# MAR-Team 企业固定资产管理应用 设计文档

MAR-Team 企业固定资产管理应用 设计文档

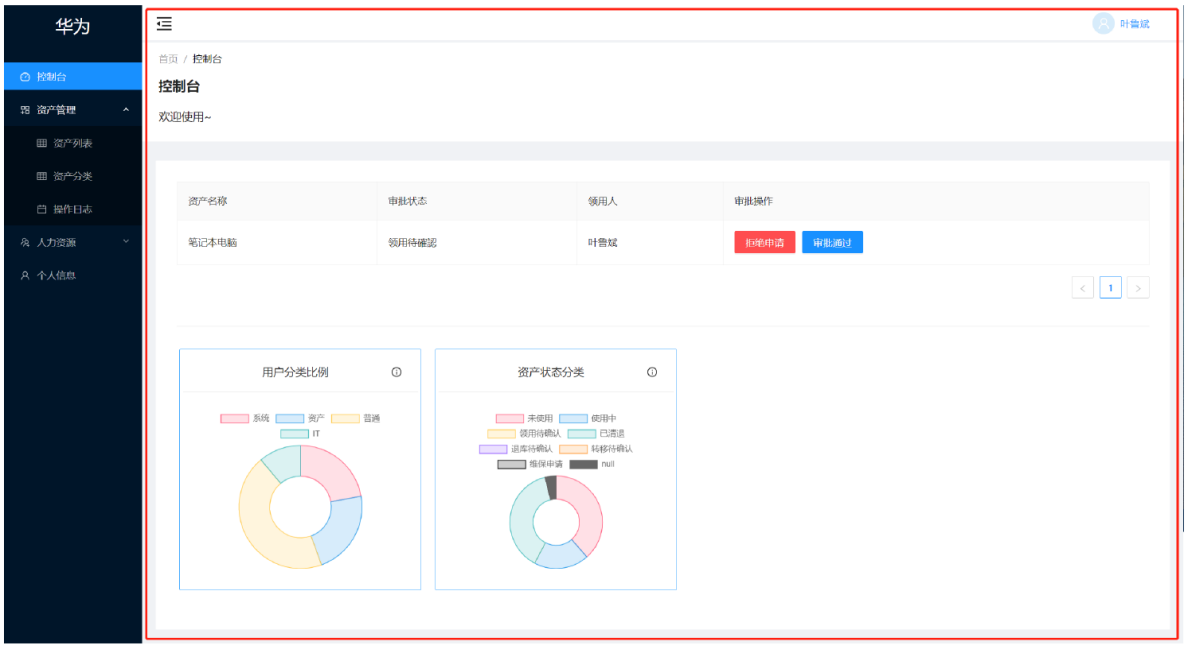
开发人员：梁迪轩 叶鲁斌 戴咏雅 李浩源 李天勤

# Part A 需求分析

## 1 统一门户

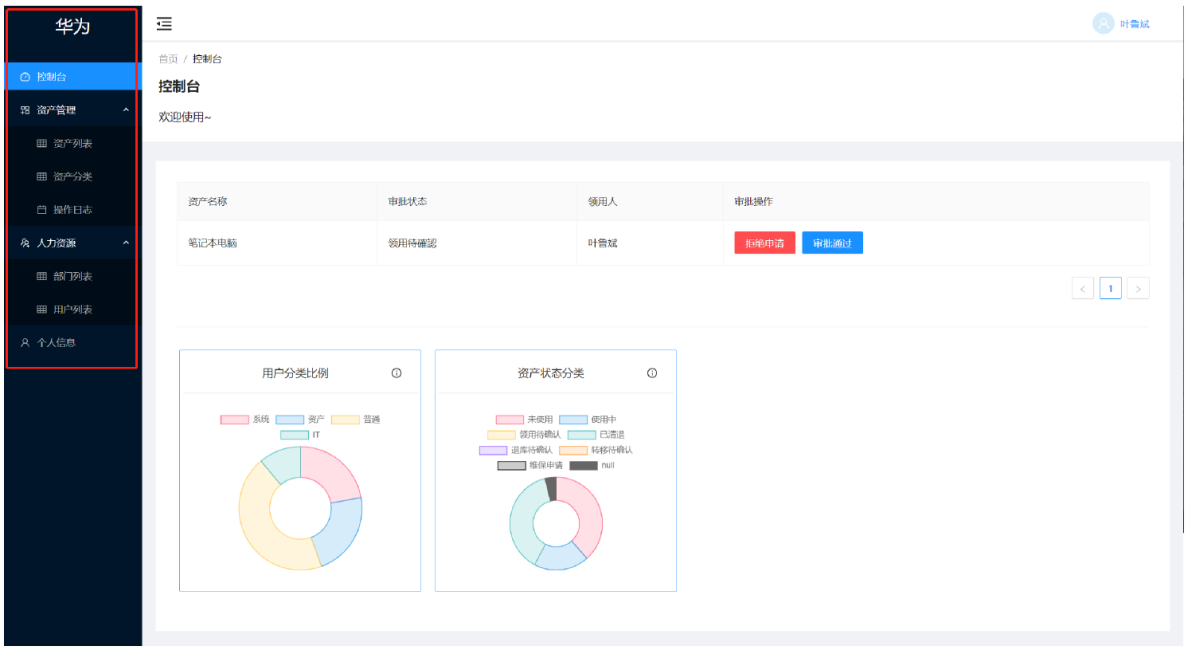
### 1.1 应用门户

员工登录系统后，显示不同应用的门户入口。




### 1.2 菜单管理

系统管理员可以管理应用的菜单项和层级。应支持将第三方URL配置为一个菜单项。不同菜单项对应不同的功能权限。

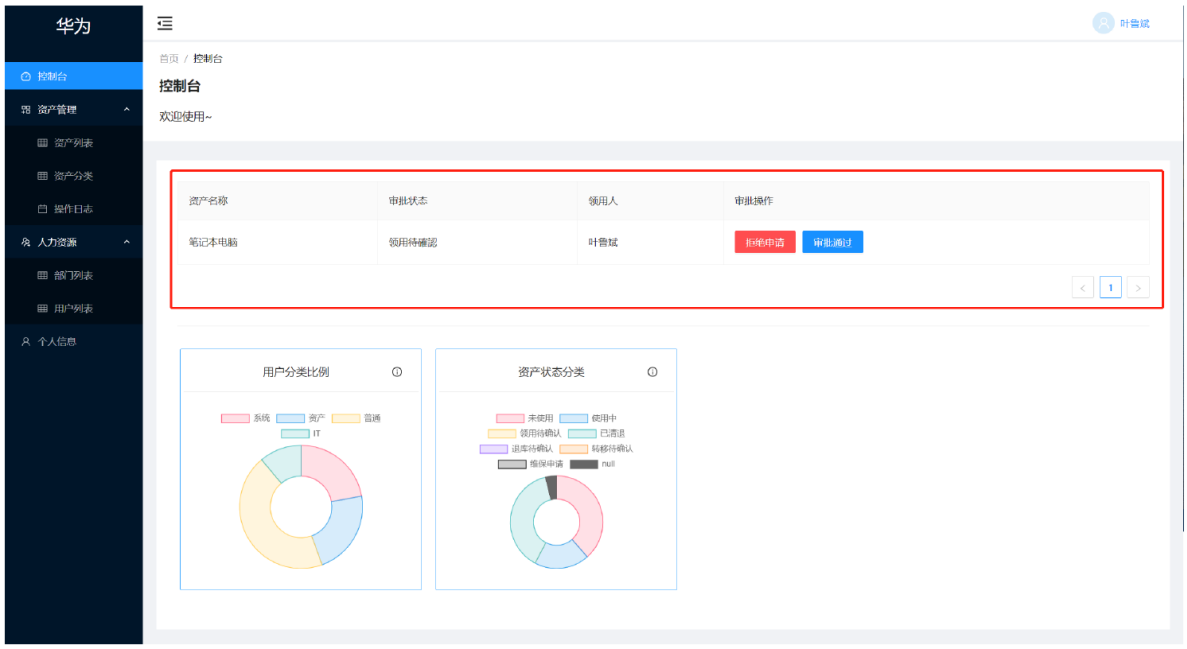



不同的菜单项，进行不同URL的跳转，没有相应权限的账户会再点击后报错。

# 2 个人工作台

## 2.1 待办任务

员工需要处理的代办任务的入口，主要是审批单。例如主管所收到的资产领用的审批单。




