企业外出审批与差旅管理系统

项目架构：

- 登录页
- 主体框架
    - 首页
    - 外出申请管理
        - 我的申请（员工端）
        - 待审批申请（审批端）
    - 差旅申请管理
    - **员工信息管理**
    - **组织架构管理**
    - **权限角色管理**
- 公共功能
    - 登录鉴权（JWT）
    - 菜单权限控制（前端动态路由）
    - 接口封装 & 模拟数据（Mock.js）

具体实现：

- 登录页

  登录页支持账号密码登录，登录后会从后台返回一个模拟的 JWT token，我们将 token 保存在本地 localStorage 中。之后每次请求需要权限的数据时，我们会通过请求拦截器自动把 token 携带在请求头中。同时我们还做了登录失败提示、表单校验等交互优化

- 主体框架

  登录成功后跳转到系统首页，整体采用的是 Element-UI 的基础布局组件。左侧是动态渲染的菜单栏，顶部是用户信息栏，右侧是主体内容。菜单栏会根据用户的角色权限动态显示不同的页面入口，比如员工用户只能看到自己的申请，而审批管理员可以看到审批模块

    - 首页
    - 外出申请管理
        - 我的申请（员工端）

          “员工可以进入『我的申请』页面提交外出申请。申请表单包括外出时间、外出原因、外出类型等字段，提交后会保存数据库。同时申请记录会以列表形式展示，状态分为‘待审批’、‘已通过’、‘已拒绝’，员工可以修改、撤销尚未审批的申请

        - 待审批申请（审批端）

          审批人可以看到所有下属提交的申请。每条记录可以点击进入查看详情，并进行‘通过’、‘驳回’等操作，状态会在前端实时更新

    - 差旅申请管理

这个模块跟外出申请类似，但表单内容更复杂一些，比如还要填写交通工具、住宿地点、预算费用等。提交后进入待审批流程。这个模块也可以继续扩展出差报销等功能

- **员工信息管理**

  员工信息管理模块我采用了典型的左树右表布局。左侧是组织架构树，使用的是 Element-UI 提供的 `el-tree` 组件，右侧是员工列表，用 `el-table` 展示当前选中部门下的所有员工。通过监听树节点的点击事件，我会根据筛选条件更新右侧表格，实现联动效果

  > `el-table` 是根据传入的 :data="" 中的数据渲染表格的，我们可以根据点击树节点获取到的 id 利用数组的 filter 方法从所有数据中筛选出需要的数据放入 data 中就完成了效果

- **组织架构管理**

  组织结构是用 Element-UI 的 `el-tree` 树形控件实现的，我们定义了一个嵌套结构的数据格式，用来表示企业的各个部门。支持新增部门、编辑名称、删除操作等。这个模块和员工信息模块联动，可以通过组织结构筛选员工

  > `el-tree` 根据 :data="" 里面的数据渲染结构，不过里面数据需要通过 children属性嵌套才能实现树形结构，普通数据可以通过递归重新组织结构。用户可以通过操作按钮进行新增、编辑部门名称、删除部门等基本管理操作（都是调用相应操作的接口，然后再重新获取组织结构数据并更新到data，页面重新渲染），弹窗使用 `el-dialog` 搭配 `el-form` 实现。

- **权限角色管理**

  系统内的用户分为几类角色，比如普通员工、审批人、系统管理员。每种角色可以访问的页面不同。在用户登录后根据角色动态设置菜单项，并通过 Vue Router 路由守卫来限制用户进入不该访问的页面，实现了简单的前端权限控制

  > 我们在前端会提前定义好哪些是所有人都可以访问的公共路由，哪些是需要角色权限的动态路由。登录成功后，后端会返回用户的角色，我们根据这个角色在前端筛选出一份路由表并用 Vue Router 的 `addRoute` 方法动态注册路由（在登录逻辑中动态注册）。同时，利用 `meta.roles` 字段（配置路由时添加）进行页面权限控制，在路由守卫中判断用户是否有权限访问对应页面。`el-menu` 利用 v-for 基于筛选出的路由表渲染 `el-menu-item` 实现动态设置侧边菜单栏
  >
  > 为什么要"设置路由守卫"?
  >
  > 即使我们动态添加了路由，但用户可以直接在地址栏输入任何 URL，如果不加拦截，他还是可以强行访问管理员页面（即使可能没有注册管理员页面的路由，但添加守卫更保险）

- 公共功能
  - 登录鉴权（JWT）
  - 菜单权限控制（前端动态路由）
  - 接口封装 & 模拟数据（Mock.js）

    在这个项目中我使用了 Mock.js 来模拟接口，解决了后端接口尚未完成时前端无法调试的问题。我在 `mock/index.js` 中定义了登录、用户列表等模拟接口，使用 `Mock.mock()` 拦截请求并返回

