开发文档

起始开发框架

项目地址

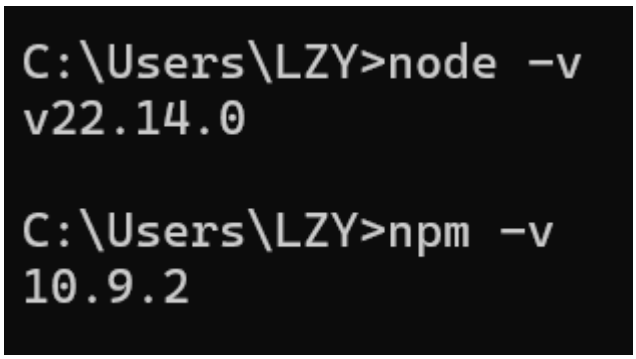
https://github.com/HollowLamp/copaint

开发工具

需要node.js

https://nodejs.org/zh-cn

直接下载安装，完成后如图验证



需要git

编辑器建议vscode

克隆

代码块

```
git clone https://github.com/HollowLamp/copaint.git
```

进文件夹运行

代码块

```
cd copaint

# 安装依赖
```

```
npm install

# 启动运行
npm run dev
```

现有框架解释

> 只是提个初始结构出来，可以随时提建议修改，代码基本直接让AI生成了

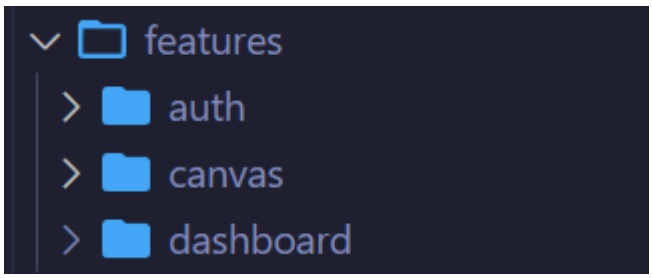
文件夹与内容解释

代码块

```
src/
├── assets/              # 静态资源（图片、字体）
├── components/          # 通用组件库（按钮等）
├── contexts/            # 状态管理
├── features/            # 各页面模块
├── hooks/               # 自定义Hooks（主题切换等）
├── router/              # 路由配置
├── services/            # 服务层
├── styles/              # 全局样式与CSS变量
├── utils/               # 工具方法集合
```

assets放静态内容，图片等

按设计图，需要写三个页面，在features下的三个文件夹内分别有初始的代码



contexts下是状态管理相关内容，管理应用的全局状态，可能也不会用上，先创建出来了。

写前端主要是将页面拆分成UI组件。有些组件在各个地方都会用到，可以提取到components文件夹下统一引用，后续修改时方便些。

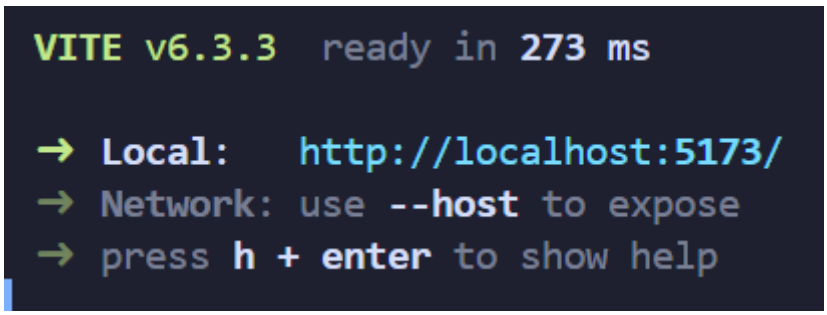
hooks下存放自定义hook，组件是对公共UI的封装，hook就是对公共逻辑的封装，现在有一个切换明暗主题的hook，后面解释样式的时候会细嗦。

router下管理路由，就是管理导航栏里不同的url地址会导向什么页面，后续也会管理页面重定向（比如未登录用户试图直接访问画板url的时候强制重定向到登录页面上）