# 3 系统管理

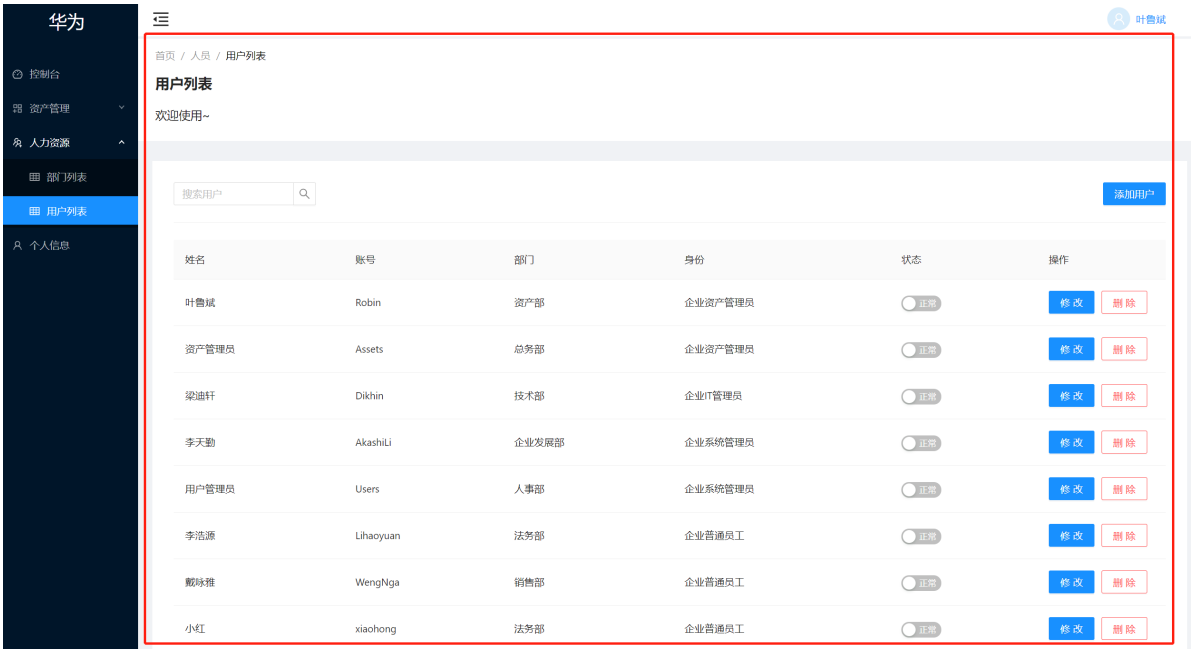
## 3.1 登录登出

用户可以登录系统，可以登出系统。


利用cookie实时操作认证用户身份、在线情况、权限等级等。

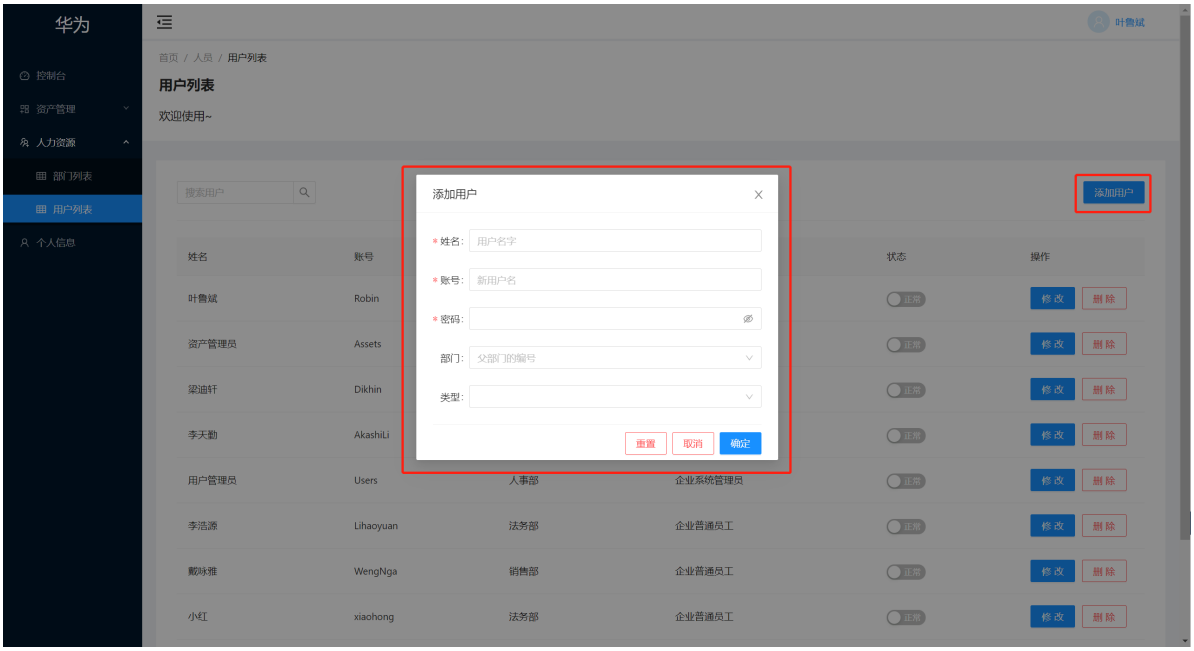
## 3.2 用户管理

系统管理员可以创建维护系统的用户，重置用户密码，锁定解锁用户，设置用户角色。




## 3.3 角色权限

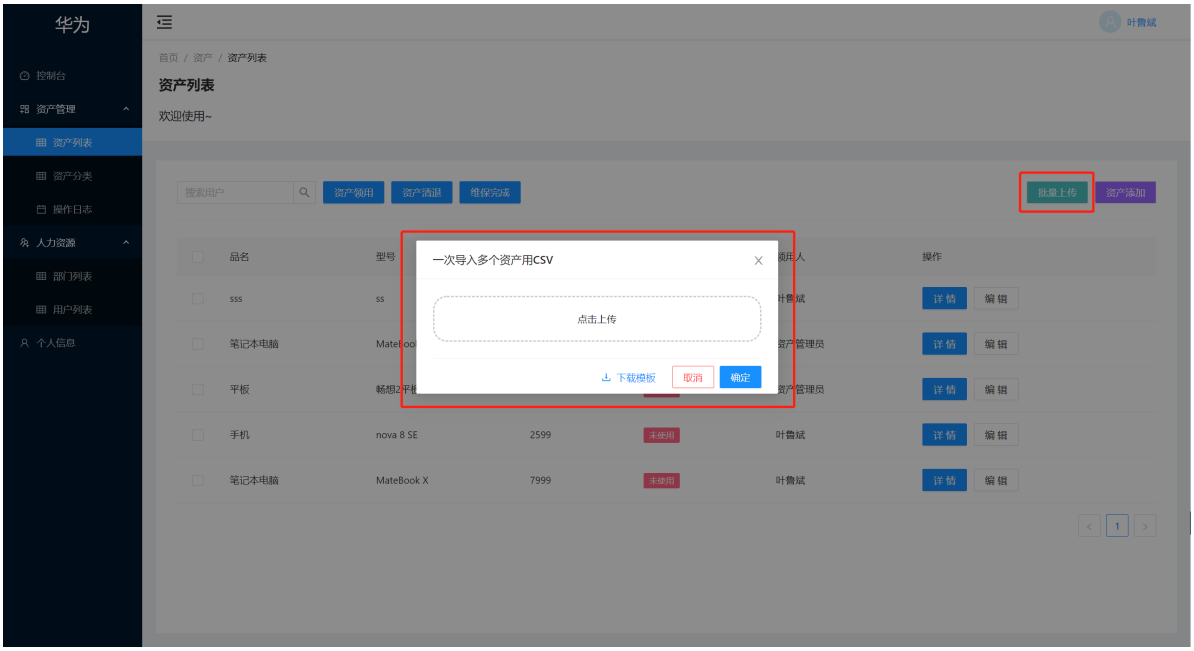
系统管理员可以创建维护系统的角色。




创建的用户可以选择类型，匹配需要的权限。

## 3.4 导入导出管理

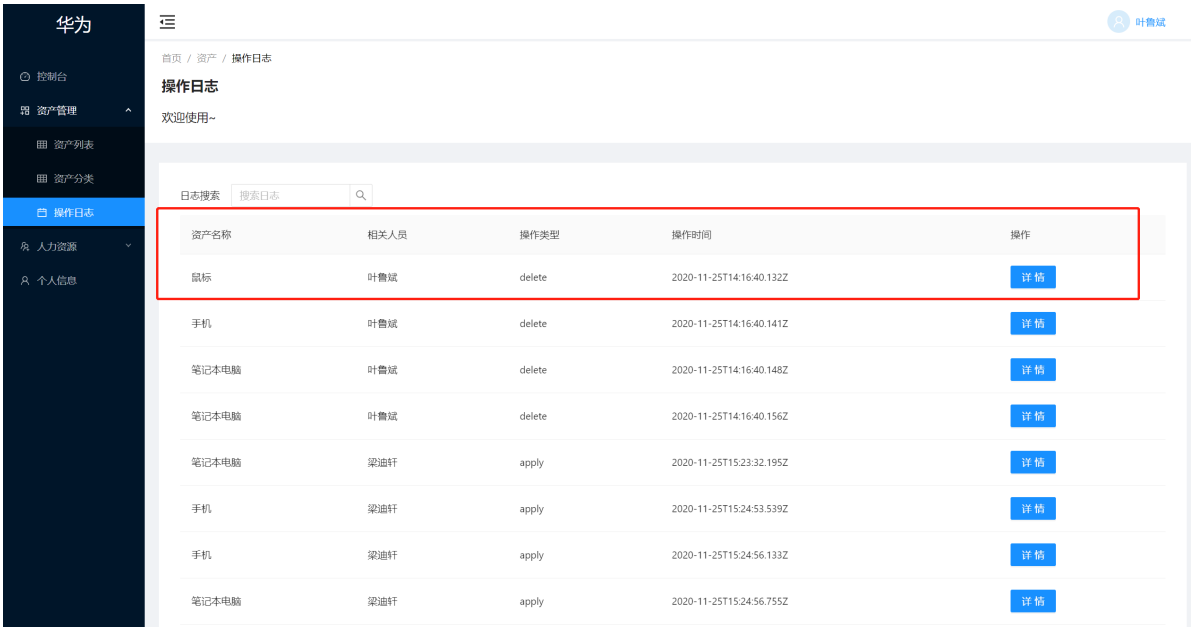
批量导入或导出时需要创建异步任务。系统管理员可以查看和管理异步任务，包括下载失败记录文件，重新执行任务，下载任务结果文件等。




