size="small" danger>删除</Button>
```

```
            </>
        }
    },

];
const data2: DataType2[] = [
    {
        key: '1',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '2',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '3',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '4',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '5',
        carNo: '京A88888',
```

```
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '6',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '7',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },
    {
        key: '8',
        carNo: '京A88888',
        name: "王丽",
        tel: "18876543210",
        type: '长租车',
        rest: "135天",
        time: "0天",
        pic: "",
    },

];
const items:TabsProps['items']=[
    {
        key:"1",
        label:"充电记录",
        children:<Table columns={columns} dataSource={data}/>
    },
    {
        key:"2",
        label:"园内车辆列表",
        children:<Table columns={columns2} dataSource={data2}/>
```

```
    }
]

function Car(){
    return <div>
        <Card>
            <Row gutter={16}>
                <Col span={8}>
                    <Input placeholder="请输入车牌号、手机号或者联系人"/>
                </Col>
                <Col span={8}>
                    <Button type="primary" className="ml">查询</Button>
                </Col>
            </Row>
        </Card>
        <Card className="mt">
            <Tabs items={items}></Tabs>
        </Card>
    </div>
}

export default Car
```

20.报修管理页面开发

> 如果表格里的某个数据的渲染需要依赖该行数据其他的内容，可以使用render函数中的record，获取其他的数据，进行动态渲染

```
interface DataType {
    key: string;
    orderNo: string;
    name: string;
    tel: string;
    address: string;
    description: string;
    status: string;
    time: string;
}
const columns: TableProps<DataType>['columns'] = [
    {
        title: "No.",
        key: "index",
        render: (text, record, index) => index + 1,
    },
    {
        title: '维修单号',
        dataIndex: 'orderNo',
```

```
        key: 'orderNo',

    },
    {
        title: '报修人',
        dataIndex: 'name',
        key: 'name',
    },
    {
        title: '报修人电话',
        dataIndex: 'tel',
        key: 'tel',
    },
    {
        title: '报修地址',
        dataIndex: 'address',
        key: 'address',

    },
    {
        title: '故障描述',
        dataIndex: 'description',
        key: 'description',
    },
    {
        title: '维修状态',
        dataIndex: 'status',
        key: 'status',
        render:(text,record)=>{
            if(text==1){
                return  <Tag color="#f50">待维修</Tag>
            }else if(text==2){
                return  <Tag color="#2db7f5">维修中</Tag>
            }else{
                return  <Tag color="green">已完成</Tag>
            }
        }
    },
    {
        title: '报修时间',
        dataIndex: 'time',
        key: 'time',
    },
    {
        title: '操作',
        dataIndex: 'operate',
        key: 'operate',
        render: (text, record) => {
            if(record.status=="1"){
```

```
                return <>
                <Button type="primary" size="small">指派</Button>
            </>
            }else if(record.status=="2"){
                return <>
                 <a >维修中...</a>
            </>
            }else{
                return  <Button type="primary" size="small">完成</Button>
            }


        }
    },

];

const data: DataType[] = [
    {
        key: '1',
        orderNo: 'BX1236984',
        name: "刘宽",
        tel: '13498765432',
        address: "A2幢写字楼502",
        description: "空调制冷问题,间断性制冷，且制冷效果不佳",
        status: "1",
        time: "2024-05-30 13:37",
    },
    {
        key: '2',
        orderNo: 'BX1236984',
        name: "刘宽",
        tel: '13498765432',
        address: "A2幢写字楼502",
        description: "空调制冷问题,间断性制冷，且制冷效果不佳",
        status: "2",
        time: "2024-05-30 13:37",
    },
    {
        key: '3',
        orderNo: 'BX1236984',
        name: "刘宽",
        tel: '13498765432',
        address: "A2幢写字楼502",
        description: "空调制冷问题,间断性制冷，且制冷效果不佳",
        status: "3",
        time: "2024-05-30 13:37",
    },
    {
```

