同济大学

TONGJI UNIVERSITY

《WEB 技术》

实验报告（大作业）

实验名称	实验二 前端设计
小组成员	黄保翔（2351753）
学院（系）	计算机科学与技术学院
专　　业	计算机科学与技术
任课教师	郭玉臣
日　　期	2025 年　5 月　3　日

实验原理

电梯监控网站前端通过现代前端开发技术，实现了电梯运行状态、传感器数据和网络信息的实时展示与动态交互。本实验旨在构建一个用户友好的界面，支持多标签页切换（如电梯详细信息、联网信息等），并通过模块化组件化和响应式设计提升开发效率和用户体验。其核心原理基于以下技术和工具：

1. **Vue.js 框架**：Vue.js 是一个渐进式 JavaScript 框架，用于构建用户界面。本实验采用 Vue 3，利用其响应式数据绑定和组件化特性：

   响应式数据绑定：通过 Vue 的 reactive 和 ref 机制，电梯的传感器数据（如楼层、人次、功率、温度）和联网信息（如 IP 地址、信号强度）能够实时更新并反映到界面上。例如，elevatorDetail 和 sensorData 对象的变化会自动触发 DOM 更新。

   组件化开发：界面被拆分为多个独立组件（如 ElevatorInfo.vue 和 NetworkInfo.vue），每个组件负责特定功能（如展示电梯基本信息或联网状态）。通过 props 传递数据，组件之间实现松耦合，提高代码可维护性和复用性。

   动态组件切换：利用 Vue 的 <component :is> 指令，通过状态变量 activeTab 动态渲染不同标签页内容(如“电梯详细信息”或“联网信息”)，无需页面刷新即可切换视图。

2. **Vite 构建工具**：Vite 是一个高性能的前端构建工具，基于 ES 模块（ESM）提供快速开发和构建体验。本实验使用 Vite 作为开发服务器和打包工具：

   快速开发服务器：Vite 利用浏览器原生的 ES 模块支持，通过按需加载模块实现热模块替换（HMR），开发者修改代码后界面可即时更新，显著提升开发效率。

   高效打包：Vite 使用 Rollup 进行生产环境打包，生成优化的静态资源，支持代码分割和 Tree Shaking，减小最终 bundle 体积，提升页面加载速度。

3. **Vue Router（可选）**：Vue Router 是 Vue.js 的官方路由管理库，用于实现页面导航。本实验中，路由用于从电梯列表页面跳转到电梯详情页面（通过 deviceId 参数），并支持返回功能（router.push）。动态组件切换替代了嵌套路由，简化了标签页切换逻辑。

4. **数据管理**：前端通过模拟数据（或未来与后端 API 交互）管理电梯信息：
   - **模拟数据**：使用 JavaScript 对象（如 elevatorData 和 networkDataMap）模拟电梯和联网信息，基于 deviceId 动态加载对应数据。例如，E01801-010-001 对应的联网状态为“在线”，IP 地址为“192.168.1.101”。
   - **动态更新**：通过 Vue 的响应式系统，数据变化（如信号强度低于 30%）会触发界面更新，结合条件样式（如红色高亮）提示异常。
   - **未来扩展**：可通过 HTTP 请求（如 axios）与后端 API 交互，获取实时数据，替换模拟数据。

实验步骤

1.建立 vue 框架

使用 pycharm 安装 vue 插件，更新 node.js 到 20 版本以适应最新版的 vite 打包工具

完成后对原目录进行修改

2.下载 router 插件

为了方便在不同页面之间进行切换，使用 router 插件完成

npm install – save router

然后配置 router.js 在 router/index.js 中加入需要跳转的路由

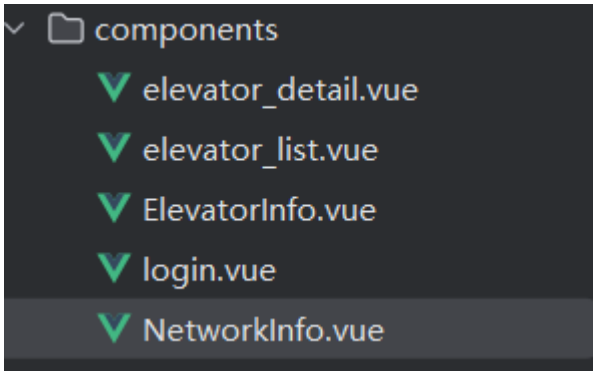
3.配置 main.js

```
import router from './router/index.js'

const app = createApp(App)
app.use(router)
app.mount('#app')
```