前端实现了异步上传等功能外，还提供了上传文件的模板下载服务来进行最合适的填写指导。

## 3.5 操作日志

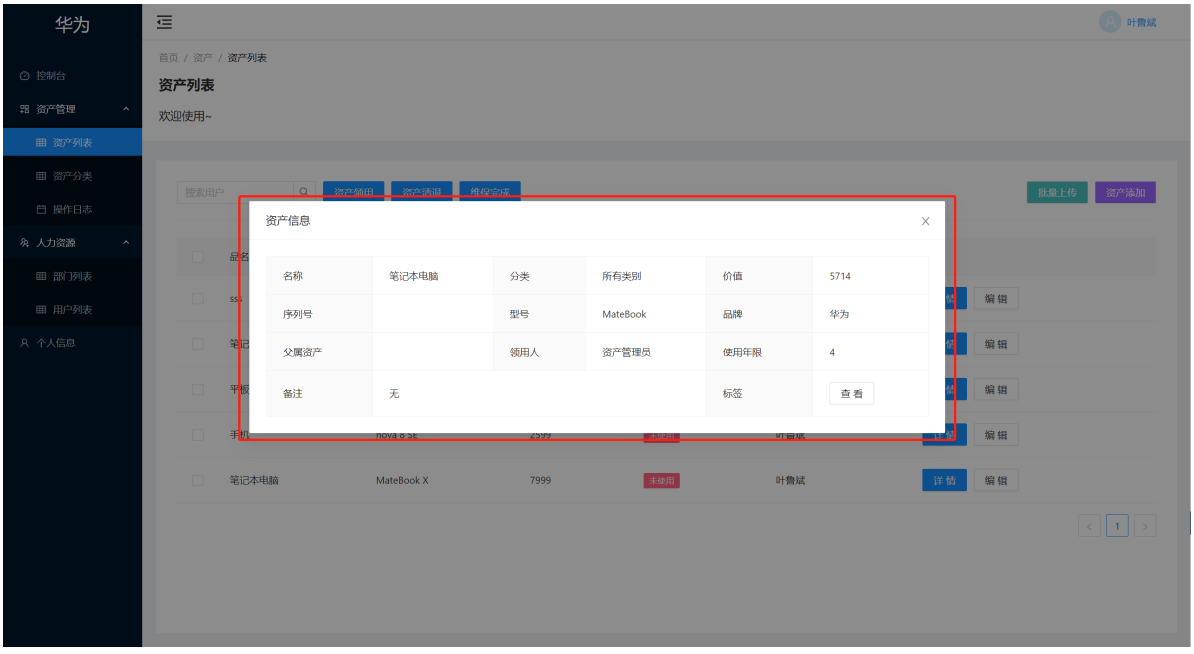
系统管理员可以查看系统操作日志，包括登录日志、关键数据修改日志。




# 4 基础数据

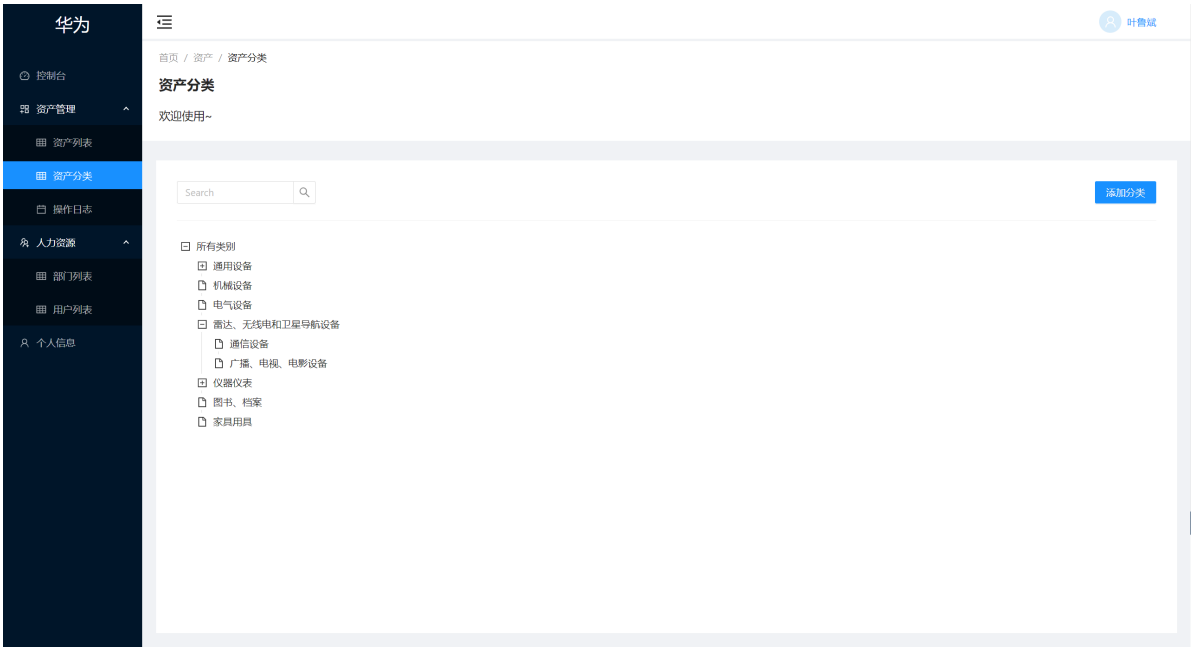
## 4.1 资产属性定义

IT 管理员定义某业务实体下资产的自定义属性，用于业务定制场景。



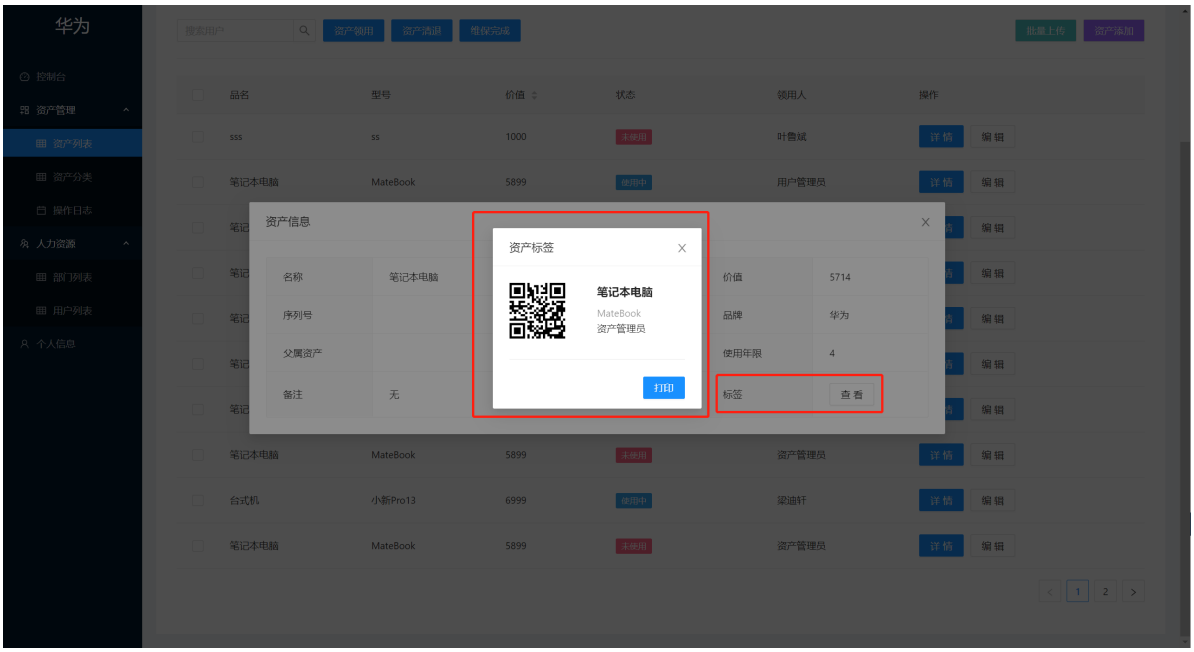

## 4.2 资产分类定义

IT 管理员定义资产的层级分类树。具体的资产将必须挂靠在某个层级分类下。该层级分类为资产的自然分类，即品类。




## 4.3 资产标签定义

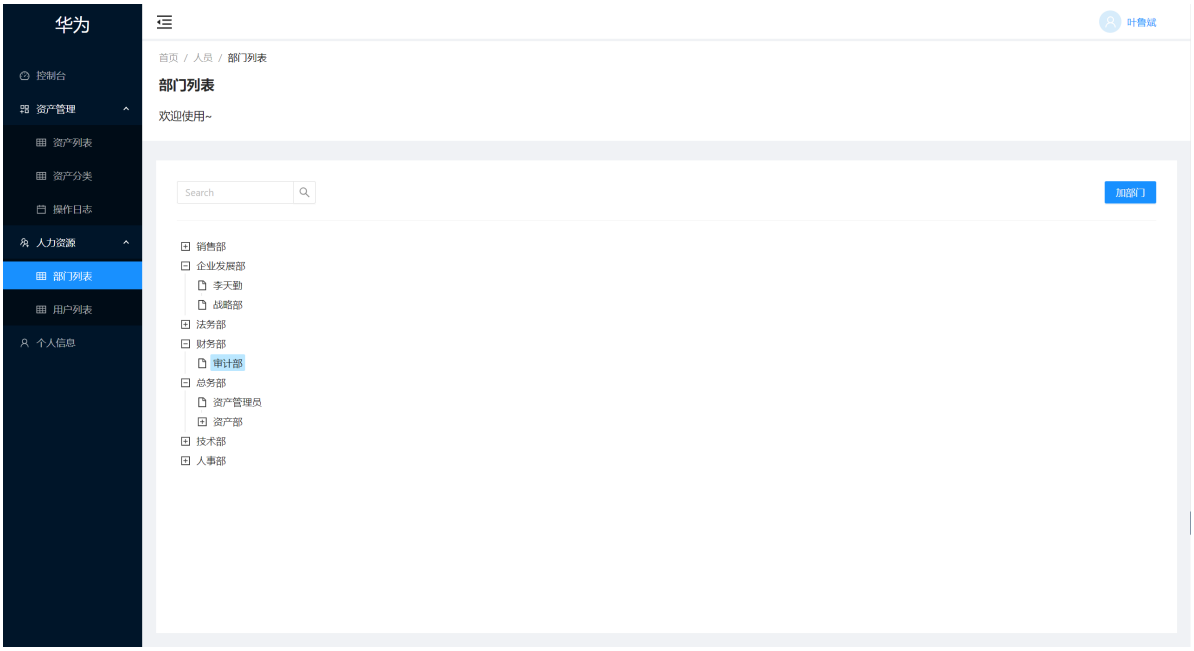
IT 管理员定义打印出来的资产标签卡片内容的模板。打印出来的资产标签用于粘贴在资产设备上。




# 5 组织管理

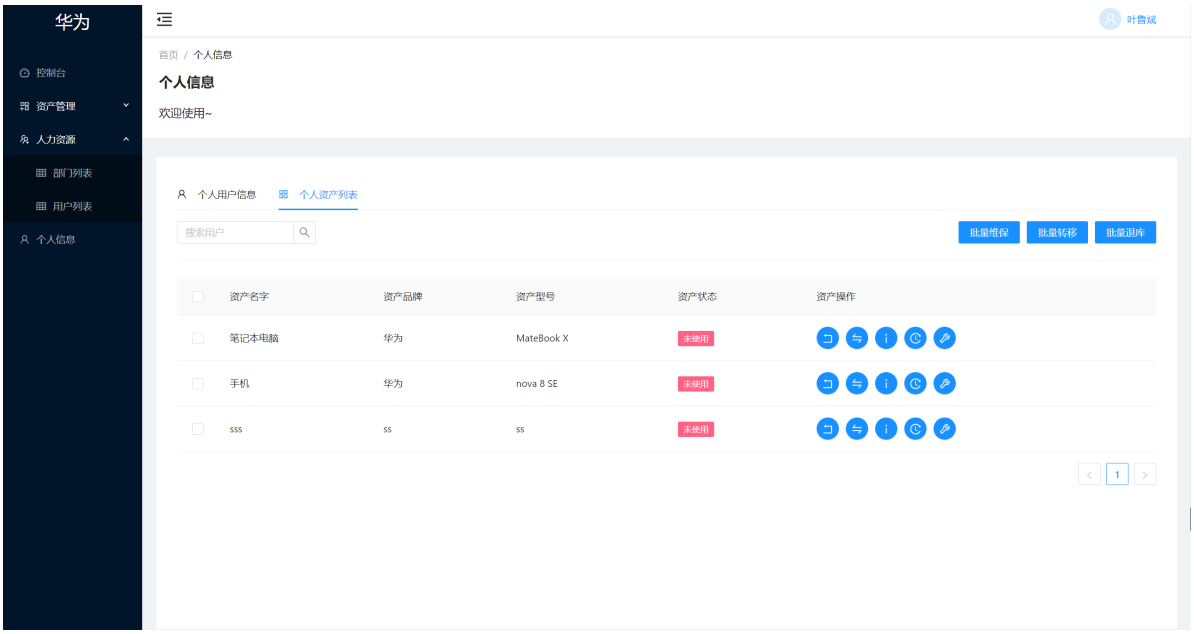
## 5.1 组织数据管理

资产管理员查看和维护企业的组织部门数据。组织可以是多级。组织用于员工的归属和资产的挂账部门。




## 5.2 员工数据管理

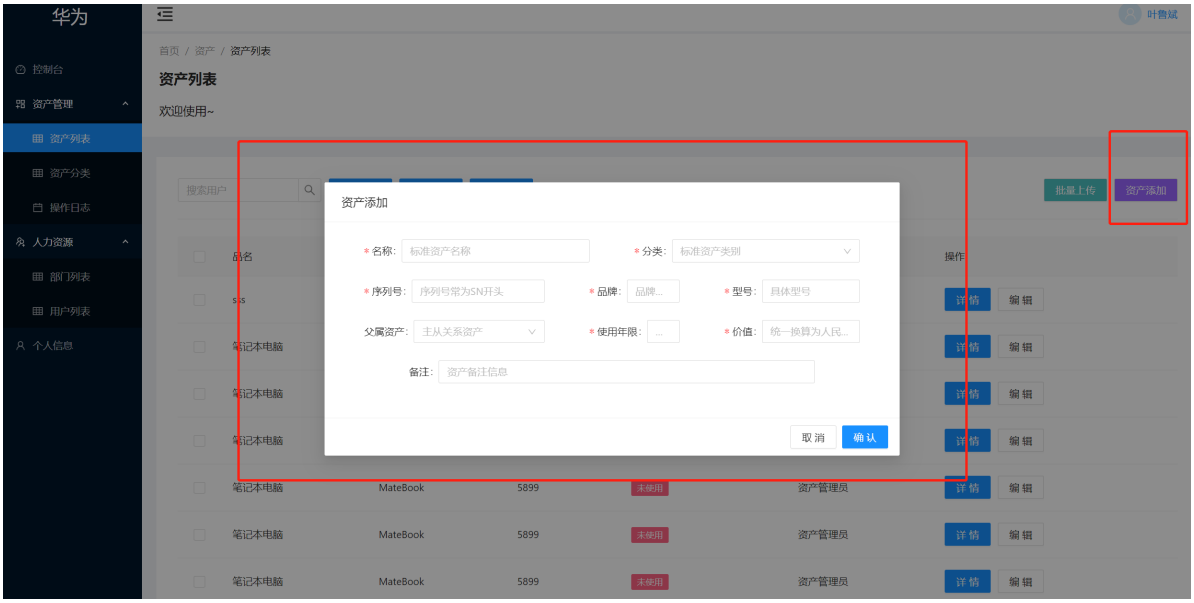
资产管理员查看和维护企业的员工数据。员工和登录用户存在关联。员工用于资产的挂账人。