```
        key: '4',
        orderNo: 'BX1236984',
        name: "刘宽",
        tel: '13498765432',
        address: "A2幢写字楼502",
        description: "空调制冷问题,间断性制冷，且制冷效果不佳",
        status: "1",
        time: "2024-05-30 13:37",
    },
    {
        key: '5',
        orderNo: 'BX1236984',
        name: "刘宽",
        tel: '13498765432',
        address: "A2幢写字楼502",
        description: "空调制冷问题,间断性制冷，且制冷效果不佳",
        status: "3",
        time: "2024-05-30 13:37",
    },
    {
        key: '6',
        orderNo: 'BX1236984',
        name: "刘宽",
        tel: '13498765432',
        address: "A2幢写字楼502",
        description: "空调制冷问题,间断性制冷，且制冷效果不佳",
        status: "2",
        time: "2024-05-30 13:37",
    },
```

21.合同管理页面开发

> 缓存页面的实现逻辑

1.列表页面获取数据，将数据存储到redux中，存储到redux中的原因是为了下次再进入这个页面直接从redux中取，而不需要重新加载，达到缓存的目的

2.如果是从详情页返回列表页，那么列表页数据应该缓存，如果是从其他页面进入列表页，那么列表页应该刷新

```
    useEffect(() => {
        if(!data.length|| (!isReturn)){loadData()}
    }, []);
```

> 列表页跳转到详情页携带数据

```
navigate("/finance/surrender?contractNo="+contractNo)
```

```
const [searchParams, setSearchParams] = useSearchParams();
const searchValue = searchParams.get('contractNo');
```

> 详情页返回列表页携带标记，列表页可以判断是否是从详情页返回

```
<Button type="primary" onClick={()=>navigate("/finance/contract?return=true")}>返回
</Button>
```

```
const isReturn = searchParams.get('return');
```

22.账单管理页面开发

1.导出excel核心实现

1.安装

```
npm install xlsx file-saver
```

1. 利用`xlsx`库将这些数据转换为Excel文件。
2. 使用`file-saver`库来触发文件下载。

2.引入

```
import * as XLSX from 'xlsx';
import { saveAs } from 'file-saver';
```

3.核心代码

```
const exportToExcel = () => {
  // 使用XLSX的工具函数json_to_sheet将JSON数据转换为工作表
  // 这里的selectedRows是包含已选表格行的数据数组
  const ws = XLSX.utils.json_to_sheet(
    selectedRows,
    {
```

```
      // 指定导出的列头，确保selectedRows中的对象具有这些属性
      // 这也定义了列的顺序，以及Excel中显示的列
      header: ['accountNo', 'status', 'roomNo', 'carNo', 'tel', 'costName1',
'costName2', 'costName3', 'startDate', 'endDate', 'preferential', 'money', 'pay']
    }
  );

  // 使用XLSX的工具函数book_new创建一个新的工作簿
  const wb = XLSX.utils.book_new();

  // 将上面创建的工作表添加到工作簿中
  // 第一个参数是工作簿对象，第二个参数是工作表对象，第三个参数是这个工作表的名称
  XLSX.utils.book_append_sheet(wb, ws, "Sheet1");

  // 生成Excel文件的二进制内容
  // bookType指定文件类型，'xlsx'为Excel 2007+格式
  // type:'buffer'指定返回的是一个ArrayBuffer对象，用于在浏览器中处理二进制数据
  const buf = XLSX.write(wb, {bookType:'xlsx', type:'buffer'});

  // 使用file-saver库中的saveAs函数保存生成的Excel文件
  // 第一个参数是一个Blob对象，它封装了要下载的数据内容
  // 第二个参数是下载文件的名称
  saveAs(new Blob([buf],{type:"application/octet-stream"}), "selected-data.xlsx");
};
```

2.导出pdf核心实现

1.安装

```
npm install antd @react-pdf/renderer
```

2.创建PDF文档组件

```
import React from 'react';
import { Page, Text, View, Document, StyleSheet } from '@react-pdf/renderer';

const styles = StyleSheet.create({
  page: {
    flexDirection: 'column',
    backgroundColor: '#E4E4E4'
  },
  section: {
    margin: 10,
    padding: 10,
    flexGrow: 1
  }
```

```
});

