

课程名称：前端程序设计

期末考核内容：

- 开发一个小型信息管理系统（题目自己定），要求设计登录页面、主页面，主页面中有功能菜单，菜单应有响应，能与后端程序交互数据。在课程结束2周后答辩（直接交报告），根据页面数、完整性、美观和交互等打分。撰写课程设计报告（格式不限，但需美观易读），报告中需要包含所有的核心代码。
- 如果使用了大模型，请详细描述：
  - 原始目录结构
  - 每一步提示语句（Prompt语句）和执行后的效果（截图）。

班级：信息安全实验221班

学号：2022132006

姓名：杨佳倪

提交日期：2025年4月7日

## 主要目录结构

```
src/
├── assets/
│   └── vue.svg
├── mock/
│   └── mock.js
├── views/
│   ├── LoginView.vue
│   └── StudentView.vue
├── App.vue
├── router.js
├── style.css
└── main.js
```

主要代码：

## mock.js

这段代码主要实现了一个基于 Mock.js 的学生信息管理系统的模拟后端 API，主要功能包括：

### 1. 初始化学生数据：

- 使用 Mock.js 生成 100 条模拟学生数据（如果 sessionStorage 中没有数据）
- 每条学生数据包含：学号(id)、姓名(name)、性别(gender)、年龄(age)、专业(major)和班级(class)
- 数据存储在浏览器的 sessionStorage 中

### 2. 登录功能：

- 提供 `/api/login` POST 接口
- 验证用户名和密码（硬编码为 admin/123456）
- 返回登录成功或失败的响应

### 3. 学生数据 CRUD 操作：

- 获取学生列表：

- GET `/api/students` 接口
- 支持分页(page/pageSize)、搜索(searchQuery)和排序(sortField/sortOrder)
- 新增学生:
  - POST `/api/students` 接口
  - 自动生成学号
- 编辑学生:
  - PUT `/api/students/:id` 接口
  - 更新学生信息 (保留原学号)
- 删除学生:
  - DELETE `/api/students/:id` 接口
  - 根据学号删除学生

#### 4. 数据持久化:

- 所有操作都会同步更新到 sessionStorage
- 使用 `saveToSessionStorage()` 函数统一处理存储逻辑

#### 5. 辅助功能:

- `getNextId()` 函数用于生成新学生的学号

这个代码主要用于前端开发时的模拟 API 服务, 让前端可以在没有真实后端的情况下进行开发和测试, 实现了完整的学生信息管理功能, 包括列表展示、搜索、排序、分页以及增删改查等操作。

```
import Mock from "mockjs";

// ===== 初始化学生数据 =====
let students = JSON.parse(sessionStorage.getItem("students")) || [];

if (!students.length) {
  students = Mock.mock({
    'students|100': [
      {
        id: '@increment',
        name: '@cname',
        gender: '@pick(["男", "女"])',
        'age|18-25': 1,
        major: '@ctitle(3, 5)',
        class: '@string("number", 2)-@string("number", 1)'
      },
    ],
  }).students;

  saveToSessionStorage();
}

// 保存数据到 sessionStorage 的函数
function saveToSessionStorage() {
  sessionStorage.setItem("students", JSON.stringify(students));
}

// 获取下一个唯一学号
function getNextId() {
  return students.length > 0 ? Math.max(...students.map((s) => s.id)) + 1 : 1;
}

// ===== 登录接口 =====
Mock.mock("/api/login", "post", (options) => {
```

```

const body = JSON.parse(options.body);
if (body.username === "admin" && body.password === "123456") {
  return { code: 200, message: "登录成功", token: "mock-token" };
}
return { code: 401, message: "用户名或密码错误" };
});

// ===== 获取学生列表接口 =====
Mock.mock(/\/api\/students/, "get", (req) => {
  const urlParams = new URLSearchParams(req.url.split("?")[1]);
  const page = parseInt(urlParams.get("page")) || 1;
  const pageSize = parseInt(urlParams.get("pageSize")) || 10;
  const searchQuery = urlParams.get("searchQuery") || "";
  const sortField = urlParams.get("sortField") || "";
  const sortOrder = urlParams.get("sortOrder") || "asc";

  let filtered = students;

  // 支持学号和姓名搜索
  if (searchQuery) {
    filtered = filtered.filter((student) => {
      const idString = String(student.id); // 学号转为字符串
      return student.name.includes(searchQuery) ||
idString.includes(searchQuery);
    });
  }

  console.log("搜索关键词: ", searchQuery);
  console.log("过滤后的数据: ", filtered);

  // 排序
  if (sortField) {
    filtered.sort((a, b) => {
      if (sortOrder === "asc") {
        return a[sortField] > b[sortField] ? 1 : -1;
      } else {
        return a[sortField] < b[sortField] ? 1 : -1;
      }
    });
  }

  // 分页
  const start = (page - 1) * pageSize;
  const end = page * pageSize;
  const paginatedStudents = filtered.slice(start, end);

  return {
    code: 200,
    data: {
      total: filtered.length,
      students: paginatedStudents,
    },
    message: "获取成功",
  };
});

// ===== 新增学生接口 =====
Mock.mock("/api/students", "post", (req) => {