# 6 资产台账

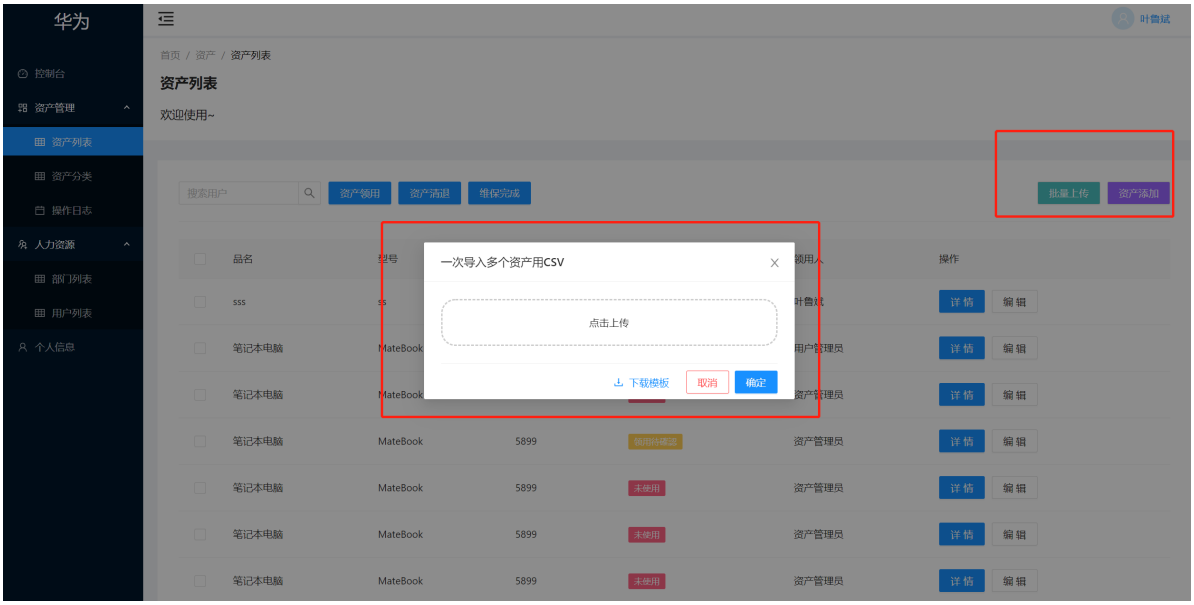
## 6.1 资产录入

资产管理员可以一次录入多个资产，包括条目型资产和数量型资产。多个资产之间存在主从关系，例如计算机主机 和显示器、耳机。 新录入的资产先挂账在某个资产管理员下。



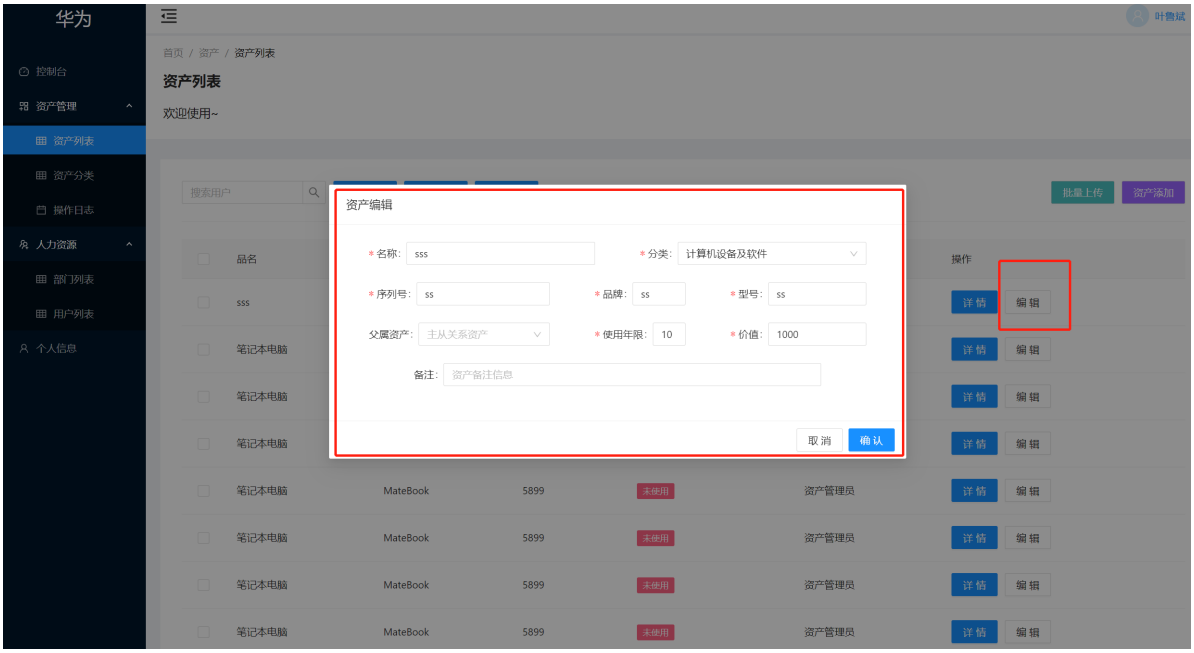

## 6.2 资产批量导入

资产管理员可以通过文件批量导入资产。



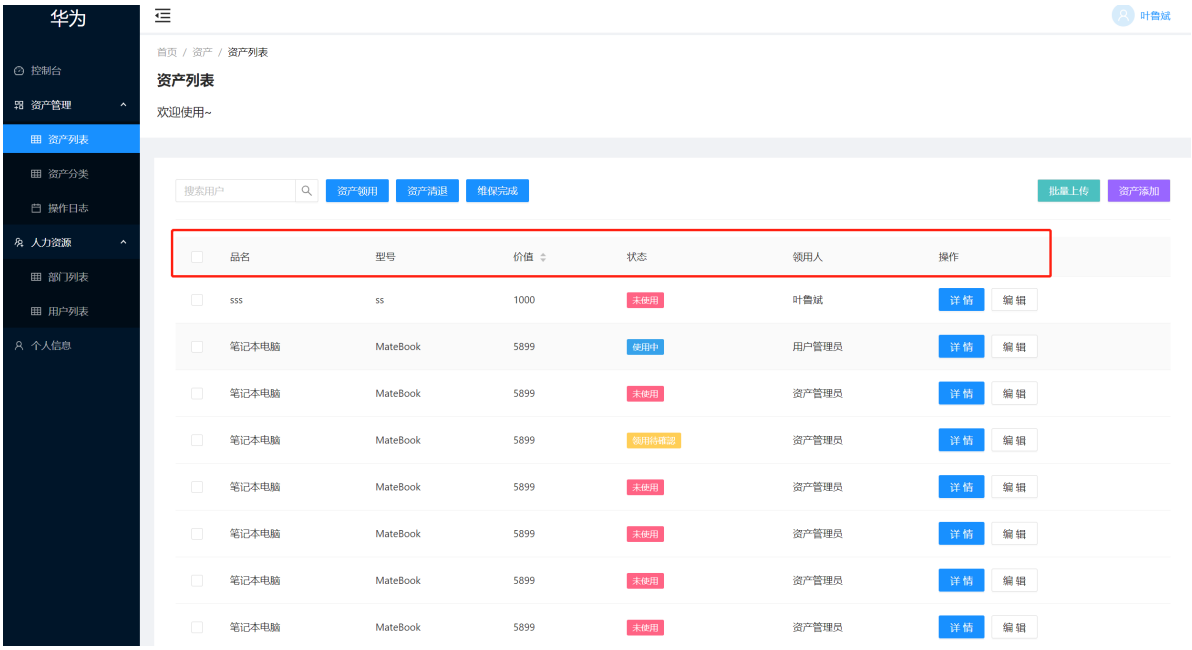

## 6.3 资产信息变更

资产管理员可以修改单个资产实例的信息，包括基本资料、位置、价值、数量等。



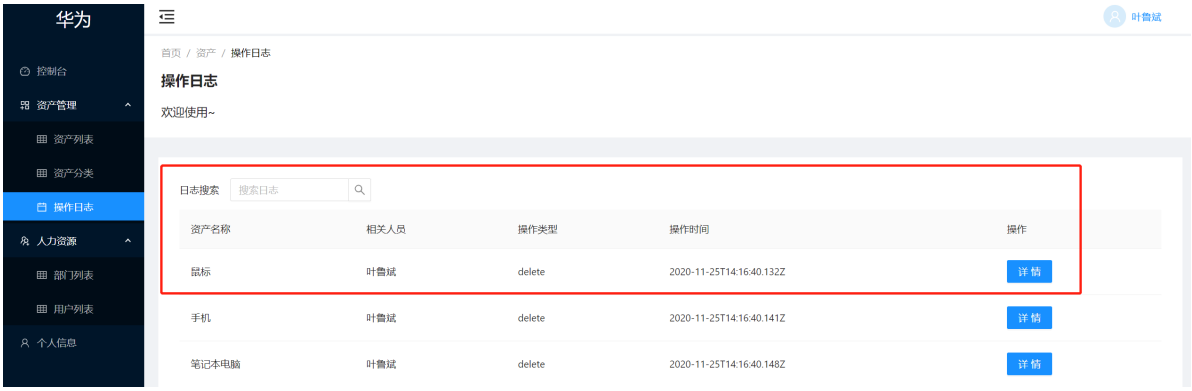

## 6.4 资产查询

资产管理员根据常用条件如名称、描述、分类等查询自己管辖的资产。




## 6.5 资产历史

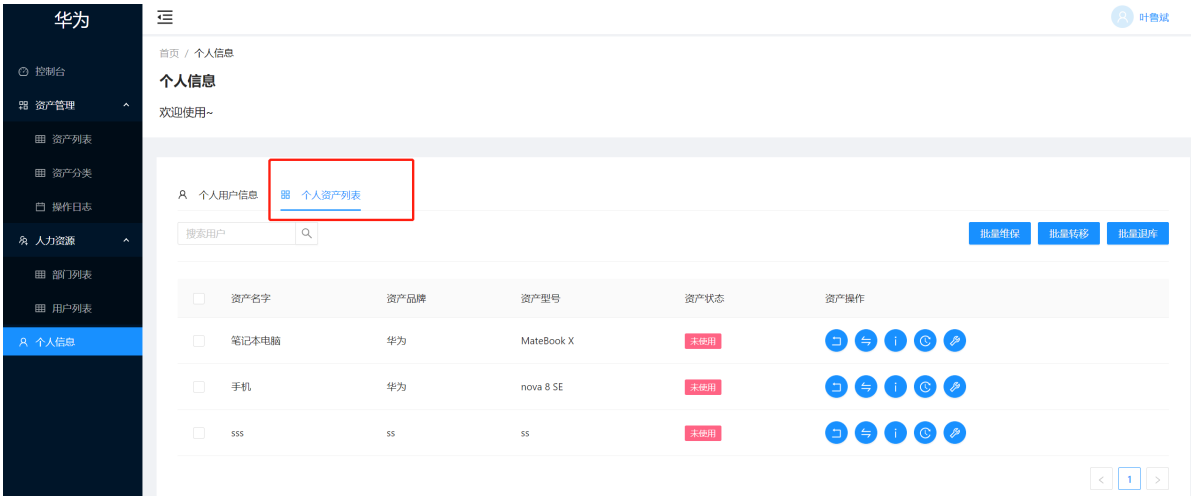
资产管理员或专员根据常用条件查询资产的变更历史，包 括维保、转移、借用等。




# 7 我的资产

## 7.1 我的资产

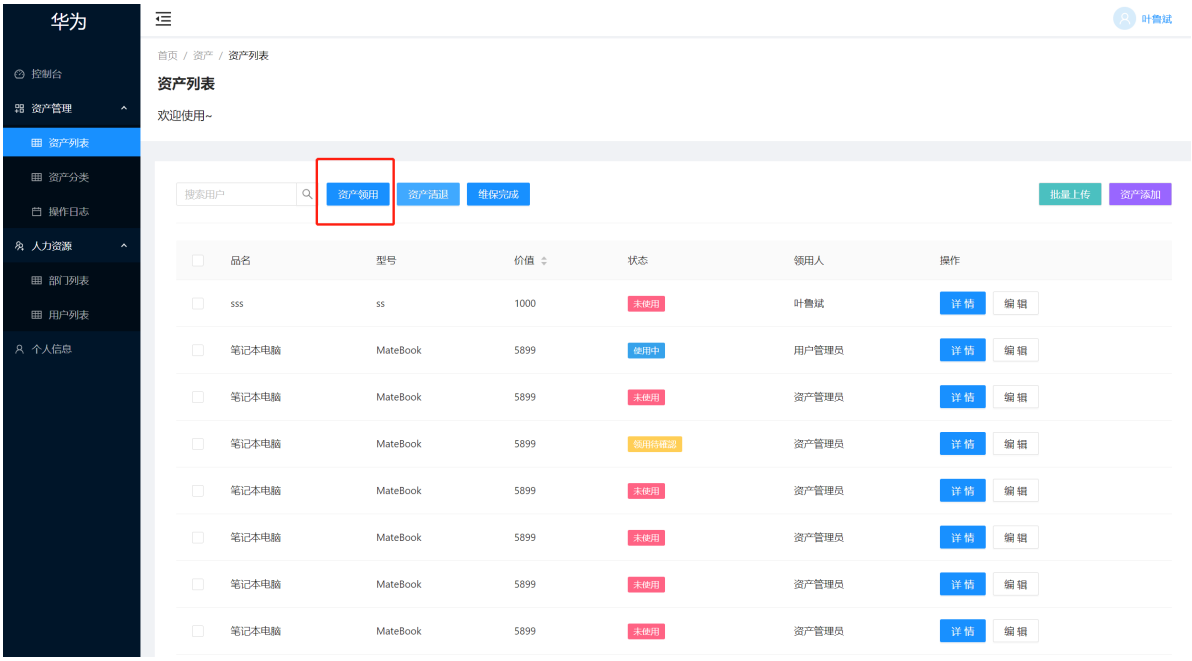
当前员工查看自己名下挂账了哪些资产。




# 8 资产使用

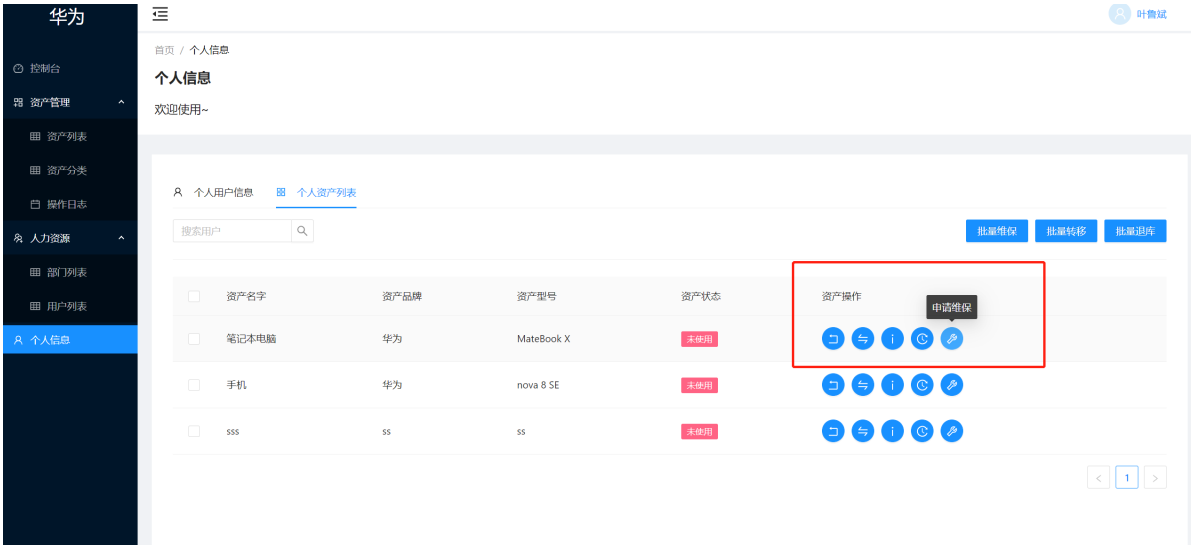
## 8.1 资产领用

员工向资产管理员领用资产。一次可以领用多个资产。员工先提交领用申请，经过主管审批后，资产管理员确认之 后，资产会挂账到当前员工名下。



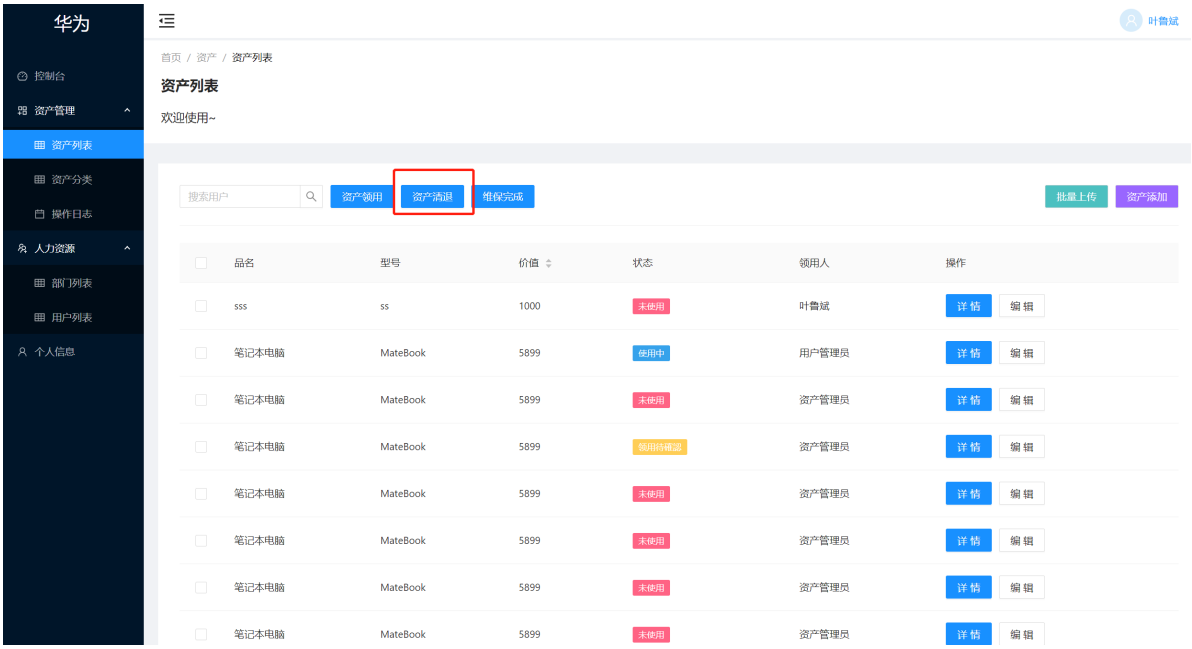

## 8.2 资产维保

员工提交资产维修保养申请，维保责任人接到申请后将资 产拿去维护处理。完成维护之后再交回给当前使用人。



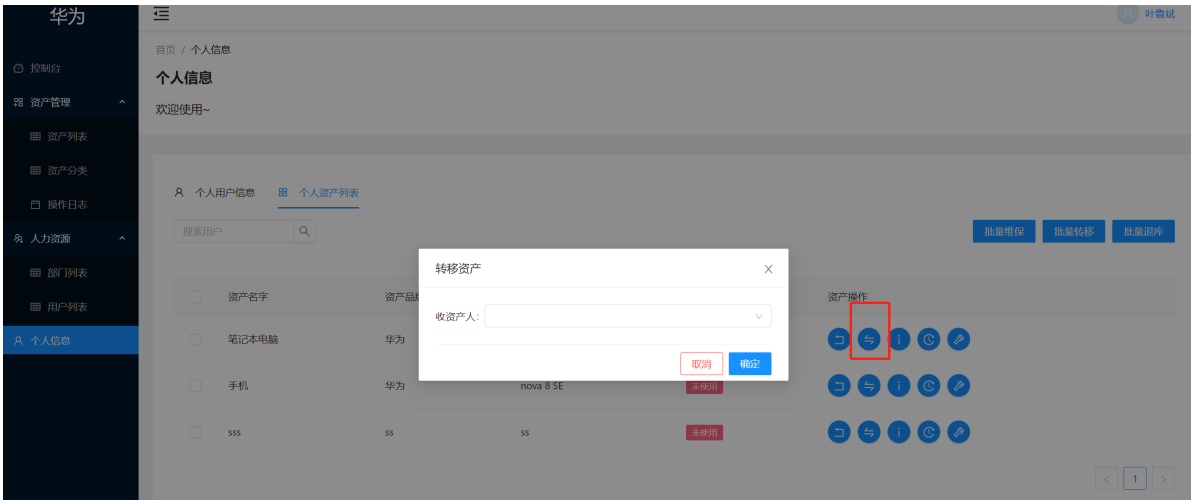

## 8.3 资产清退

资产管理员对已经服役期满或损坏报废的资产进行清退处理。



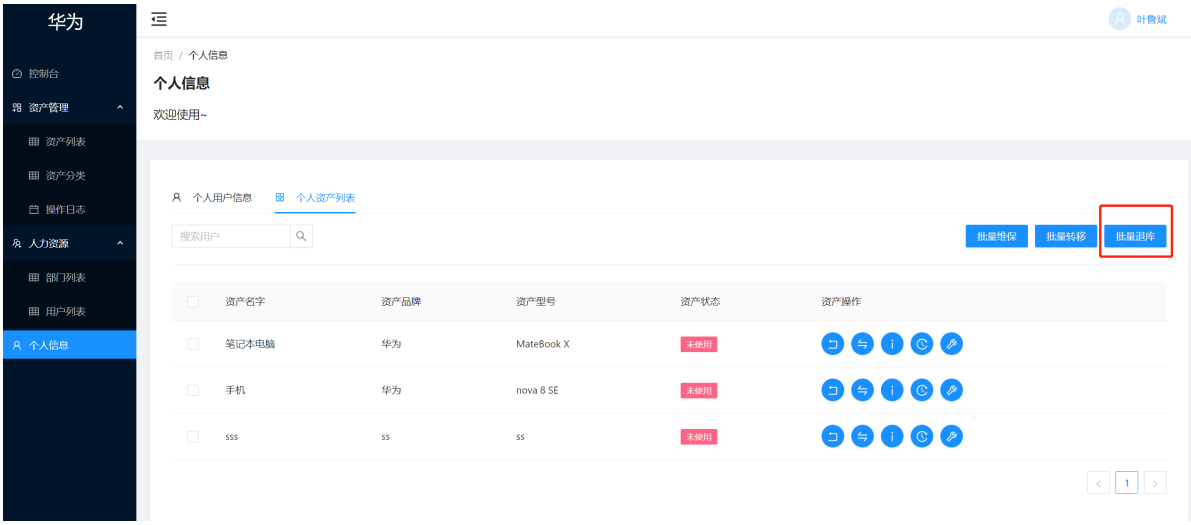

## 8.4 资产转移

员工将某些资产转移给另一个员工。资产转移提交申请后，经过主管审批通过后，资产的挂账人和部门切换为新的员工和部门。资产的位置也跟着迁移到新位置。




## 8.5 资产退库

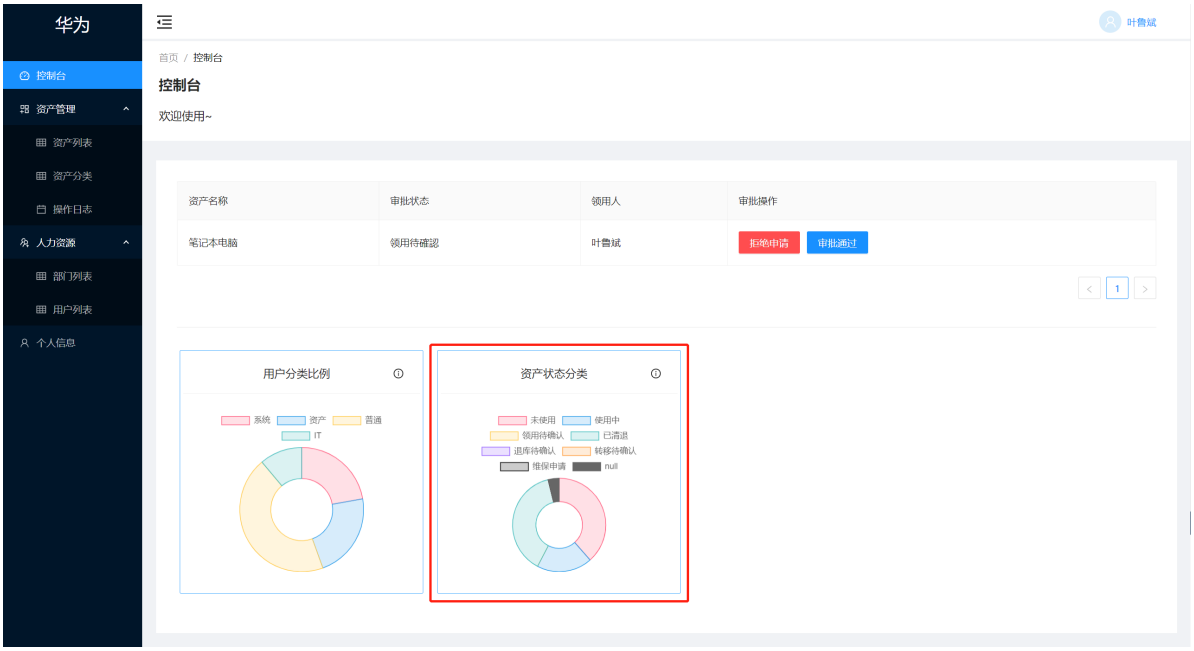
员工的资产不再使用时，可以提交资产退库申请。当前部门的资产管理员收到申请后进行确认处理，资产就移交回 资产管理员名下，待再次被领用。




# 9 资产分析

## 9.1 资产统计

资产管理员可以查看自己管辖内的资产的统计情况。包括：有多少资产；不同状态的资产分布；不同部门的资产分布；资产净值的变化曲线；




## 9.2 资产折旧

系统自动根据资产使用年限，根据年限平均模式，对所有资产进行折旧运算，更新净值。

实现方式在后台，会根据使用年限，自动更新在资产详情里。

# Part B 前端技术框架和实现方式

# 1 Introduction

我们的团队使用React框架来创建企业固定资产管理应用程序的前端。React是Facebook创建的开源JavaScript库。 但是，由于React只关心将数据呈现给DOM，因此需要使用其他库来进行状态管理，路由和其他功能。我们的 应用程序使用的其他JavaScript库是Axios，Chart和React-Router。我们使用React的组件库AntDesign来创建 美观，优雅的交互式用户界面。本文档将描述我们的团队如何使用React和AntDesign实现该应用程序。

# 2 React

## 2.1 Set Up

为了设置React环境，您可以在GitHub上克隆我们的项目，称为asset-app-frontend。然后在项目目录中，运行 以下命令以安装必要的依赖项并查看开发服务器。

```
$ npm install
$ npm start
```

这将在http://localhost上打开服务器。它应该显示一个登录页面。测试帐户的用户名和密码分别为user test和12345。

## 2.2 App.js and Index.js

App.js和Index.js是React的两个默认JavaScript文件。Index.js是所有node.js应用程序的传统入口和实际入口。

在React中，它的代码告诉我们要渲染什么以及在哪里渲染。下面是其代码实现。

```
import React from 'react';
import ReactDOM from 'react-dom'; import './index.css';

import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render( <App />, document.getElementById('root'));
serviceWorker.unregister();
```

另一方面，App.js包含React应用程序的根组件。因为每个视图和组件都是通过React中的层次结构来处理的，其中< App / >是层次结构中最顶层的组件。已使用路由器对其进行了修改，以帮助我们在整个应用程序中导航。这将在React-Router部分中进一步说明。

```
import React from 'react'; import './App.css';
import { BrowserRouter, Route, Switch} from 'react-router-dom'; import Login
from "./views/login/index";
import MainPageLayout from "./views/main/index";
import PrivateRouter from './components/privateRouter/index';

function App() { return (
```

```
<BrowserRouter>
<Switch>
