下面我将基于Ant Design + React实现一个用户管理UI，分两个阶段进行开发：第一阶段使用Mock API实现功能，第二阶段连接真实API。

一、项目搭建与配置

1. **创建React项目**：

```
npx create-react-app user-management
cd user-management
```

1. **安装依赖**：

```
npm install antd axios @ant-design/icons
npm install @ant-design/v5-patch-for-react-19 --save
```

1. **配置Ant Design**（在 `src/index.js` 中）：

```
import 'antd/dist/reset.css';
import '@ant-design/v5-patch-for-react-19';
```

二、第一阶段：Mock API实现

1. Mock服务配置（`src/mock/userApi.js`）

```
import { v4 as uuidv4 } from 'uuid';

let users = [
  { id: '1', name: '张三', gender: '男', age: 25, email: 'zhang@example.com', phone: '13800138000' },
  { id: '2', name: '李四', gender: '女', age: 30, email: 'li@example.com', phone: '13900139000' }
];

export const mockUserApi = {
  getUsers: () => new Promise(resolve =>
    setTimeout(() => resolve({ data: users }), 500)),

  createUser: (user) => new Promise(resolve => {
    setTimeout(() => {
      const newUser = { ...user, id: uuidv4() };
      users = [...users, newUser];
      resolve({ data: newUser });
    }, 500);
  }),

  updateUser: (id, user) => new Promise(resolve => {
    setTimeout(() => {
      users = users.map(u => u.id === id ? { ...u, ...user } : u);
      resolve({ data: { id, ...user } });
    }, 500);
  }),

  deleteUser: (id) => new Promise(resolve => {
    setTimeout(() => {
      users = users.filter(u => u.id !== id);
      resolve({ data: { id } });
    }, 500);
  })
};
```

2. 用户管理界面（`src/components/UserManagement.jsx`）

```jsx
import React, { useState, useEffect } from 'react';
import { Table, Button, Modal, Form, Input, InputNumber, message } from 'antd';
import { PlusOutlined } from '@ant-design/icons';
import { mockUserApi } from '../mock/userApi';

const UserManagement = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(false);
  const [visible, setVisible] = useState(false);
  const [currentUser, setCurrentUser] = useState(null);
  const [form] = Form.useForm();

  const columns = [
    { title: '姓名', dataIndex: 'name', key: 'name' },
    { title: '性别', dataIndex: 'gender', key: 'gender' },
    { title: '年龄', dataIndex: 'age', key: 'age' },
    { title: '邮箱', dataIndex: 'email', key: 'email' },
    { title: '电话', dataIndex: 'phone', key: 'phone' },
    {
      title: '操作',
      key: 'action',
      render: (_, record) => (
        <>
          <Button type="link" onClick={() => handleEdit(record)}>编辑</Button>
          <Button type="link" danger onClick={() => handleDelete(record.id)}>删除</Button>
        </>
      ),
    },
  ];

  const fetchUsers = async () => {
    setLoading(true);
    try {
      const response = await mockUserApi.getUsers();
      setUsers(response.data);
    } catch (error) {
      message.error('获取用户列表失败');
    } finally {
      setLoading(false);
```

```
    }
  };

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

  const handleCreate = () => {
    form.resetFields();
    setCurrentUser(null);
    setVisible(true);
  };

  const handleEdit = (user) => {
    form.setFieldsValue(user);
    setCurrentUser(user);
    setVisible(true);
  };

  const handleDelete = async (id) => {
    Modal.confirm({
      title: '确认删除',
      content: '确定要删除该用户吗？',
      onOk: async () => {
        await mockUserApi.deleteUser(id);
        message.success('删除成功');
        fetchUsers();
      },
    });
  };

  const handleSubmit = async () => {
    try {
      const values = await form.validateFields();
      if (currentUser) {
        await mockUserApi.updateUser(currentUser.id, values);
        message.success('更新成功');
      } else {
        await mockUserApi.createUser(values);
        message.success('创建成功');
```

```
      }
      setVisible(false);
      fetchUsers();
    } catch (error) {
      console.error('提交失败:', error);
    }
  };

  return (
    <div style={{ padding: 24 }}>
      <div style={{ marginBottom: 16 }}>
        <Button
          type="primary"
          icon={<PlusOutlined />}
          onClick={handleCreate}
        >
          添加用户
        </Button>
      </div>

      <Table
        columns={columns}
        dataSource={users}
        rowKey="id"
        loading={loading}
        pagination={{ pageSize: 10 }}
      />

      <Modal
        title={currentUser ? "编辑用户" : "创建用户"}
        open={visible}
        onOk={handleSubmit}
        onCancel={() => setVisible(false)}
      >
        <Form form={form} layout="vertical">
          <Form.Item
            name="name"
            label="姓名"
            rules={[{ required: true, message: '请输入姓名' }]}
          >
```

```
            <Input placeholder="张三" maxLength={50} />
          </Form.Item>

          <Form.Item
            name="gender"
            label="性别"
            rules={[{ required: true, message: '请输入性别' }]}
          >
            <Input placeholder="男/女" maxLength={10} />
          </Form.Item>

          <Form.Item
            name="age"
            label="年龄"
            rules={[{ required: true, message: '请输入年龄' }]}
          >
            <InputNumber min={0} style={{ width: '100%' }} />
          </Form.Item>

          <Form.Item
            name="email"
            label="邮箱"
            rules={[
              { required: true, message: '请输入邮箱' },
              { type: 'email', message: '邮箱格式不正确' }
            ]}
          >
            <Input placeholder="example@domain.com" maxLength={100} />
          </Form.Item>

          <Form.Item
            name="phone"
            label="电话"
            rules={[{ required: true, message: '请输入电话' }]}
          >
            <Input placeholder="13800138000" maxLength={20} />
          </Form.Item>
        </Form>
      </Modal>
    </div>
```

```
  );
};

export default UserManagement;
```