```

```
const newStudent = JSON.parse(req.body);

// 自动生成唯一学号
newStudent.id = getNextId();

students.push(newStudent);
saveToSessionStorage(); // 同步到 SessionStorage

return { code: 200, message: "新增成功", student: newStudent };
});

// ===== 编辑学生接口 =====
Mock.mock(/\/api\/students\/\d+/, "put", (req) => {
  const updatedStudent = JSON.parse(req.body);
  const studentId = parseInt(req.url.split("/").pop());

  // 查找学生
  const oldIndex = students.findIndex((s) => s.id === studentId);
  if (oldIndex === -1) {
    return { code: 404, message: "学生不存在" };
  }

  // 更新除学号以外的字段
  students[oldIndex] = {
    ...students[oldIndex],
    ...updatedStudent,
    id: students[oldIndex].id, // 确保学号不变
  };

  saveToSessionStorage(); // 同步到 SessionStorage

  return { code: 200, message: "编辑成功" };
});

// ===== 删除学生接口 =====
Mock.mock(/\/api\/students\/\d+/, "delete", (req) => {
  const studentId = parseInt(req.url.split("/").pop());

  // 过滤掉指定 ID 的学生
  const newStudents = students.filter((s) => s.id !== studentId);

  if (newStudents.length === students.length) {
    return { code: 404, message: "学生不存在" };
  }

  students = newStudents;
  saveToSessionStorage(); // 同步到 SessionStorage

  return { code: 200, message: "删除成功" };
});
```

# views/LoginView.vue

这段代码实现了一个 **登录页面** 的功能，基于 Vue 3（`<script setup>` 语法）和 Element Plus 组件库。以下是主要功能分析：

## 1. 页面结构 (Template)

- 整体布局：**
  - 使用 `el-card` 作为登录卡片容器，居中显示在页面上。
  - 包含一个标题（`<h2>登录</h2>`）和一个表单（`el-form`）。
- 表单内容：**
  - 用户名输入框：** `el-input` 绑定到 `form.username`，提示文字为“请输入用户名”。
  - 密码输入框：** `el-input` 绑定到 `form.password`，类型为 `password`（隐藏输入）。
  - 登录按钮：** `el-button` 触发 `handleLogin` 方法。

## 2. 逻辑功能 (Script)

- 响应式数据：**
  - 使用 `reactive` 定义表单数据 `form`，包含 `username` 和 `password` 字段。
- 登录逻辑：**
  - 请求接口：** 点击按钮时，调用 `handleLogin` 方法，通过 `axios` 发送 POST 请求到 `/api/login`，携带用户名和密码。
  - 成功处理：**
    - 如果返回 `code === 200`，显示成功消息（`ElMessage.success`）。
    - 将 `token` 存储到 `localStorage`。
    - 跳转到 `/students` 路由（学生列表页）。
  - 失败处理：**
    - 如果用户名或密码错误，显示错误消息（`ElMessage.error`）。
- 依赖工具：**
  - `vue-router`：用于页面跳转（`router.push`）。
  - `element-plus`：提供 UI 组件（`ElMessage`、`ElCard`、`ElForm` 等）。

