为了开发一个 **采购系统和库存管理系统**，我们需要构建一个完整的系统，包括 **前端用户界面** 和 **后端 API 服务**，并涵盖以下关键功能模块：

1. **组织架构管理**（管理部门、职位和权限）
2. **商品维护**（商品的添加、编辑和删除）
3. **采购录入**（采购订单的创建和管理）
4. **会签审批**（采购订单的审批流程）
5. **统计分析**（采购数据和库存情况的统计）
6. **验收单**（确认采购商品验收）
7. **付款凭证**（生成付款相关凭证）
8. **打印单据**（生成并打印采购和库存相关的单据）

**1. 系统设计与模块划分**

**1.1 模块划分**

* **组织架构管理模块**
  + 管理组织的部门、职位、员工以及相应的权限。
* **商品维护模块**
  + 商品的新增、编辑、删除和查询。
* **采购录入模块**
  + 采购订单录入，包括采购商品、数量、价格、供应商等。
* **会签审批模块**
  + 采购订单的审批，分层级审批机制。
* **统计分析模块**
  + 对采购和库存数据进行统计，包括采购总额、各供应商采购量、库存余量等。
* **验收单模块**
  + 验收订单时生成验收单，确认商品已接收并入库。
* **付款凭证模块**
  + 为采购订单生成付款凭证，记录付款信息。
* **打印单据模块**
  + 支持打印采购订单、验收单、付款凭证等。

**1.2 数据库设计**

1. **组织架构表**
   * departments: 管理各部门信息，如 id、name、manager\_id。
   * employees: 员工信息表，包括员工编号、姓名、部门、职位、权限等。
2. **商品表**
   * items: 商品信息表，包括 id、name、price、quantity、supplier\_id。
   * suppliers: 供应商信息表，包括 id、name、contact\_info。
3. **采购表**
   * purchase\_orders: 采购订单表，包含 id、employee\_id、status、total\_amount、created\_at、approved\_at。
   * purchase\_items: 采购订单详情表，包含 order\_id、item\_id、quantity、price。
4. **审批表**
   * approval\_logs: 采购订单的审批日志，包含 order\_id、employee\_id、status、comment、timestamp。
5. **验收表**
   * receipts: 验收单表，记录 id、purchase\_order\_id、received\_at、status。
6. **付款凭证表**
   * payment\_vouchers: 付款凭证表，记录 id、purchase\_order\_id、amount\_paid、payment\_date。

**2. 关键功能设计**

**2.1 组织架构管理**

1. **组织架构页面**：管理部门、职位、权限分配，新增、编辑和删除组织架构信息。

// departments.js

Page({

data: {

departments: []

},

onLoad() {

this.fetchDepartments();

},

fetchDepartments() {

wx.request({

url: '/api/departments',

method: 'GET',

success: (res) => {

this.setData({ departments: res.data });

}

});

},

addDepartment(name) {

wx.request({

url: '/api/departments',

method: 'POST',

data: { name },

success: () => {

this.fetchDepartments();

}

});

}

});

**2.2 商品维护**

* **商品维护页面**：提供商品的新增、编辑和删除功能，包括商品的名称、价格、库存和供应商信息。

// items.js

Page({

data: {

items: []

},

onLoad() {

this.fetchItems();

},

fetchItems() {

wx.request({

url: '/api/items',

method: 'GET',

success: (res) => {

this.setData({ items: res.data });

}

});

},

addItem(itemData) {

wx.request({

url: '/api/items',

method: 'POST',

data: itemData,

success: () => {

this.fetchItems();

}

});

}

});

**2.3 采购录入**

* **采购录入页面**：输入采购商品、数量、单价，生成采购订单。

// purchaseOrder.js

Page({

data: {

items: [],

selectedItems: []

},

onLoad() {

this.fetchItems();

},

fetchItems() {

wx.request({

url: '/api/items',

method: 'GET',

success: (res) => {

this.setData({ items: res.data });

}

});

},

createPurchaseOrder(orderData) {

wx.request({

url: '/api/purchase\_orders',

method: 'POST',

data: orderData,

success: () => {

wx.navigateTo({ url: '/pages/purchaseOrders/purchaseOrders' });

}

});

}

});