3. 主应用入口（`src/App.js`）

```
import UserManagement from './components/UserManagement';

function App() {
  return (
    <div className="App">
      <UserManagement />
    </div>
  );
}

export default App;
```

三、第二阶段：连接真实API

1. 创建API服务（`src/services/userService.js`）

```
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/users';

export const userService = {
  getUsers: () => axios.get(API_URL),
  createUser: (user) => axios.post(API_URL, user),
  updateUser: (id, user) => axios.put(`${API_URL}/${id}`, user),
  deleteUser: (id) => axios.delete(`${API_URL}/${id}`)
};
```

2. 修改用户管理组件

更换 mockUserApi ==>userService

```
// 在UserManagement.jsx中替换：
// import { mockUserApi } from '../mock/userApi';
import { userService } from '../services/userService';

// 修改所有API调用方法：
const fetchUsers = async () => {
  setLoading(true);
  try {
    const response = await userService.getUsers();
    setUsers(response.data);
  } catch (error) {
    message.error('获取用户列表失败');
  } finally {
    setLoading(false);
  }
};

// 同样修改create/update/delete方法中的API调用
```

3. 添加请求拦截器（`src/utils/http.js`）

```javascript
import axios from 'axios';

const instance = axios.create({
  baseURL: 'http://localhost:8080',
  timeout: 5000
});

// 请求拦截器
instance.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// 响应拦截器
instance.interceptors.response.use(
  response => response.data,
  error => {
    if (error.response) {
      switch (error.response.status) {
        case 401:
          // 处理未授权
          break;
        case 404:
          // 处理未找到
          break;
        default:
          console.error('API错误:', error);
      }
    }
    return Promise.reject(error);
  }
);

export default instance;
```

四、功能特点说明

1. **用户界面**:
   - 响应式表格展示用户数据
   - 表单验证（必填项、邮箱格式、年龄最小值）
   - 操作反馈（成功/错误提示）
2. **Mock API实现**:
   - 使用内存数据模拟CRUD操作
   - 添加500ms延迟模拟网络请求
   - UUID生成唯一ID
3. **Ant Design组件使用**:
   - Table展示数据
   - Modal+Form实现编辑/创建
   - 消息提示和确认对话框
4. **真实API集成**:
   - Axios HTTP客户端
   - 请求/响应拦截器
   - 错误统一处理

五、项目结构

```
src/
├── components/
│   └── UserManagement.jsx
├── services/
│   └── userService.js
├── mock/
│   └── userApi.js
├── utils/
│   └── http.js
├── App.js
└── index.js
```

六、测试建议

1. **Mock阶段测试**:

```
// 在组件中添加测试按钮
<Button onClick={() => {
  mockUserApi.getUsers().then(res => console.log(res));
}}>
  测试Mock API
</Button>
```

1. **真实API测试**:

```
// 在浏览器控制台测试
userService.getUsers().then(console.log);
```

1. **错误处理测试**:

```
// 在userService.js中模拟错误
deleteUser: () => Promise.reject(new Error('模拟错误'))
```