## 3. 样式 (Style)

- 登录容器：**
  - 使用 Flex 布局使卡片垂直水平居中。
  - 设置背景色为浅灰色（`#f5f5f5`）。
- 登录卡片：**
  - 固定宽度 `300px`，内边距 `20px`，文本居中。

```
<template>
  <div class="login-container">
    <el-card class="login-card">
      <h2>登录</h2>
      <el-form :model="form" label-width="80px">
        <el-form-item label="用户名">
          <el-input v-model="form.username" placeholder="请输入用户名" />
        </el-form-item>
      </el-form>
    </el-card>
  </div>
</template>
```

```

    </el-form-item>
    <el-form-item label="密码">
      <el-input
        v-model="form.password"
        placeholder="请输入密码"
        type="password"
      />
    </el-form-item>
    <el-form-item>
      <el-button type="primary" @click="handleLogin">登录</el-button>
    </el-form-item>
  </el-form>
</el-card>
</div>
</template>

```

```

<script setup>
import { useRouter } from "vue-router";
import axios from "axios";
import { ElMessage } from "element-plus";
import { reactive } from "vue";

const router = useRouter();
const form = reactive({
  username: "",
  password: "",
});

const handleLogin = async () => {
  const res = await axios.post("/api/login", form);
  if (res.data.code === 200) {
    ElMessage.success(res.data.message);
    localStorage.setItem("token", res.data.token);
    router.push("/students");
  } else {
    ElMessage.error(res.data.message);
  }
};
</script>

```

```

<style scoped>
.login-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100%;
  background-color: #f5f5f5;
}

.login-card {
  width: 300px;
  padding: 20px;
  text-align: center;
}
</style>

```

# views/StudentView.vue

## 1. 学生列表展示

- 表格显示学号、姓名、性别、年龄、专业、班级等信息。
- 支持 **分页** (`el-pagination`)、**搜索** (按学号或姓名)、**排序** (点击表头)。

## 2. 增删改查 (CRUD)

- **新增学生**: 点击“新增”按钮, 填写表单提交。
- **编辑学生**: 点击“编辑”按钮, 修改信息后保存。
- **删除学生**: 点击“删除”按钮, 确认后移除数据。

## 3. 用户交互

- 搜索框实时过滤数据。
- 弹窗表单 (`el-dialog`) 用于新增/编辑, 带表单验证 (必填字段)。
- 操作成功/失败提示 (`ElMessage`)。

## 4. 登出功能

- 清除本地 token 并跳转至登录页。

## 5. 数据管理

- 通过 `axios` 与后端 API 交互 (`/api/students`)。
- 响应式更新表格数据 (Vue 3 的 `reactive/ref`)。

```
<template>
  <div class="students-container">
    <!-- 卡片容器 -->
    <el-card>
      <!-- 顶部搜索和新增按钮 -->
      <div class="header">
        <!-- 搜索框 -->
        <el-input
          v-model="searchQuery"
          placeholder="搜索学号或姓名"
          prefix-icon="el-icon-search"
          @input="handleSearch"
        />
        <!-- 新增按钮 -->
        <el-button type="primary" @click="openDialog('add')"
          >新增学生</el-button>
      </div>
      <!-- 登出按钮 -->
      <el-button type="danger" @click="logout">登出</el-button>
    </div>

    <!-- 学生表格 -->
    <el-table
      :data="filteredStudents"
      style="width: 100%"
      border
      @sort-change="handleSortChange"
    >
      <el-table-column prop="id" label="学号" width="100" sortable="custom" />
      <el-table-column prop="name" label="姓名" sortable="custom" />
      <el-table-column prop="gender" label="性别" sortable="custom" />
      <el-table-column
        prop="age"
        label="年龄"
      />
    </el-table>
  </div>
</template>
```

```

        width="100"
        sortable="custom"
      />
    <el-table-column prop="major" label="专业" sortable="custom" />
    <el-table-column prop="class" label="班级" sortable="custom" />
    <el-table-column label="操作" width="180">
      <!-- 操作按钮 -->
      <template #default="scope">
        <el-button
          size="small"
          @click="openDialog('edit', scope.row)"
          type="primary"
        >
          编辑
        </el-button>
        <el-button
          size="small"
          @click="deleteStudent(scope.row.id)"
          type="danger"
        >
          删除
        </el-button>
      </template>
    </el-table-column>
  </el-table>

  <!-- 分页组件 -->
  <div class="block">
    <el-pagination
      background
      layout="prev, pager, next"
      :total="totalStudents"
      :page-size="pageSize.value"
      :current-page.sync="currentPage.value"
      @current-change="handlePageChange"
    />
  </div>
</el-card>

<!-- 弹窗表单 -->
<el-dialog
  :title="dialogMode === 'add' ? '新增学生' : '编辑学生'"
  v-model="dialogVisible"
>
  <el-form
    :model="dialogForm"
    :rules="rules"
    label-width="80px"
    ref="formRef"
  >
    <el-form-item label="学号" prop="id">
      <!-- 新增模式下允许输入，编辑模式下设置为只读 -->
      <el-input
        v-model="dialogForm.id"
        :readonly="dialogMode === 'edit'"
        placeholder="请输入学号"
      />
    </el-form-item>
  </el-form>