假数据。在项目初始化时（如 `main.js` 中）判断是开发环境才引入 `mock/index.js`，确保线上不启用 Mock。这样我可以独立开发页面逻辑，同时保证模拟数据结构贴近真实后端返回，提升了开发效率和团队并行开发能力

Mock.js 的基本使用流程（项目实战）

- **步骤 1：安装**

```
npm install mockjs
```

- **步骤 2：创建 `mock` 文件夹，并配置接口**

```
import Mock from 'mockjs'

// 模拟用户登录接口
Mock.mock('/api/login', 'post', (options) => {
  return {
    code: 200,
    message: '登录成功',
    data: {
      token: 'fake-jwt-token',
      username: 'admin'
    }
  }
})

// 模拟获取用户列表
Mock.mock('/api/users', 'get', {
  code: 200,
  message: '获取成功',
  data: Mock.mock({
    'list|5-10': [{
      'id|+1': 1,
      name: '@cname',
      age: '@integer(20, 50)',
      email: '@email'
    }]
  })
})
```

- **步骤 3：在 `main.js` 中引入（确保在你发请求之前）**

```
import './mock/index.js'
```

- **步骤 4：在前端发请求就会被 Mock 拦截**

```
axios.get('/api/users').then(res => {
  console.log(res.data)
})
```

这时你访问的 `/api/users` 不会去请求真实后端，而是 Mock.js 返回你配置好的假数据。

Mock.js 常见面试问法（基础）

1.Mock.js 是什么?

Mock.js 是一个用来拦截 `Ajax` 请求，并返回模拟数据的前端开发库。它可以拦截你的接口请求，并根据配置返回伪造的 JSON 数据

2.Mock.js 是干什么的?

用于在开发阶段模拟接口数据，在后端接口未完成时也能进行前端开发，提升开发效率

3.它是如何工作的?

Mock.js 会拦截 XMLHttpRequest（或 Fetch）请求，根据你配置的规则生成随机数据，并返回给你模拟的结果

4.Mock.js 能在生产环境使用吗?

不能！Mock.js 只适合开发环境。它拦截的是前端请求，不具备真实后端逻辑和安全机制，生产环境应使用真实接口

5.项目中怎么区分是否开启 Mock 模式?

我们可以使用 `process.env.NODE_ENV` 来判断环境，在开发环境中引入 `mock/index.js`，在生产环境就不引入

完整的git开发流程（两人协作）

git框架：

- master 已上线/准备上线的稳定版本（仅登录页 + 首页）
- dev 双人开发主力分支（平时都在这个分支干活）
- bugfix 修复线上紧急 bug（从 master 拉出，只用于修 bug）

流程：

1. **首次从远程 gitee 仓库克隆项目并切换到 `dev` 分支**

   远程仓库已有 `dev` 分支，但你克隆完默认是在 `master` 分支上！

   ```
   git clone https://gitee.com/your-project.git
   cd your-project
   git checkout dev
   git pull origin dev
   ```

2. **两人都在同一个远程的 dev 分支上开发不同模块**

   要在 `src/views/` 中按模块划分不同目录，互不干扰

   我开发：

   - `src/views/user-manage/` 员工信息管理
   - `src/views/org-manage/` 组织架构管理
   - `src/views/role-manage/` 权限角色管理

3. **每写完一个模块，先 pull 再 push，避免覆盖对方代码**

```
# 提交你写的模块
git add .
git commit -m "feat: 新增权限角色管理模块"

# 关键步骤：先拉别人的更新（避免冲突）
git pull origin dev

# 合并完没有冲突，再推送到远程
git push origin dev
```

> 如果你不先 `pull`，而直接 `push`，就很可能出现「拒绝推送」或「覆盖别人的代码」的冲突

4. **项目开发完后，将 dev 合并到 master（其中一人完成）**

```
git checkout master
git pull origin master
git merge dev
git push origin master
```

紧急修改bug

线上稳定功能出现 bug，必须从 `master` 分出 `bugfix` 修复。如果**尚未上线**，还在开发中的模块出 bug 在 `dev` 中修复即可，建议从 `dev` 拉 `fix/xxx` 分支去修

流程：

1. 从 `master` 拉出 bugfix 分支：

```
git checkout master
git pull origin master    # 确保本地是最新的
git checkout -b bugfix/login-error-fix
```

> 命名建议：`bugfix/模块名-问题描述`（如 `bugfix/login-error-fix`）

2. 修复 bug，提交修改：

```
# 修改代码
git add .
git commit -m "修复登录错误跳转的问题"
```

3. 推送到远程仓库：

```
git push origin bugfix/login-error-fix
```

4. 发起合并请求（PR）到 `master` 分支（模拟上线）

```
# Gitee 或 GitHub 网页上点「新建 Pull Request」，选择合并到 master
```

5. 代码审核通过后，合并 & 删除 bugfix 分支：

```
# 合并成功后，本地和远程都可以删除该分支
git branch -d bugfix/login-error-fix
git push origin --delete bugfix/login-error-fix
```