// 创建文档组件
const BillPDF = ({ items }) => (
  <Document>
    <Page size="A4" style={styles.page}>
      {items.map((item, index) => (
        <View style={styles.section} key={index}>
          <Text>账单号：{item.accountNo}</Text>
          <Text>缴费状态：{item.status === 1 ? '已缴费' : '未缴费'}</Text>
          <Text>房屋号：{item.roomNo}</Text>
          <Text>车位号：{item.carNo}</Text>
          <Text>手机号：{item.tel}</Text>
          <Text>物业费(年)：{item.costName1}</Text>
          <Text>车位费：{item.costName2}</Text>
          <Text>房屋租金：{item.costName3}</Text>
          <Text>开始时间：{item.startDate}</Text>
          <Text>结束时间：{item.endDate}</Text>
          <Text>优惠金额：{item.preferential}</Text>
          <Text>合计应收金额：{item.money}</Text>
          <Text>支付方式：{item.pay}</Text>
        </View>
      ))}
    </Page>
  </Document>
);

export default BillPDF;
```

3. 集成PDF导出到主组件

```
import React, { useState, useEffect } from 'react';
import { Button } from 'antd';
import { PDFDownloadLink } from '@react-pdf/renderer';
import BillPDF from './BillPDF'; // 引入刚才创建的组件

function Bill() {
  const [selectedData, setSelectedData] = useState([]);
  const [dataList, setDataList] = useState([]);

  // 加载数据等其他代码...

  const exportPDFButton = useMemo(() => (
    <PDFDownloadLink
      document={<BillPDF items={selectedData} />}
      fileName="selected-bills.pdf"
    >
      {({ blob, url, loading, error }) =>
        loading ? '加载文档...' : '导出PDF'
```

```
      }
    </PDFDownloadLink>
  ), [selectedData]);

  return (
    <div>
      {/* 其他 UI 和表格代码 */}
      {selectedData.length > 0 && exportPDFButton}
    </div>
  );
}

export default Bill;
```

3.表格跨页选择

核心是配置 rowSelection

```
const rowSelection={
    preserveSelectedRowKeys:true
  }
```

23.设备管理页面开发

关于自定义hooks的实现

```
import { useState, useEffect, useCallback } from 'react';



// FormData类型需要在调用时传入具体类型，以增强灵活性和重用性
type FormData = {
    [key: string]: any; // 使用索引签名，具体类型由调用时定义
};

// 定义一个函数类型 DataFetcher，用于执行数据的获取任务。
interface DataFetcher<T> {
    (args: T & { page: number; pageSize: number }): Promise<any>;
}

// 自定义Hook的实现
function useDataList<T extends FormData,U>(initialFormData: T, fetchData:
DataFetcher<T>) {
    const [dataList, setDataList] = useState<U[]>([]);
    const [page, setPage] = useState<number>(1);
    const [pageSize, setPageSize] = useState<number>(10);
    const [total, setTotal] = useState<number>(0);
    const [loading, setLoading] = useState<boolean>(false);
```

```
    const [formData, setFormData] = useState<T>(initialFormData);

    const loadData = useCallback(async () => {
        console.log("canshushi",formData)
        setLoading(true);
        try {
            const { data:{list,total}  } = await fetchData({ ...formData, page,
pageSize });
            setDataList(list);
            setTotal(total);
        } catch (error) {
            console.error('Failed to load data:', error);
        } finally {
            setLoading(false);
        }
    }, [ formData,page, pageSize, fetchData]);

    useEffect(() => {
        loadData();
    }, [loadData]);

    const onChange = (pageNumber: number, pageSize: number) => {
        setPage(pageNumber);
        setPageSize(pageSize);
    };

    // 通用的输入变化处理器
    const handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = event.target;

        console.log(777,name,value)
        setFormData(prevState => ({
            ...prevState,
            [name]: value
        }));

    };

    return {
        dataList,
        page,
        pageSize,
        total,
        loading,
        formData,
        setDataList,
        setPage,
        setPageSize,
        setTotal,
```

```
        setLoading,
        setFormData,
        loadData,
        onChange,
        handleChange
    };
}

export default useDataList;
```

24.系统设置页面开发

> 重点方法

```
function extractTreeKeys(data:any){
    let keys:string[]=[];
    data.forEach((item:any)=>{
        if(item.children&&item.children.length>0){
            const childKeys:string[]=extractTreeKeys(item.children);
            keys=keys.concat(childKeys)
        }else{
            keys.push(item.key)
        }
    })
    return keys
}
```

因为我们菜单项数据和tree组件中要求的数据不一样，我们需要对菜单数据进行转换

一个关键的问题是，我们应该只保存子集的key，而不应该保存父级的key，因为一旦保存了父级的key，子集会被都选上

所以这里用了递归操作，将所有的key存到一个数组中

25.个人中心页面开发

26.按钮级别的权限控制

> 1.封装一个高阶组件控制按钮的显示