```

```

    <el-form-item label="姓名" prop="name">
      <el-input v-model="dialogForm.name" />
    </el-form-item>
    <el-form-item label="性别" prop="gender">
      <el-select v-model="dialogForm.gender" placeholder="请选择性别">
        <el-option label="男" value="男" />
        <el-option label="女" value="女" />
      </el-select>
    </el-form-item>
    <el-form-item label="年龄" prop="age">
      <el-input v-model="dialogForm.age" type="number" />
    </el-form-item>
    <el-form-item label="专业" prop="major">
      <el-input v-model="dialogForm.major" />
    </el-form-item>
    <el-form-item label="班级" prop="class">
      <el-input v-model="dialogForm.class" />
    </el-form-item>
  </el-form>
  <template #footer>
    <el-button @click="dialogVisible = false">取消</el-button>
    <el-button type="primary" @click="saveStudent">保存</el-button>
  </template>
</el-dialog>
</div>
</template>

```

```

<script setup>
import { reactive, ref, computed, onMounted, watch } from "vue";
import axios from "axios";
import { ElMessage } from "element-plus";
import { useRouter } from "vue-router";

// ===== 状态变量 =====
const students = reactive([]); // 学生数据
const searchQuery = ref(""); // 搜索关键字
const currentPage = ref(1); // 当前页码
const pageSize = ref(10); // 每页显示条数
const totalStudents = ref(0); // 学生总数
const sortField = ref(""); // 排序字段
const sortOrder = ref(""); // 排序顺序
const dialogVisible = ref(false); // 弹窗是否可见
const dialogMode = ref("add"); // 弹窗模式（新增/编辑）
const formRef = ref(null); // 表单引用
const router = useRouter(); // 路由

// 弹窗表单初始数据
const dialogForm = reactive({
  name: "",
  gender: "",
  age: null,
  major: "",
  class: "",
});

// 表单验证规则
const rules = {
  name: [{ required: true, message: "请输入姓名", trigger: "blur" }],

```

```

gender: [{ required: true, message: "请选择性别", trigger: "change" }],
age: [{ required: true, message: "请输入年龄", trigger: "blur" }],
major: [{ required: true, message: "请输入专业", trigger: "blur" }],
class: [{ required: true, message: "请输入班级", trigger: "blur" }],
};

// ===== 数据获取 =====

// 计算属性：按搜索关键字过滤学生数据
const filteredStudents = computed(() => {
  if (!searchQuery.value) return students; // 如果没有搜索关键字，返回所有数据
  return students.filter((student) => {
    return (
      student.name.includes(searchQuery.value) ||
      String(student.id).includes(searchQuery.value)
    ); // 支持学号搜索
  });
});

// 获取学生列表
const fetchStudents = async () => {
  try {
    const response = await axios.get("/api/students", {
      params: {
        page: currentPage.value,
        pageSize: pageSize.value,
        searchQuery: searchQuery.value,
        sortField: sortField.value,
        sortOrder: sortOrder.value,
      },
    });
    if (response.data.code === 200) {
      students.splice(0, students.length, ...response.data.data.students);
      totalStudents.value = response.data.data.total;
    }
  } catch (error) {
    ElMessage.error("获取学生数据失败");
  }
};

// ===== 事件处理 =====

// 排序变更
const handleSortChange = ({ prop, order }) => {
  sortField.value = prop;
  sortOrder.value = order === "ascending" ? "asc" : "desc";
  fetchStudents();
};

// 搜索变更
const handleSearch = () => {
  currentPage.value = 1;
  fetchStudents();
};

// 分页变更
const handlePageChange = (newPage) => {
  currentPage.value = newPage;
  fetchStudents();
};