**2.4 会签审批**

* **审批页面**：审批采购订单，包括审批通过或驳回的操作。

// approval.js

Page({

data: {

orders: []

},

onLoad() {

this.fetchPendingOrders();

},

fetchPendingOrders() {

wx.request({

url: '/api/purchase\_orders/pending',

method: 'GET',

success: (res) => {

this.setData({ orders: res.data });

}

});

},

approveOrder(orderId) {

wx.request({

url: `/api/purchase\_orders/${orderId}/approve`,

method: 'POST',

success: () => {

this.fetchPendingOrders();

}

});

},

rejectOrder(orderId) {

wx.request({

url: `/api/purchase\_orders/${orderId}/reject`,

method: 'POST',

success: () => {

this.fetchPendingOrders();

}

});

}

});

**2.5 统计分析**

* **统计页面**：统计采购数据、库存数据以及各供应商的采购量。

// statistics.js

Page({

data: {

statistics: {}

},

onLoad() {

this.fetchStatistics();

},

fetchStatistics() {

wx.request({

url: '/api/statistics',

method: 'GET',

success: (res) => {

this.setData({ statistics: res.data });

}

});

}

});

**2.6 验收单**

* **验收单页面**：生成并查看验收单，确认商品已接收并入库。

// receipts.js

Page({

data: {

receipts: []

},

onLoad() {

this.fetchReceipts();

},

fetchReceipts() {

wx.request({

url: '/api/receipts',

method: 'GET',

success: (res) => {

this.setData({ receipts: res.data });

}

});

},

createReceipt(receiptData) {

wx.request({

url: '/api/receipts',

method: 'POST',

data: receiptData,

success: () => {

this.fetchReceipts();

}

});

}

});

**2.7 付款凭证**

* **付款凭证页面**：为采购订单生成付款凭证。

// paymentVoucher.js

Page({

data: {

vouchers: []

},

onLoad() {

this.fetchVouchers();

},

fetchVouchers() {

wx.request({

url: '/api/payment\_vouchers',

method: 'GET',

success: (res) => {

this.setData({ vouchers: res.data });

}

});

},

createVoucher(voucherData) {

wx.request({

url: '/api/payment\_vouchers',

method: 'POST',

data: voucherData,

success: () => {

this.fetchVouchers();

}

});

}

});

**2.8 打印单据**

* **打印页面**：支持打印采购订单、验收单、付款凭证等。

// print.js

Page({

printOrder(orderId) {

wx.request({

url: `/api/print/purchase\_order/${orderId}`,

method: 'GET',

success: (res) => {

wx.showToast({ title: '打印成功' });

}

});

},

printReceipt(receiptId) {

wx.request({

url: `/api/print/receipt/${receiptId}`,

method: 'GET',

success: (res) => {

wx.showToast({ title: '打印成功' });

}

});

}

});

3. 后端实现（在使用网页架构搭建一个 **采购系统和库存管理系统** 时，我们可以选择使用常见的 **前端框架（如 React, Vue, Angular）** 搭建访问端，使用 **后端框架（如 Node.js + Express）** 搭建服务端。以下是前后端的整体实现方案。

**1. 系统架构概述**

* **前端 (Client)**：基于 React (或 Vue, Angular) 构建用户界面，负责处理用户交互和发送 API 请求。
* **后端 (Server)**：基于 Node.js + Express 构建 RESTful API，处理数据库操作和业务逻辑。
* **数据库 (Database)**：使用 MySQL, PostgreSQL 或 MongoDB 管理数据。

**2. 具体功能模块**

前后端都会分别实现以下功能：

1. 组织架构管理
2. 商品维护
3. 采购录入
4. 会签审批
5. 统计分析
6. 验收单
7. 付款凭证
8. 打印单据

）

**3. 前端实现 (React 版)**

**3.1 项目结构**

/client

/src

/components

Department.js

Item.js

PurchaseOrder.js

Approval.js

Statistics.js

Receipt.js

PaymentVoucher.js

Print.js

/services

api.js

App.js

index.js

3.2 安装依赖

npx create-react-app client

cd client

npm install axios react-router-dom

3.3 services/api.js（封装 API 调用）

import axios from 'axios';

const API\_BASE\_URL = 'http://localhost:3000/api';

export const getDepartments = () => axios.get(`${API\_BASE\_URL}/departments`);