```
// 定义高阶组件函数使用 TypeScript 泛型
```

```
function withPermissions(requiredPermissions:string[],userPermissions:string[]):
(Component: React.FC) => React.FC {
    return function (Component: React.FC): React.FC {
        return function (props: any): React.ReactElement | null {
            // 检查用户是否具有所有必要的权限
            const hasPermission: boolean = requiredPermissions.every(permission =>
userPermissions.includes(permission));
            console.log(88,hasPermission)
            if (!hasPermission) {
                // 如果用户没有权限，返回null或权限不足的提示
                return null; // 或者返回 <div>No permission</div>
            }

            // 如果有权限，渲染传入的组件
            return <Component {...props} />;
        };
    };
}

export default withPermissions;
```

> 2.修改mock和登录接口，返回按钮权限信息，并存储到本地存储

```
if (username == 'admin' && password == 'admin123123') {
    return {
      code: 200,
      message: '登录成功',
      data: {
        userId: 1,
        username: 'admin',
        token: 'mocktoken123456admin',
        btnAuth:["add","edit","delete"]
      }
    }
```

```
function handleLogin() {
        form.validateFields().then(async res => {
            setLoading(true)
            const { data: { token,btnAuth } } = await login(res);
            console.log(11,btnAuth)
            sessionStorage.setItem("btnAuth",JSON.stringify(btnAuth))
            dispatch(setToken(token))
            navigate("/dashboard", { replace: true });
        }).catch((error: any) => {
            setLoading(false)
            console.log("error", error)
        })
    }
```

3.使用

```
const
AuthButton:React.FC<any>=withPermissions(['delete'],JSON.parse(sessionStorage.getItem("
btnAuth") as string))(Button)
```

```
<AuthButton size="small" danger type="primary">删除账号</AuthButton>
```

27.项目中的环境变量

配置环境变量可以非常方便地帮助你在不同的开发阶段（例如开发、测试和生产）管理不同的配置。React 官方已经在 `create-react-app` 中内置了对环境变量的基础支持

> **1.在项目根目录下，你可以创建特定的文件来设定环境变量：**

- `.env`：默认环境变量文件，适用于所有环境。
- `.env.local`：当地开发环境专用文件，通常不会被提交到版本控制系统。
- `.env.development`，`.env.test`，`.env.production`：分别用于开发、测试、生产环境。

> **2.定义环境变量**
>
> 在这些文件中定义变量时，变量名必须以 `REACT_APP_` 开头。例如：

```
REACT_APP_API_URL=https://api.example.com
REACT_APP_SECRET_CODE=abcdefgh
```

这个前缀 REACT_APP_ 是必需的，因为 create-react-app 的环境变量加载方案只会加载以 REACT_APP_ 开头的变量，以避免意外地暴露敏感数据。

> **3.使用环境变量**

> 在你的React代码中，你可以使用 `process.env` 来访问这些变量。例如：

```
console.log('API URL:', process.env.REACT_APP_API_URL);
```

> **4.区分环境变量**

为了在不同的环境（例如，本地开发环境和生产环境）中使用不同的环境变量，你应该根据当前的工作环境调整相应的 `.env` 文件。比如：

- 在 `.env.development` 文件中，可以设置 `REACT_APP_API_URL=https://dev-api.example.com`
- 而在 `.env.production` 文件中，设置 `REACT_APP_API_URL=https://api.example.com`

> **5: 启动和构建项目**

当你启动或构建你的应用时，`create-react-app` 的内置 Webpack 配置将会自动根据当前的 `NODE_ENV` 值加载对应的环境变量文件。例如：

```
npm start # 自动使用 .env.development
npm test # 自动使用 .env.test
npm run build # 自动使用 .env.production
```

28.项目的打包部署

1.项目运行npm run build打包

2.阿里云购买服务器

3.安装宝塔面板

4.安全组放行宝塔面板的端口号

5.宝塔面板中新建站点

6.将我们的项目传到服务器根目录下

7.修改nginx配置解决刷新404

```
server {
    listen 80;
    server_name your-domain.com; # 替换为您的域名

    index index.html;
    root /path/to/your/react/app/build; # 修改为您的实际React应用构建目录

    location / {
        try_files $uri $uri/ /index.html;
    }

}
```