<Route exact component = {Login} path = "/"/>
<PrivateRouter component={MainPageLayout} path = "/main"/>
</Switch>
</BrowserRouter>
);
}
export default App;
```

在上面的代码段中，初始组件将是“登录”页面。登录后，用户将到达MainPageLayout组件，该组件处理应

用程序的布局，包括侧栏导航，标题栏和内容。

## 2.3 Code Structure

React组件结构的核心思想是将页面中的每个区域或元素都视为一个组件。组件是指同时包含html，css，js和image元

素的聚合。所有组件的基本结构可以分为几个部分。伪代码如下所示：

```
import React, { Component } from 'react' import ... (1)

class MainPageLayout extends Component { constructor () { (2)
super(); this.state = { (3)
loading: true,
... };
}
... (4)

render() { (5) return(
<div> </div> (6)
)
};
}
```

1. 导入所有必要的库
2. React允许我们将组件定义为类或函数。
3. 状态是可更新的结构，用于包含有关组件的数据或信息。可以根据用户操作或系统更改随时间更改组件 中的状态。
4. 包含组件所需的功能。例如，在安装React组件后立即调用componentDidMount()函数。此函数通常调

用loadData()函数，该函数检索填充组件所需的数据。

5. 渲染是我们必须在React.Components的子类中定义的唯一方法。调用它时，应检查this.props和this.state并

返回React Native DOM组件或用户定义的复合组件。

6. 包含我们的React元素，例如按钮，模式，表单等。

## 2.4 React-Router

如引言中所述，React默认不支持路由。路由器帮助创建URL，并帮助用户在组成应用程序的不同URL之间导航。它允许用户在保留用户状态的同时在应用程序的各个组件之间移动。该应用程序使用React Router，这是React的流行路由框架。该软件包是使用npm安装的：

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

如上一节中的代码片段所示，路由器已放置在App.js中。PrivateRouter的代码如下：

```
import React from "react";
import {Route, Redirect} from "react-router-dom"; import {getToken} from
"../../utils/cookies";

const PrivateRouter = ({component: Component, ...rest}) => {

return (
<Route {...rest} render={routeProps => (
getToken() ? <Component {...routeProps} /> : <Redirect to="/"/>
)} />
);
};
export default PrivateRouter;
```

由于App.js位于层次结构的顶部，因此在用户成功登录后，在App.js组件中配置ReactRouter允许用户访问 应用程序的所有部分。上面的代码段处理了应用程序不同组件/页面之间的切换。GetToken确认用户确实已正 确登录。该路由器还防止用户简单地输入不同组件的URL来访问它们。但是，如果这样做，它将重定向到“登录”页面。该应用程序分为7个不同的页面：

1. Dashboard: essential information
2. Asset List: available assets
3. Asset Classification: asset hierarchy
4. History: asset operation log
5. Department List: department hierarchy
6. User List: list of users
7. Personal Page: about me page and personal asset list

1. 仪表板：基本信息
2. 资产列表：可用资产
3. 资产分类：资产层次
4. 项目历史记录：资产操作日志
5. 部门列表：部门层次
6. 用户列表：用户列表
7. 个人页面：关于我的页面和个人资产列表

## 2.5 Axios

为了与后端api通信，应用程序必须发送HTTP发布请求以检索有关用户，部门和资产的信息。Axios是基于$http服务的轻量级HTTP。Axios也是基于承诺的。这意味着它可以与JavaScript的async和await一起使用。该软件包是使用npm安装的：

```
$ npm install axios
```

post请求后函数在项目目录的/src/api中声明和定义。axios请求分为四种类型：

- 请求资产信息（assets.js）
- 请求用户信息（users.js）
- 请求部门信息（departments.js）
- 请求帐户信息（account.js）

The process to retrieve a list of all assets is as follows.

检索所有资产列表的过程如下

1. Declare Request Function

```
export function ListAssets(data) { return service.request({
    url: "/asset_list/", method: "post", data,
    });
}
```

该请求的目标api是https://assets-app-backend-marteam.app.secoder.net/asset list/，方法是post，data是