```

```

};

// 打开弹窗
const openDialog = (mode, student = null) => {
  dialogMode.value = mode;
  dialogVisible.value = true;
  Object.assign(dialogForm, {
    id: null,
    name: "",
    gender: "",
    age: null,
    major: "",
    class: "",
  }); // 重置表单
  if (mode === "edit" && student) Object.assign(dialogForm, student); // 编辑模式
  填充数据
};

// 保存学生信息
const saveStudent = async () => {
  formRef.value.validate(async (valid) => {
    if (valid) {
      try {
        if (dialogMode.value === "add") {
          await axios.post("/api/students", { ...dialogForm });
          ElMessage.success("新增成功");
        } else {
          await axios.put(`/api/students/${dialogForm.id}`, { ...dialogForm });
          ElMessage.success("编辑成功");
        }
        dialogVisible.value = false;
        fetchStudents();
      } catch (error) {
        ElMessage.error("操作失败");
      }
    }
  });
};

// 删除学生
const deleteStudent = async (id) => {
  try {
    await axios.delete(`/api/students/${id}`);
    ElMessage.success("删除成功");
    fetchStudents();
  } catch (error) {
    ElMessage.error("删除失败");
  }
};

// ===== 监听器和初始化 =====
watch(searchQuery, handleSearch);
onMounted(fetchStudents);

// ===== 登出 =====
const logout = () => {
  // 清除 localStorage 中的 token
  localStorage.removeItem("token");
};

```

```

    // 跳转到登录页面
    router.push("/login");
  };
</script>

<style scoped>
.students-container {
  padding: 20px;
}

.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}

.el-input {
  width: 300px;
}

.el-pagination {
  margin-top: 20px;
  text-align: center;
}
</style>

```

App.vue

```

<template>
  <div id="app">
    <el-container style="height: 100vh">
      <!-- 顶部导航 -->
      <el-header
        style="
          background-color: #409eff;
          color: white;
          text-align: center;
          font-size: 20px;
        ">
        >
        学生信息管理系统
      </el-header>

      <!-- 主体部分 -->
      <el-container>
        <!-- 主内容区域 -->
        <el-main style="padding: 20px">
          <router-view />
        </el-main>
      </el-container>
    </el-container>
  </div>
</template>

```

```
<script setup>
</script>

<style scoped>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin: 0;
}

.el-container {
  display: flex;
  flex-direction: column;
  height: 100%; /* 全屏适配 */
}

.el-header {
  height: 60px;
  line-height: 60px; /* 垂直居中 */
  padding: 0;
}

.el-aside {
  padding: 10px 0;
}

.el-main {
  overflow-y: auto; /* 主内容区域可滚动 */
}
</style>
```

## main.js

### 1. 整体布局结构

- 使用 **Element Plus** 的 `el-container` 组件构建全屏布局。
- **顶部导航栏** (`el-header`) :
  - 显示系统标题 "学生信息管理系统"。
  - 固定高度 `60px`，蓝色背景 (`#409eff`)，白色文字，居中显示。
- **主内容区域** (`el-main`) :
  - 通过 `<router-view />` 动态渲染 Vue Router 匹配的页面（如登录页、学生列表页等）。
  - 支持内容滚动 (`overflow-y: auto`)。

### 2. 核心功能

- **路由管理**:
  - 作为单页应用 (SPA) 的入口，通过 `<router-view>` 加载不同的子页面（如 `/login`、`/students`）。
- **响应式设计**:
  - 使用 `height: 100vh` 确保布局占满整个视口高度。
  - 通过 Flex 布局 (`flex-direction: column`) 实现上下结构。

### 3. 样式

- 全局样式:

- 设置默认字体 (Avenir, Helvetica, Arial)、文字抗锯齿优化。
- 移除默认边距 (margin: 0)。

- 局部样式:

- 顶部导航栏文字垂直居中 (line-height: 60px)。
- 主内容区域自动滚动 (避免内容溢出)。

```
<template>
  <div id="app">
    <el-container style="height: 100vh">
      <!-- 顶部导航 -->
      <el-header
        style="
          background-color: #409eff;
          color: white;
          text-align: center;
          font-size: 20px;
        ">
        >
        学生信息管理系统
      </el-header>

      <!-- 主体部分 -->
      <el-container>
        <!-- 主内容区域 -->
        <el-main style="padding: 20px">
          <router-view />
        </el-main>
      </el-container>
    </el-container>
  </div>
</template>

<script setup>
</script>

<style scoped>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin: 0;
}

.el-container {
  display: flex;
  flex-direction: column;
  height: 100%; /* 全屏适配 */
}

.el-header {
```

```
height: 60px;
line-height: 60px; /* 垂直居中 */
padding: 0;
}

.el-aside {
padding: 10px 0;
}

.el-main {
overflow-y: auto; /* 主内容区域可滚动 */
}
</style>
```

## router.js

### 1. 路由配置

- 定义路由规则：
  - `/`: 根路径自动重定向到 `/students`。
  - `/login`: 对应 `LoginView` 组件（登录页）。
  - `/students`: 对应 `StudentView` 组件（学生信息页）。
- 路由模式：
  - 使用 `createWebHistory` 启用 HTML5 历史模式（无 `#` 的 URL）。

### 2. 路由守卫（权限控制）

- 全局前置守卫 (`beforeEach`):
  - 非登录页访问检查:  
如果用户访问非 `/login` 的页面（如 `/students`），会检查 `localStorage` 中是否存在 `token`。
    - 无 `token`: 强制跳转到 `/login`。
    - 有 `token`: 允许继续访问。
  - 登录页访问: 直接放行。

### 3. 核心功能

- 根据 URL 路径动态渲染对应的 Vue 组件（如 `LoginView` 或 `StudentView`）。
- 保护需要登录的页面（如学生页），未登录用户自动跳转至登录页。
- 支持浏览器前进/后退操作（通过 `createWebHistory`）。

```
import { createRouter, createWebHistory } from "vue-router";
import LoginView from "../views/LoginView.vue";
import StudentView from "../views/StudentView.vue";

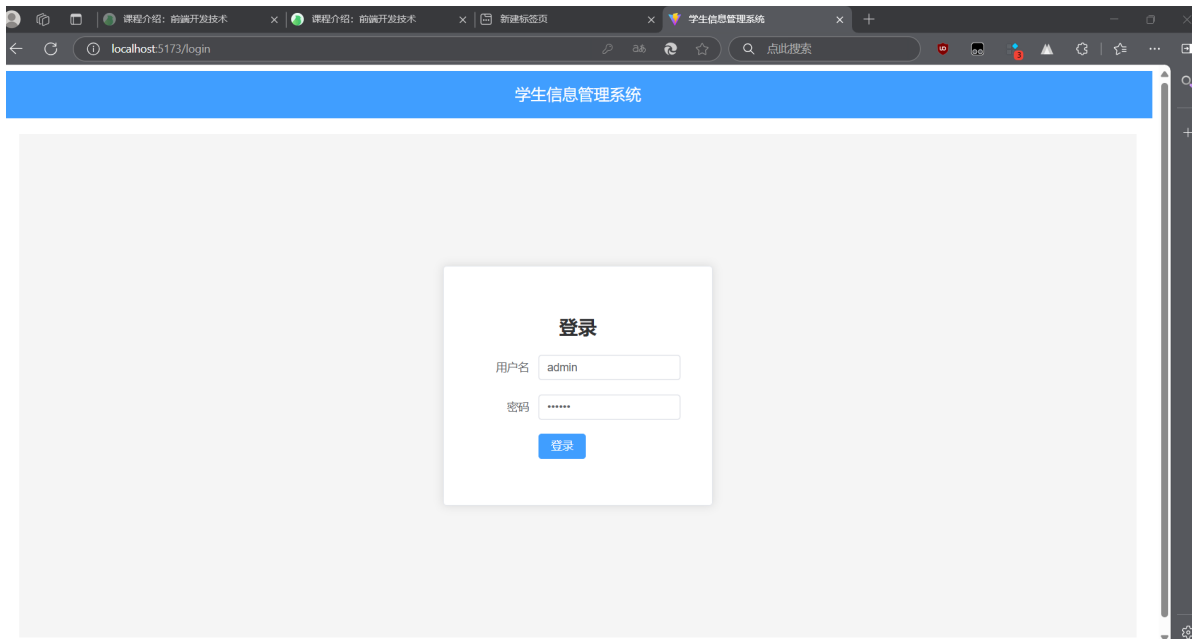
const routes = [
  { path: "/", redirect: "/students" },
  { path: "/login", component: LoginView },
  { path: "/students", component: StudentView }
];
```