关于 Pull Request

Pull Request 是一种代码协作机制，用来：向别人发出请求：请你审核我的代码，然后决定是否合并到主干分支里。它的本质就是：**发起一个"合并代码"的请求**

假设你从 `dev` 分支新建了 `feature/employee-manage` 分支，完成了一个功能模块开发。你想把它合并到 `dev`，但不直接合并，而是： **通过 Pull Request 提交你的代码变更，供队友审核，然后再决定是否合并。**

流程：

1. 你本地新建并开发了一个分支，比如 `feature/employee-manage`，提交代码并推送到了远程仓库：

```
git push origin feature/employee-manage
```

2. 登录 Gitee，打开你的仓库，在仓库首页点击右上角的 **「新建 Pull Request」**

3. 选择两个分支：

   - **源分支（source）**：你开发完成的分支（如 `feature/employee-manage`）
   - **目标分支（target）**：你想合并到的分支（如 `dev` ）

4. 填写说明：比如你做了什么功能开发，做了什么改动？

5. 提交 PR，等待另一个人审核并点击「合并」按钮
6. 合并成功之后本地切换到目标分支，并拉取最新代码，然后可以删除这个开发分支（本地 + 远程）

```
# 本地切换到目标分支
git checkout dev
git pull origin dev

# 合并成功后，本地和远程都可以删除该分支

# 本地
git branch -d feature/employee-manage
# 远程
git push origin --delete feature/employee-manage
```

数字权益发放与管理平台

项目架构：

原电商项目：

- 登录页
- 首页架子
  - 首页
  - 分类页
  - 购物车
  - 我的
- 搜索页
- 搜索列表页
- 商品详情页
- 结算支付页
- 我的订单页

数字权益项目：

- 登录页
- 首页架子
  - 首页
  - 权益分类页
  - 权益领取页
  - 我的
- 搜索页
- 搜索列表页
- 权益详情页

- 权益状态管理
- 权益数据分析
- 权益管理页
- 权益发放记录

具体实现：

- 登录页
- 首页架子
    - 首页

      顶部轮播图 Banner（展示活动/福利），推荐权益卡片列表（如：生日券、新用户礼包等）
    - 权益分类页

      展示权益分类列表（如：节日券、活动券），点击分类可跳转到对应分类权益列表页

      > 分类页通过查询参数将 catergoryld 传到列表页
    - 权益领取页

      展示可领取的权益（满足条件时才显示领取按钮），点击按钮发起权益领取（调用发放接口），全选反选

      > 判断用户是否已领取过（状态展示控制），权益领取按钮状态：已领取 / 可领取 / 不可领取，弹窗确认领取 + 权益说明页跳转。也可全选统一领取或删除 isChecked 配合数组方法实现全选反选
    - 我的

      展示用户头像、昵称，查看我的领取记录（历史券），权益状态管理

      > 调用获取权益状态接口，按状态分类展示（`van-tab`），支持“退出登录”逻辑处理
- 搜索页

  输入关键词搜索权益，提供历史记录、热门搜索推荐

  > 搜索页中的搜索信息通过查询参数传到列表页，历史记录数组存在数组中，通过数组方法调整数组中的数据 vue 动态渲染
- 搜索列表页

  展示搜索结果列表（卡片形式），支持条件筛选（分类/有效期/是否已领取），可跳转至权益详情

  > 卡片组件复用，`/prodetail/${item.goods_id}` 动态路由传参跳转到权益详情页
- 权益详情页

展示权益券详情：图片、描述、有效期、使用规则等，若满足条件，可直接领取，显示使用说明/适用范围

> 接收动态路由参数 goods_id ，获取权益详情数据，使用 `v-if` 判断状态渲染按钮，调用领取接口，更新状态并反馈弹窗（dialog）

- 权益状态管理

  将领取到的权益按状态分组显示：未使用、已使用、已过期、待审批，用户可一键使用或查看使用说明

  > Tab 切换，利用 `van-tab` 组件配合 `name` 属性，`name` 跟当前显示页相关联，将 `name` 作为参数发请求拿数据

- 权益数据分析

  图表展示权益发放总数、使用率、各分类发放分布，可切换时间范围（如：近7天、近30天），展示活跃用户数、热门权益排行等数据

  > 使用 ECharts 实现图表，动态更新图表数据，表格 + 图表联动（点击图表可过滤表格数据）

- 权益管理页

  创建和编辑权益内容（如券名、描述、图片、使用规则），设置权益类型、有效期、适用范围，支持上下架管理

  > 使用 `el-form` 表单实现新增/编辑，表格展示所有权益列表，支持分页、搜索，使用 dialog 弹窗编辑权益

- 权益发放记录

  查看所有用户的权益发放记录，包含发放时间、领取状态、使用时间等字段

  > 表格分页展示 + 筛选条件（员工名、部门、状态）,使用 `el-table` + `el-pagination` 实现