请求的内容

2. 当服务器使用HTTP成功代码进行响应时，Axios将履行请求承诺。或服务器响应HTTP错误时拒绝请求

承诺。

3. 使用then()，catch() 处理响应。

```
import { ListAssets } from "../../../api/assets";
...

ListAssets(data).then(response =>  { if(response.data.code === 0) {...} else
{...}}).catch(error => { message.error ( error.toString(); )};
...
```

数据是发送到后端的请求内容。如果请求正确，则“then”主体将根据响应类型处理响应数据。“catch”主

体将处理在请求期间发生的任何错误

## 2.6 Other

该应用程序使用的其他两个JavaScript库是Chart.js和React-papaparse。Chart.js通过使用HTML5canvas元素帮 助将数据绘制到图表上。它用于以甜甜圈图的形式显示有关资产状态和用户分类的信息。React-Papaparse是功 能强大的浏览器内React CSV解析器。它允许资产管理器通过上传CSV格式的文档一次添加多个资产。这两个都可以通过npm安装，如下所示：

```
$ npm install chartjs
$ npm  react-papaparse
```

为了使用chart.js，我们必须导入库，创建图表，并用数据填充它。下图使用从用户列表api检索的数据。

```
...
import Chart from "chart.js";
...

GetUserListApi(token).then(response => {
    ...

    new Chart(ctx, { type: "doughnut", data: {
        labels: labels, datasets: [{
            label: label, data: data,
            backgroundColor: [...], borderColor: [...], borderWidth: 1
            }]
        }
    });
}
...

render() {
    return ( ...
        <canvas className="myChart" id="UserDistributionChart" width="250px"
height="200px"></canvas>
    )
}
```

界面如下图所示：

| | |
|---|---|
| | |

Figure 1: User List classification chart

为了使用React-papaparse，我们首先必须创建一个模板，用户可以通过按下按钮下载该模板。然后可以填

写，上传该模板，并将其发送到后端进行处理。实现如图所示

```
...
import { CSVReader } from 'react-papaparse';
...

<CSVReader onDrop={this.handleOnDrop} onError={this.handleOnError} noDrag
addRemoveButton onRemoveFile={this.handleOnRemoveFile}\>
    <span>点击上传</span>
</CSVReader>
...
<Button className="button-download" icon={<DownloadOutlined />} href="..."
type="link"> 下载模板
</Button>
```

界面如下图所示：

| | |
|---|---|
| | |

Figure 2: CSV

# 3 Ant Design

## 3.1 Ant Design of React

AntDesign（又名antd）包含一组用于构建丰富的交互式用户界面的高质量组件。它包含了我们在整个项目中

都使用过的一组高质量的React组件，例如按钮，表格和输入框，这免除了我们使用css / scss单独设置每个组

件样式的麻烦。可以使用npm安装该库：

```
$ npm install antd
```

例如，为了使用库中的一个组件，如果需要使用按钮，则必须按以下方式导入它：

```
import { Button } from "antd";
```

antd的组件和典型的React组件之间没有区别。因此，按钮将包含在类的render（）部分中，如下所示：

```
<Button className= "Button"\>
```

## 3.2 Components

组件的文档可以在https://ant.design/components/overview/中找到。因此，将只讨论应用程序中使用的关键 组件。

### 3.2.1 Message

消息组件将显示在页面的顶部和中心，并且将自动关闭。消息用作响应用户操作的反馈。一些用户操作包括

- 登录
- 信息检索，即用户数据库的检索
- 资产操作，即移动/还资产

在向api发送发布请求后，该消息组件主要用于显示响应消息。例如，如果用户提交了错误的密码，则会出现以 下消息组件。

| | |
|---|---|
| | |

Figure 3: Message Component: Login Error

或者，如果用户尝试执行超出其权限范围的操作，例如，如果普通用户尝试操纵部门（资产经理保留的权

限），那么他也会收到错误消息

| | |
|---|---|
| | |

Figure 4: Message Component: User does not have that power

### 3.2.2 Modal

模态是显示在当前网页上的对话框或弹出窗口。它允许用户与应用程序进行交互，但不会中断用户的工作流 程。在整个应用程序中使用了模式，以在当前页面上创建一个浮动层，以显示和操纵用户，部门和资产信息。以下是AntDesign的模式组件的一些实现。