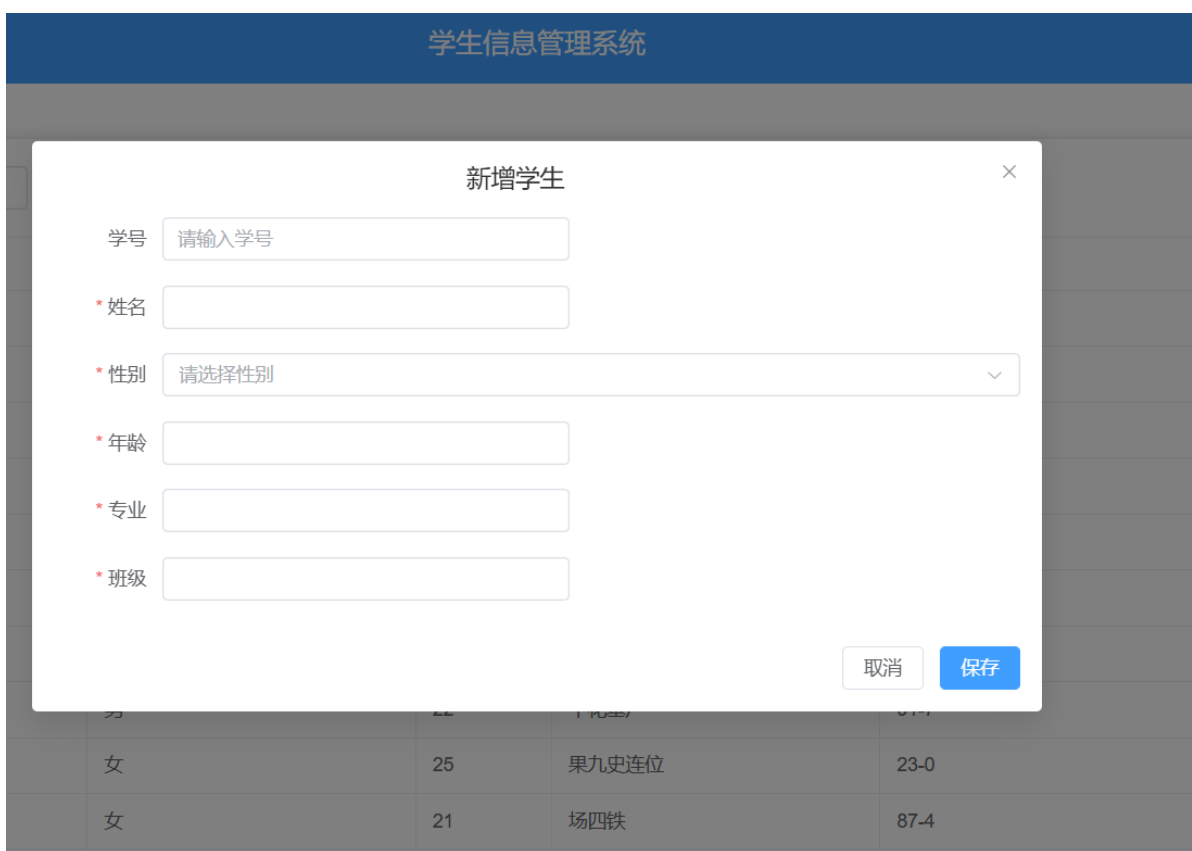
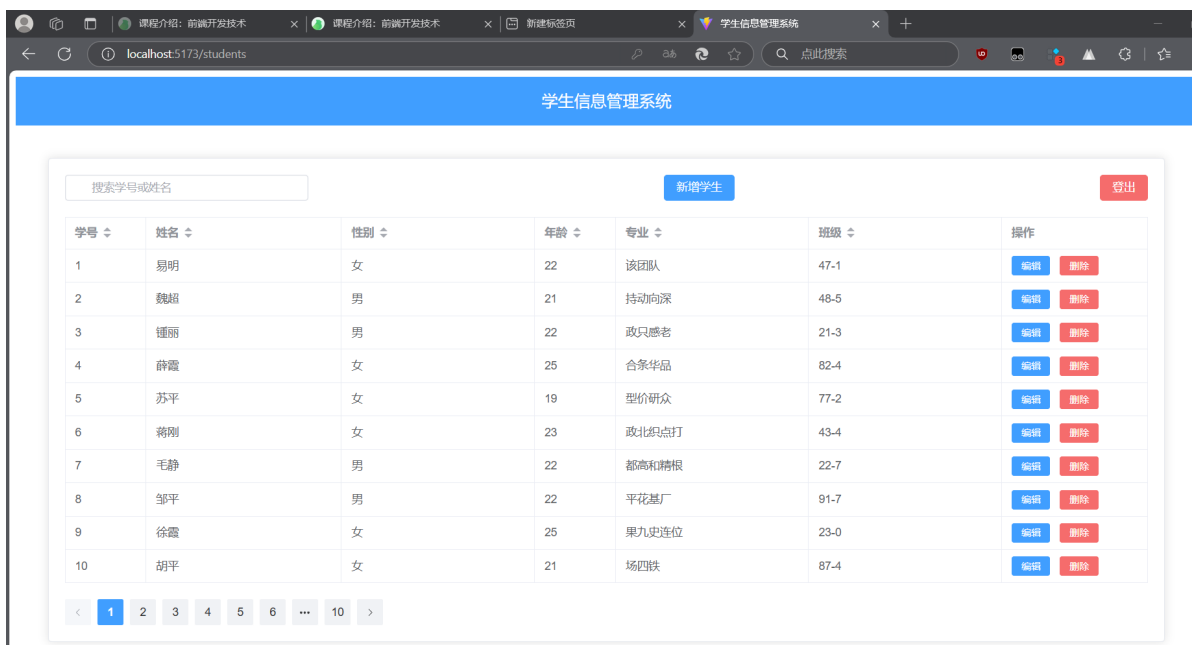
```
const router = createRouter({
  history: createWebHistory(),
  routes
})

router.beforeEach((to, from, next) => {
  if (to.path !== '/login') {
    let token = localStorage.getItem('token');
    if (!token) {
      next('/login');
    } else {
      next();
    }
  } else {
    next();
  }
});

export default router;
```