export const getItems = () => axios.get(`${API\_BASE\_URL}/items`);

export const createPurchaseOrder = (order) => axios.post(`${API\_BASE\_URL}/purchase\_orders`, order);

export const getPendingOrders = () => axios.get(`${API\_BASE\_URL}/purchase\_orders/pending`);

export const approveOrder = (id) => axios.post(`${API\_BASE\_URL}/purchase\_orders/${id}/approve`);

export const getStatistics = () => axios.get(`${API\_BASE\_URL}/statistics`);

export const createReceipt = (receipt) => axios.post(`${API\_BASE\_URL}/receipts`, receipt);

export const createPaymentVoucher = (voucher) => axios.post(`${API\_BASE\_URL}/payment\_vouchers`, voucher);

export const printOrder = (id) => axios.get(`${API\_BASE\_URL}/print/purchase\_order/${id}`);

export const printReceipt = (id) => axios.get(`${API\_BASE\_URL}/print/receipt/${id}`);

3.4 components/Department.js（组织架构管理页面）

import React, { useState, useEffect } from 'react';

import { getDepartments } from '../services/api';

const Department = () => {

const [departments, setDepartments] = useState([]);

useEffect(() => {

getDepartments().then(response => setDepartments(response.data));

}, []);

return (

<div>

<h2>组织架构</h2>

<ul>

{departments.map(dept => (

<li key={dept.id}>{dept.name}</li>

))}

</ul>

</div>

);

};

export default Department;

3.5 components/Item.js（商品维护页面）

import React, { useState, useEffect } from 'react';

import { getItems } from '../services/api';

const Item = () => {

const [items, setItems] = useState([]);

useEffect(() => {

getItems().then(response => setItems(response.data));

}, []);

return (

<div>

<h2>商品维护</h2>

<ul>

{items.map(item => (

<li key={item.id}>{item.name} - 库存: {item.quantity}</li>

))}

</ul>

</div>

);

};

export default Item;

3.6 components/PurchaseOrder.js（采购录入页面）

import React, { useState } from 'react';

import { createPurchaseOrder } from '../services/api';

const PurchaseOrder = () => {

const [order, setOrder] = useState({ items: [] });

const handleSubmit = (e) => {

e.preventDefault();

createPurchaseOrder(order).then(() => alert('采购订单已创建'));

};

return (

<div>

<h2>采购录入</h2>

<form onSubmit={handleSubmit}>

<input

type="text"

placeholder="商品名称"

onChange={(e) => setOrder({ ...order, name: e.target.value })}

/>

<button type="submit">提交</button>

</form>

</div>

);

};

export default PurchaseOrder;

3.7 components/Approval.js（会签审批页面）

import React, { useState, useEffect } from 'react';

import { getPendingOrders, approveOrder } from '../services/api';

const Approval = () => {

const [orders, setOrders] = useState([]);

useEffect(() => {

getPendingOrders().then(response => setOrders(response.data));

}, []);

const handleApprove = (id) => {

approveOrder(id).then(() => alert('订单已批准'));

};

return (

<div>

<h2>会签审批</h2>

<ul>

{orders.map(order => (

<li key={order.id}>

{order.name}

<button onClick={() => handleApprove(order.id)}>批准</button>

</li>

))}

</ul>

</div>

);

};

export default Approval;

3.8 App.js（主页面路由）

import React from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Department from './components/Department';

import Item from './components/Item';

import PurchaseOrder from './components/PurchaseOrder';

import Approval from './components/Approval';

const App = () => {

return (

<Router>

<Switch>

<Route path="/departments" component={Department} />

<Route path="/items" component={Item} />

<Route path="/purchase-orders" component={PurchaseOrder} />

<Route path="/approvals" component={Approval} />

</Switch>

</Router>

);

};

export default App;