Figure 5: Modal Component: Asset Information

| | |
|---|---|
| | |

Figure 6: Modal Component: Manipulate Asset Information

### 3.2.3 Table

表格显示数据。由于应用程序必须具有显示信息列表的能力，因此表是一个重要的组件，因为它使用户可以一

目了然地看到信息行。然后，用户可以独立地或成组地操纵每一行。需要使用表显示的信息包括：

- 用户列表
- 个人资产列表
- 审批/拒绝资产操作列表
- 全资产列表

用户列表表的界面如下所示

Figure 7: User Table List

用户管理人不仅可以查看数据库中所有用户的基本用户信息，还可以分别对其进行操作，比如个人的密码，用

户密码，等等。

### 3.2.4 Tree

树是分层列表结构的组件。由于该应用程序必须具有显示部门和用户的层次结构的能力，因此使用树组件表示这两件事之间的层次关系。可以展开，折叠树，并可以选择树上的节点。该应用程序使用两棵树，一棵代表部门之间的层次关系，一棵代表资产之间的层次关系。部门树的界面如下所示：

| | |
|---|---|
| | |

Figure 8: Asset Category Group

### 3.2.5 Menu

菜单用于导航。该应用程序利用侧面导航菜单来允许用户快速，高效地浏览站点。导航菜单可以展开和折叠。

导航菜单如下所示：

| | |
|---|---|
| | |

Figure 9: Navigation Menu

# Part C 后端技术框架和实现方式

## 1 简介

考虑各方面因素，包括系统规模和负载，考虑了各种技术栈后（例如Nginx，Apache等作为网页服务器，PHP，Java等作为后端处理语言，亦考虑过基于Python的Flask），我们使用了基于Python的Django作为后端开发的核心框架，方便部署。

## 2 技术栈

Python 3.8.3

Django 3.1.2

Django Rest Framework 3.12.1

## 3 处理客户端的HTTP请求

### 3.1 本地调试

在本地调试中，我们使用了Django自带的runserver作为网页服务器，处理接入的HTTP请求，启动服务器指令如下：

```
python manage.py runserver 0.0.0.0:8000
```

## 3.2 Docker部署

在云端部署中，我们使用了Nginx作为网页服务器，处理接入的HTTP请求，设置静态路径提供静态内容，直接由Nginx处理返回，动态内容（包含所有API接口）透过uWSGI把动态传给Python的Django处理后再返回.

# 4 数据库调用

我们使用Django提供的Models机制与数据库对接，在models.py编写数据库表格储存资料，如下图所示：

```
class StaffGroup(models.Model):
    unique_id = models.CharField(unique=True, max_length=100)
    display_name = models.CharField(max_length=100)

    def __str__(self):
        return self.display_name


class Department(models.Model):
    unique_id = models.CharField(unique=True, max_length=100)
    display_name = models.CharField(max_length=100)
    location = models.CharField(max_length=100)
    parent = models.ForeignKey("self", on_delete=models.PROTECT, null=True, blank=True)

    def __str__(self):
        return self.display_name


class Staff(models.Model):
    unique_id = models.CharField(unique=True, max_length=100)
    group = models.ForeignKey(StaffGroup, on_delete=models.PROTECT)
    department = models.ForeignKey(Department, on_delete=models.PROTECT)
    display_name = models.CharField(unique=True, max_length=100)
    username = models.CharField(unique=True, max_length=100)
    password = models.CharField(max_length=100)
    locked = models.BooleanField(default=False)

    def __str__(self):
        return self.display_name
```

使用如下指令migrate生效到数据库：

```
python manage.py makemigrations
python manage.py migrate
```

具体数据库设计可以参看下一章节Part D数据库设计。

# 5 实现接口处理

## 5.1 处理函数

我们在Django的views编写后端的操作逻辑，Django已为我们解析好请求，并封装成request参数，传入到我们自己编写的函数，如下图为用户登录login的接口：

```python
@csrf_exempt
def login(request):
    # check if the request is by POST
    if request.method != 'POST':
        return method_not_allow()

    # check if the request body is in valid json format
    try:
        client_json = json.loads(request.body)
    except:
        return JsonResponse({"code": -1, "message": "Requested body is not in valid JSON format"})

    try:
        # main function logic

        username = client_json["username"]
        password = client_json["password"]
        if models.Staff.objects.filter(username=username, password=password).exists():
            return JsonResponse({"code": 0, "message": "Login successful",
                                 "token": auth.login(username, password)})
        else:
            return JsonResponse({"code": -4, "message": "Login failed"})

    # deal with errors
    except KeyError:
        return JsonResponse({"code": -2, "message": "Parameters not correct"})
    except:
        return JsonResponse({"code": -3, "message": "Unknown Error"})
```

执行完成后，返回一个JsonResponse，Django会把把资料封装成Json并返回给客户端。基本所有接口都是此原理。

## 5.2 用户登录认证机制

我们使用token机制对用户进行认证。用户登陆时，如果身份认证成功（用户名和密码正确），则会生成一个token，这个token与该用户对应，并在后端记录下来，最初打算使用redis，后面改用了数据库表格储存，可以实现用户长期登录，直至用户主动登出为止。

客户端调用接口时需要附带token，后端收到请求后，会检查这个token对应的账号以及相应权限，只允许有该权限的用户才能执行操作，否则返回错误信息。

## 5.3 接口功能处理

后端接口验证用户身份后，进入功能处理流程，按照各个接口的具体功能作相应实现，过程中透过models访问数据库进行读写，查找或修改资料。所有接口列表和具体功能可以参照最后一章Part E接口介绍。

# Part D 数据库设计

基于《企业固定资产管理应用》文檔进行了分析，定义了固定资产管理应用系统数据库。

## 1 数据库的分布特性

企业固定资产管理数据库，支持企业管理固定资产的归属和流转。包括：资产入库、资产领用、资产借用、资产转移、资产清退、资产维修、资产卡片查看和资产标签打印等。支持基本的审批流程。

# 2 数据库技术

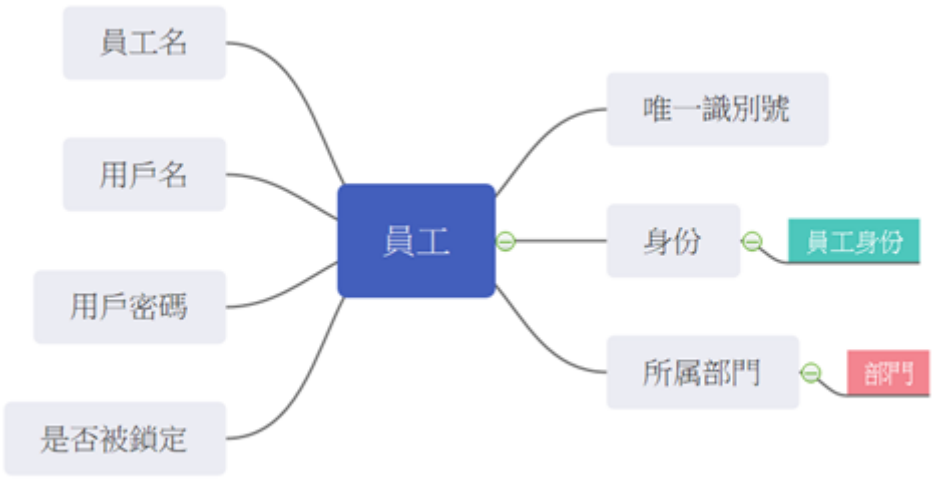
本次项目中数据库规模不大，使用了SQLite，以单个文件形式储存数据库资料，不另外设立MySQL服务器。在Django中，使用了Django自带的sqlite3引擎对接数据库，让后端使用models访问使用数据库。

# 3 数据库架构

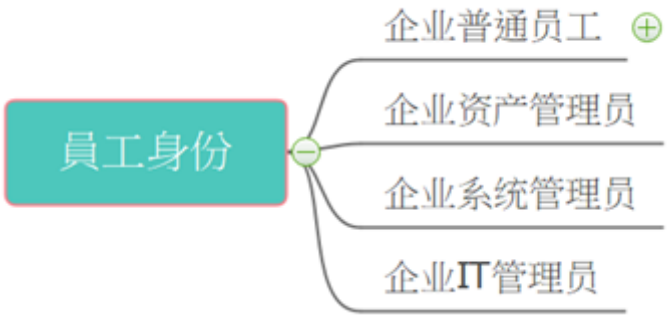
## 3.1 数据库结构图

### 3.1.1 员工相关模块

#### 3.1.1.1 员工模块:

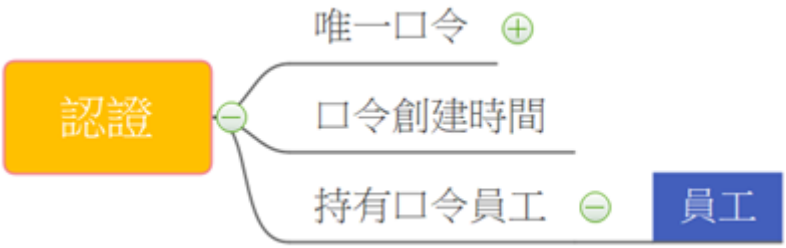



#### 3.1.1.2 员工身份模块



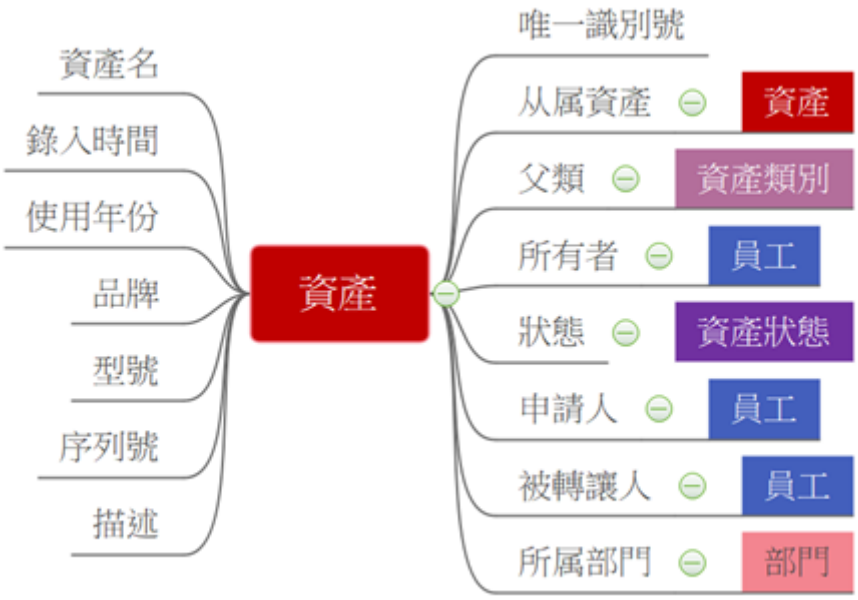
#### 3.1.1.3 部门模块


#### 3.1.1.4 员工认证模块

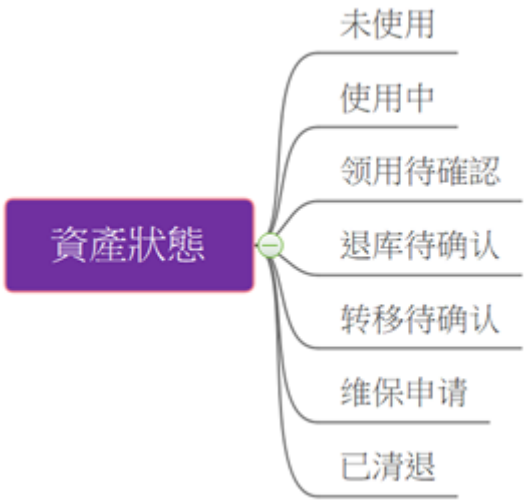



### 3.1.2 资产相关模块

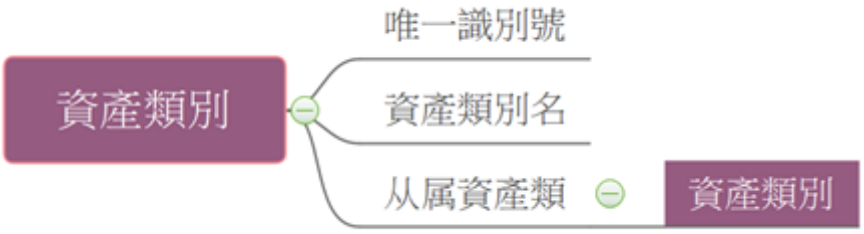
#### 3.1.2.1 资产模块:



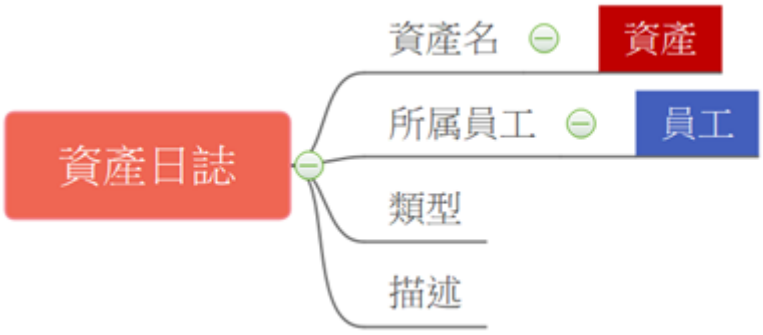

#### 3.1.2.2 资产状态模块:




**3.1.2.3** 资产类别模块:




**3.1.2.4** 资产日志模块:




## 3.2 数据库结构表

### 3.2.1 员工相关模块

**3.2.1.1** 员工模块:

| 字段名 | 数据类型 | 字段说明 | 描述 |
| --- | --- | --- | --- |
| **unique_id** | CharField | 唯一识别号 | 凭此识别符可以确定唯一员工 |
| **group** | ForeignKey | 身份 | 绑定到员工身份模块 |
| **department** | ForeignKey | 所属部门 | 绑定到员工部门模块 |
| **display_name** | CharField | 员工名 | 员工姓名 |
| **username** | CharField | 用户名 | 登陆名 |
| **password** | CharField | 用户密码 | 登陆密码 |
| **locked** | BooleanField | 锁定状态 | 若为true,表示该用户被锁定; 若为false,则表示没有被锁定 |

**3.2.1.2** 员工身份模块:

| Unique id | 字段说明 | 描述 |
| --- | --- | --- |
| **generalstaff** | 企业普通员工 | 企业内的每一个员工 |
| **assetadmin** | 企业资产管理员 | 每个部门有自己的资产管理员，负责管理仓库内的资产，并 为员工提供资产相关服务。 |
| **systemadmin** | 企业系统管理员 | 负责管理用户、角色、日志 |
| **itadmin** | 企业IT管理员 | 配置初始数据和参数，增加新功能，维护应用里的核心数据角色 |

**3.2.1.3** 部门模块:

| 字段名 | 数据类型 | 字段说明 | 描述 |
| --- | --- | --- | --- |
| **unique_id** | CharField | 唯一识别号 | 凭此识别符可以确定唯一部门 |
| **display_name** | CharField | 部门名 | 部门名 |
| **location** | CharField | 部门位置 | 部门所在位置 |
| **parent** | ForeignKey | 父部门 | 每个部门都会有一个父部门，绑定到员工部门模块 |

**3.2.1.4** 员工认证模块**:**

| 字段名 | 数据类型 | 字段说明 | 描述 |
| --- | --- | --- | --- |
| **token** | CharField | 唯一口令 | 凭此口令可以确定用户是否拥有各类权限 |
| **authtime** | DateTimeField | 口令创建时间 | 口令创建时间 |
| **user** | ForeignKey | 持有口令员工 | 绑定到员工模块 |

## **3.2.2** 资产相关模块

**3.2.2.1** 资产模块**:**

| 字段名 | 数据类型 | 字段说明 | 描述 |
| --- | --- | --- | --- |
| **unique_id** | CharField | 唯一识别号 | 凭此识别符可以确定唯一资产 |
| **under** | ForeignKey | 从属资产 | 若本资产是其他资产的配件，则绑定为该资产 |
| **group** | ForeignKey | 父类 | 每个资产都属于一个资产类别，绑定到资产类别模块 |
| **belongs_to** | ForeignKey | 所有者 | 绑定到一个员工模块 |
| **status** | ForeignKey | 状态 | 绑定到一个资产状态模块 |
| **applicant** | ForeignKey | 申请人 | 绑定到一个员工模块 |
| **belongs_to_pending** | ForeignKey | 被转让人 | 绑定到一个员工模块 |
| **belongs_department** | ForeignKey | 所属部门 | 绑定到员工部门模块 |
| **display_name** | CharField | 资产名 | 同左 |
| **date** | DateField | 录入时间 | 同左 |
| **use_year** | PositiveIntegerField | 使用年份 | 资产寿名 |
| **brand** | CharField | 品牌 | 同左 |
| **model** | CharField | 型号 | 同左 |
| **serial_number** | CharField | 序列号 | 同左 |
| **remarks** | CharField | 描述 | 同左 |

**3.2.2.2** 资产状态模块**:**

| Unique id | 字段说明 |
| --- | --- |
| **unused** | 未使用 |
| **inuse** | 使用中 |
| **get_confirming** | 领用待确认 |
| **return_confirming** | 退库待确认 |
| **transfer_confirming** | 转移待确认 |
| **repair_apply** | 维保申请 |
| **deleted** | 已清退 |

**3.2.2.3** 资产类别模块:

| 字段名 | 数据类型 | 字段说明 | 描述 |
| --- | --- | --- | --- |
| unique_id | CharField | 唯一识别号 | 凭此识别符可以确定唯一资产类别 |
| display_name | CharField | 资产类别名 | 同左 |
| parent | ForeignKey | 从属资产类 | 每个资产类别都有一个父类资产类别，绑定到资产类别模块 |

**3.2.2.4** 资产日志模块:

| 字段名 | 数据类型 | 字段说明 | 描述 |
| --- | --- | --- | --- |
| **asset** | ForeignKey | 资产名 | 日志所描述的资产对象，绑定到资产模块 |
| **related_user** | ForeignKey | 所属员工 | 日志所描述的资产对象的所有者，绑定到员工模块 |
| **type** | CharField | 类型 | transfer/apply/return/delete分别对应转移/领用/退库/清退 |
| **description** | CharField | 描述 | 描述所属员工对该资产进行了何种操作 |

# Part E 接口介绍

设计后端接口时对《企业固定资产管理应用》文档进行了分析，对django提供的数据库接口进行封装并提供给前端。

后端接口按功能大致可分为4类：用户登录、数据查看、数据管理、业务流程。这些接口都通过对url的post请求访问，请求的参数通过请求体传递，格式为json。

调用接口的返回数据格式也为json，包含返回码，返回信息以及请求的数据（若有），返回码为0代表成功，其他值代表不同类型的错误，详细出错信息由返回信息提供。

# 1 用户登录

用户登录时发送账户和密码，服务器验证通过后返回一个随机字串作为token，用户的后续操作都需要通过token验证身份。除登录接口外，以下接口的请求默认带有token参数。

| 接口 | 功能 | 参数 |
| --- | --- | --- |
| 用户登录 | 登录资产管理平台 | 用户名、密码 |

# 2 数据查看

数据查看接口用于查看按指定条件筛选的数据，包括对用户个人信息、资产信息、部门信息、历史记录的查看和搜索。这类接口不会对数据库进行改动，大部分对所有登录用户开放。

| 接口 | 功能 | 额外参数 |
| --- | --- | --- |
| 用户信息 | 返回当前登录用户的详细个人信息 | 无 |
| 用户列表 | 返回一个列表，包含所有用户的基本信息 | 无 |
| 员工挂账资产 | 返回员工名下挂账的资产 | 员工识别码 |
| 资产信息 | 返回单个资产的信息 | 资产识别码 |
| 资产列表 | 返回一个列表，包含所有资产的基本信息 | 无 |
| 资产搜索 | 按搜索字串在资产名和描述中搜索 | 搜索字串 |
| 资产历史 | 返回资产的领用历史 | 资产识别码 |
| 资产分组 | 返回一个层级列表，包含资产分类树 | 无 |
| 待审批列表 | 返回待审批领用、退库、维保申请的列表 | 无 |
| 部门列表 | 返回一个层级列表，包含部门结构树 | 无 |

# 3 数据管理

数据管理类接口用于直接对部分的数据添加、删除、修改，这类接口需要有相应的管理员权限才可以访问。

## 3.1 用户管理

用户管理接口的访问需要系统管理员权限， 除创建用户外都有指定用户的唯一识别码作为参数

| 接口 | 功能 | 额外参数 |
| --- | --- | --- |
| 创建用户 | 根据参数创建一个新用户，并在内部自动生成一个唯一识别码 | 用户名、初始密码、用户角色、所属部门 |
| 删除用户 | 删除指定用户，删除前检查其名下是否还有挂账资产，只允许删除无资产用户 | 无 |
| 重置密码 | 修改指定用户的密码 | 新密码 |
| 锁定/解锁用户 | 锁定/解锁指定用户，被锁定用户无法登录 | 锁定状态 |
| 设置用户角色 | 改变指定用户的用户角色 | 用户角色 |

## 3.2 资产管理

资产管理接口的访问需要资产管理员权限

| 接口 | 功能 | 参数 |
| --- | --- | --- |
| 录入资产 | 录入列表资产，一次可录入多个 | 需要录入的资产列表 |
| 修改资产信息 | 修改单个资产信息 | 资产识别码、要修改的信息 |
| 修改资产分类树 | 资产分类树的增删改，删除一个类别时要求没有资产属于这个类别且这个类别没有子类 | 资产类别、要修改的信息 |

## 3.3 部门信息管理

部门信息管理接口的访问也需要资产管理员权限

| 接口 | 参数 |
| --- | --- |
| 添加部门 | 新部门的名称、位置、父部门 |
| 删除部门 | 部门识别码 |
| 修改部门 | 部门识别码、要修改的信息 |

# 4 业务流程

资产转移的业务流程需要多方协同完成，为保证业务流程的完整性，流程相关数据禁止直接修改，而是通过业务流程接口间接改动并留下操作记录。这类接口又可以再分为申请接口和确认接口

| 接口 | 功能 | 参数 |
| --- | --- | --- |
| 资产领用申请 | 领用未使用的资产，提交申请后进入待审批状态 | 要领用的资产列表 |
| 资产领用确认 | 通过或拒绝领用申请，通过后资产转移到申请人名下 | 资产识别码、是否通过 |
| 资产清退 | 清退列表中的资产 | 资产列表 |
| 资产转移申请 | 申请将自身名下的资产转移到其他用户 | 资产列表、转移对象 |
| 资产转移确认 | 通过或拒绝转移申请 | 资产识别码、是否通过 |
| 资产退库申请 | 申请将自身名下的资产退库 | 资产列表 |
| 资产退库确认 | 通过或拒绝申请，通过后资产挂账到管理员名下 | 资产识别码、是否通过 |
| 资产维保申请 | 对使用中的资产申请维保 | 资产列表 |
| 资产维保完成确认 | 确认资产维保完成 | 资产识别码 |