4.进行不同页面的编写

App.vue 作为根文件，设置以下文件



其中 login 作为登录界面，elevator_list 界面作为多个电梯信息查询界面，elevator_detail 作为详情导航栏，通过动态组件的方式在不同的界面之间进行切换，这里只完成了 Elevator_Info 和 NetworkInfo 界面

5.开发编译

npm run dev

6.完成后打包

npm run build

实验内容

1.登录界面

```
methods: {
  login() {
    if (this.username && this.password) {
      if(this.username === "root" && this.password === '123456') {
        this.$router.push({ name: 'elevator_list' });
      }
      else{
        alert("用户名或密码错误！");
      }
    }
    else{
      alert("用户名或密码不能为空！");
    }

  }
}
```

只设置一个用户名和信息，通过简单的逻辑完成

2.列表界面

```
<tr v-for="(elevator, index) in filteredElevators" :key="index">
  <td>{{ index + 1 }}</td>
  <td>{{ elevator.deviceId }}</td>
  <td>{{ elevator.location }}</td>
  <td :class="getStatusClass(elevator.status)">{{ elevator.status }}</td>
  <td>{{ elevator.lastUpdated }}</td>
  <td>
```

```
computed: {
  filteredElevators() {
    return this.elevators.filter(elevator => {
      const statusMatch = !this.searchStatus || elevator.status.includes(this.searchStatus);
      const dateMatch = !this.searchDate || elevator.lastUpdated.startsWith(this.searchDate);
      return statusMatch && dateMatch;
    });
  },
},
```

使用计算属性，通过 filter 进行返回所有符合要求的 item

3.详情界面

```
<span
  class="tab"
  :class="{ active: activeTab === 'info' }"
  @click="activeTab = 'info'"
>电梯详细信息</span>
```

通过导航栏改变 activeTab 的值

```
<component
  :is="currentComponent"
  :elevator-detail="elevatorDetail"
  :sensor-data="sensorData"
/>
```

使用动态组件的方式，调用对应的组件界面

对用不同的设备，使用动态路由

```
{
  path: '/elevator_detail/:deviceId',
  name: 'ElevatorDetail',
  component: () => import('../components/elevator_detail.vue'),
  props: true
}
```

点击详情时，将对应的设备 id 传递到路由当中，在传递给对应的详情界面，找到相应的模拟数据进行显示

4.根组件

在 App.vue 中使用

```
<template>
  <router-view/>
</template>
```

进行占位

在 router/index.js 中配置相应路由

```
const routes = [
  {
    path: '/',
    name: 'login',
    component: () => import('../components/login.vue')
  },
  {
    path: '/elevator_list',
    name: 'elevator_list',
    component: () => import('../components/elevator_list.vue')
  },
  {
```

```
    path: '/elevator_detail/:deviceId',
    name: 'ElevatorDetail',
    component: () => import('../components/elevator_detail.vue'),
    props: true
  }

]
```

心得体会

通过本次电梯监控网站前端开发的实验，我深入学习并实践了现代前端开发技术，包括 Vue.js、Vite 等工具，收获颇丰，同时也遇到了一些挑战。在实验中，我通过将电梯监控界面拆分为多个组件（如 ElevatorInfo.vue 和 NetworkInfo.vue）实现了模块化开发。这种方式极大地提高了代码的可维护性和复用性。例如，ElevatorInfo.vue 负责展示传感器数据，NetworkInfo.vue 专注于联网信息，二者通过 props 接收数据，逻辑清晰且互不干扰。使用 <component :is> 实现动态标签页切换让我体会到 Vue 的灵活性，activeTab 状态的切换不仅简化了交互逻辑，还提升了用户体验。本次实验让我从零开始构建了一个功能完善的电梯监控前端界面，不仅巩固了 Vue 和 Vite 的使用，还让我对前端开发的整体流程有了更全面的认识。