**4. 后端实现 (Node.js + Express)**

**4.1 项目结构**

/server

/routes

departments.js

items.js

purchaseOrders.js

approvals.js

statistics.js

receipts.js

paymentVouchers.js

print.js

app.js

4.2 安装依赖

npm init -y

npm install express body-parser cors

4.3 app.js（后端主入口）

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express();

app.use(cors());

app.use(bodyParser.json());

const departmentsRoutes = require('./routes/departments');

const itemsRoutes = require('./routes/items');

const purchaseOrdersRoutes = require('./routes/purchaseOrders');

const approvalsRoutes = require('./routes/approvals');

const statisticsRoutes = require('./routes/statistics');

const receiptsRoutes = require('./routes/receipts');

const paymentVouchersRoutes = require('./routes/paymentVouchers');

const printRoutes = require('./routes/print');

app.use('/api/departments', departmentsRoutes);

app.use('/api/items', itemsRoutes);

app.use('/api/purchase\_orders', purchaseOrdersRoutes);

app.use('/api/approvals', approvalsRoutes);

app.use('/api/statistics', statisticsRoutes);

app.use('/api/receipts', receiptsRoutes);

app.use('/api/payment\_vouchers', paymentVouchersRoutes);

app.use('/api/print', printRoutes);

const port = process.env.PORT || 3000;

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});

4.4 routes/departments.js

const express = require('express');

const router = express.Router();

let departments = [

{ id: 1, name: '采购部' },

{ id: 2, name: '财务部' }

];

router.get('/', (req, res) => {

res.json(departments);

});

module.exports = router;

4.5 routes/items.js

const express = require('express');

const router = express.Router();

let items = [

{ id: 1, name: '硬盘', quantity: 100 },

{ id: 2, name: '内存条', quantity: 50 }

];

router.get('/', (req, res) => {

res.json(items);

});

module.exports = router;

**4.6 其他路由文件**

* purchaseOrders.js：处理采购订单的录入和管理
* approvals.js：处理审批功能
* statistics.js：处理统计数据
* receipts.js：处理验收单
* paymentVouchers.js：处理付款凭证
* print.js：处理打印功能

**5. 启动前后端**

**5.1 启动前端**

cd client

npm start

5.2 启动后端

cd server

node app.js

**6. 改进建议**

**a.** 使用 JWT 实现用户认证与权限控制，确保各个模块的安全性。  
**b.** 为后端接口添加单元测试，确保功能的稳定性。

**部署文档：采购系统与库存管理系统**

在此文档中，我们将详细介绍如何打包、部署和运行整个采购系统和库存管理系统项目。项目分为 **前端 (React)** 和 **后端 (Node.js + Express)** 两部分，系统架构采用前后端分离模式。

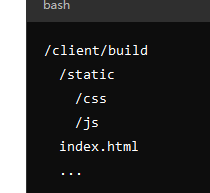
**1. 打包与安装包准备**

**1.1 前端打包 (React)**

**1.1.1 打包步骤**

1. 打开终端，进入 **client** 文件夹：

bash

1. 安装所需依赖包：npm install
2. 运行以下命令来打包前端项目（React 项目）：npm run build
3. 生成的打包文件会保存在 client/build 目录下。这个目录将包含所有需要部署的静态文件。/client/build /static /css /js index.html ... 

**1.2 后端打包 (Node.js + Express)**

**1.2.1 打包步骤**

1. **打开终端，进入 server 文件夹：**

cd server

2 安装后端依赖包：npm install

3确保 server 文件夹下有以下内容：/server

/routes

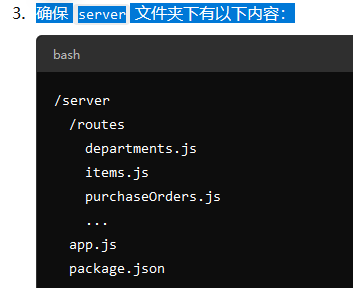
departments.js

items.js

purchaseOrders.js

...

app.js

package.json

4 将整个 **server** 文件夹打包为 zip 文件或复制到服务器上。

**2. 部署步骤**

**2.1 环境要求**

**2.1.1 前端 (静态文件)**

* Web 服务器（如 Nginx 或 Apache）
* 前端文件部署在 Web 服务器的根目录

**2.1.2 后端 (Node.js)**

* Node.js 环境 (Node.js v14+)
* 数据库：MySQL, PostgreSQL, 或 MongoDB
* 可选：进程管理工具 (如 PM2)