## 运行结果截图





## 遇到的问题

- 1、`router.beforeEach` 中未排除登录页的token检查，导致登录成功后仍被重定向到 `/login`

[Vue Router warn]: Redirected when going from `"/login"` via navigation guard to `"/login"`

### • 修复方案：

```
if (to.path === '/login') return next() // 明确排除登录页
```

- 2、新增学生时未按Mock规则生成数据（如缺少`@increment`等占位符）

Error: Invalid JSON structure from Mock.js

- 修复方案:

```
// 在POST接口中手动补全Mock字段
Mock.mock("/api/students", "post", (req) => {
  const newStudent = {
    ...JSON.parse(req.body),
    id: '@increment', // 确保符合Mock规则
    class: '@string("number", 2)-@string("number", 1)'
  }
})
```

### 3、弹窗表单的 dialogForm 未初始化数字类型字段 (如 age: null 会导致验证失败)

[ElementPlus warn]: Invalid prop: type check failed for prop "model". Expected Object, got Undefined

- 修复方案:

```
const dialogForm = reactive({
  age: 18, // 改为数字默认值而非null
  // 其他字段...
})
```

### 4、删除学生时未处理API请求失败的情况 (如重复删除同一ID)

Uncaught (in promise) Error: Request failed with status code 404

- 修复方案:

```
const deleteStudent = async (id) => {
  try {
    await axios.delete(`/api/students/${id}`)
  } catch (err) {
    ElMessage.error(err.response?.data?.message || '删除失败')
  }
}
```

### 建议解决方案:

1. 添加全局错误拦截器 (Axios + Vue Router)
2. 对Mock数据生成进行单元测试
3. 使用TypeScript强化类型检查

这些改进可显著提升代码健壮性, 减少运行时错误。