**2.2 部署前端（React）**

**2.2.1 使用 Nginx 部署**

1. 安装 Nginx（如果没有安装）：

sudo apt update

sudo apt install nginx

2将 client/build 文件夹中的所有内容上传到服务器，放置到 /var/www/html 目录下：

bash

sudo cp -r client/build/\* /var/www/html/

3配置 Nginx。打开配置文件：sudo nano /etc/nginx/sites-available/default

修改 Nginx 配置如下：server {

listen 80;

server\_name your\_domain\_or\_IP;

root /var/www/html;

index index.html;

location / {

try\_files $uri $uri/ /index.html;

}

}

4测试并重新加载 Nginx：

sudo nginx -t

sudo systemctl restart nginx

**2.2.2 使用 Apache 部署**

1. 安装 Apache：

sudo apt update

sudo apt install apache2

2将 client/build 文件夹中的所有内容复制到 /var/www/html 目录：

sudo cp -r client/build/\* /var/www/html/

3设置 Apache 的 .htaccess 文件（创建在 client/build 中的 .htaccess）：

<IfModule mod\_rewrite.c>

RewriteEngine On

RewriteBase /

RewriteRule ^index\.html$ - [L]

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule . /index.html [L]

</IfModule>

4重启 Apache：sudo systemctl restart apache2

**2.3 部署后端（Node.js + Express）**

**2.3.1 上传后端代码**

1. 将整个 server 文件夹通过 FTP、SCP 等工具上传至服务器。
2. 进入服务器上的 server 文件夹：

cd /path/to/server

**2.3.2 配置环境**

1. 如果项目中使用了环境变量（如数据库连接），请在 server 文件夹下创建 .env 文件：

DB\_HOST=localhost

DB\_USER=root

DB\_PASS=yourpassword

DB\_NAME=yourdbname

2安装依赖：npm install

**2.3.3 启动服务**

1. 直接启动应用：

node app.js

2或使用 PM2 进行进程管理：

npm install pm2 -g

pm2 start app.js

3 服务器将监听端口 3000，你可以通过 http://your-server-ip:3000 访问后端 API。

**2.3.4 设置反向代理（Nginx）**

为了让前端和后端共用同一个域名，可以设置 Nginx 反向代理。编辑 Nginx 配置文件：

bash

sudo nano /etc/nginx/sites-available/default

在配置文件中添加以下内容，代理后端的 /api 请求到 Node.js 服务器：

server {

listen 80;

server\_name your\_domain\_or\_IP;

root /var/www/html;

index index.html;

location / {

try\_files $uri $uri/ /index.html;

}

location /api/ {

proxy\_pass http://localhost:3000/;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

保存并退出后，重启 Nginx：

**3. 数据库配置与初始化**

1. 登录数据库服务器，创建数据库：

mysql -u root -p

CREATE DATABASE procurement\_system;

2为应用创建相应的数据表。以下是一个示例 SQL，用于初始化数据库表：

CREATE TABLE departments (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

CREATE TABLE items (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

quantity INT NOT NULL

);

CREATE TABLE purchase\_orders (

id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

status VARCHAR(50),

total\_amount DECIMAL(10, 2),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE approval\_logs (

id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

employee\_id INT,

status VARCHAR(50),

comment TEXT,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE receipts (

id INT AUTO\_INCREMENT PRIMARY KEY,

purchase\_order\_id INT,

received\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status VARCHAR(50)

);

CREATE TABLE payment\_vouchers (

id INT AUTO\_INCREMENT PRIMARY KEY,

purchase\_order\_id INT,

amount\_paid DECIMAL(10, 2),

payment\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

3配置数据库连接：修改 server/.env 文件，填写数据库相关信息：

DB\_HOST=localhost

DB\_USER=root

DB\_PASS=yourpassword

DB\_NAME=procurement\_system

4**最终检查**

1. 确保前后端都已正确运行。
2. 前端应通过浏览器访问 http://your-server-domain，后端 API 应通过 http://your-server-domain/api 访问。
3. 确保数据库已经正确配置，并且表结构符合需求。

5**注意事项**

* 确保前端与后端的接口路径一致。
* 确保 Nginx 或 Apache 配置正确，静态文件与 API 请求不冲突。
* 定期备份数据库数据和项目源代码。

6**未来改进方向**

* **安全性**：为后端 API 添加用户认证（如 JWT）。
* **扩展性**：可以为该系统增加更多功能，如多级权限、供应商管理等。
* **监控与日志**：使用 PM2 或其他工具来监控服务状态，并记录日志