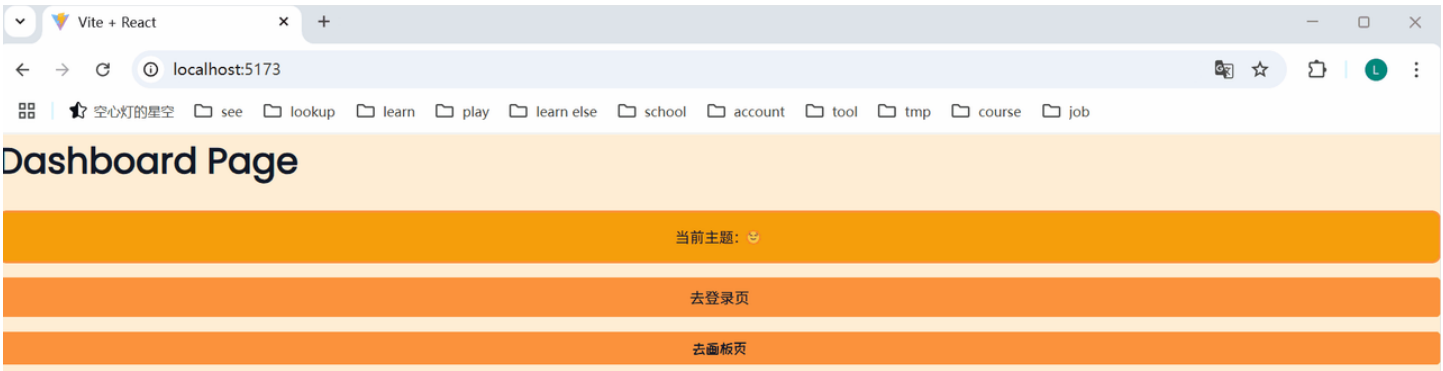
需要和firebase交互，相关逻辑准备安排写在services文件夹下

utils可以放一些工具函数

可以先按前面的指令跑起来项目，然后访问给的地址



现在页面是个空壳，给了个大的明暗按钮测试明暗主题，为了方便调试，在三个页面上都加了相互跳转的按钮，这些后续都是要按设计图和功能需求写的，比如未登录的时候只能访问到登录页，在dashboard上通过访问画板跳转到对应编号的canvas上这些。



样式

写样式时建议的写法

jsx后缀是UI组件，仅包含内容和逻辑，样式单独写在css后缀的文件里，为了防止各自创建css文件写样式导致相互冲突，请为自己负责写的jsx创建一个同名的xxx.module.css文件，然后在jsx文件里导入并使用，可以避免样式冲突。以dashboard页为例子

代码块

```css
.buttonGroup {
  margin-top: 20px;
  display: flex;
  flex-direction: column;
  gap: 12px;
}

.themeButton {
  background-color: light-dark(var(--orange-500), var(--gray-600));
  color: light-dark(var(--gray-900), var(--gray-50));
  border: 2px solid light-dark(var(--orange-400), var(--gray-700));
  border-radius: 8px;
  padding: 12px 24px;
}

.themeButton:hover {
  background-color: light-dark(var(--orange-600), var(--gray-500));
  transform: translateY(-2px);
}

```

代码块

```jsx
// DashboardPage.jsx
import { useTheme } from '../../hooks/ThemeContext';
import { useNavigate } from 'react-router';
import { Button } from '../../components/button/Button';
//这里从同目录下的CSS文件里导入样式，CSS里面写了.xxx，在标签上加styles.xxx即可
import styles from './DashboardPage.module.css';

export const Component = () => {
  const { theme, toggleTheme } = useTheme();
  const navigate = useNavigate();

  return (
    <div>
      // 一些简单样式也可以直接写在这里面，比如下一行和后面三个按钮的字体都用了不一样的
      <h1 style={{ fontFamily: 'Poppins' }}>Dashboard Page</h1>
      <div className={styles.buttonGroup}>
        <Button
          onClick={toggleTheme}
          style={{ fontFamily: 'AlibabaPuHuiTi' }}
          className={styles.themeButton}
        >
          当前主题: {theme === 'light' ? '🌞' : '🌙'}
        </Button>
```

```
24
25          <Button onClick={() => navigate('/login')} style={{ fontFamily:
    'SourceHanSansSC' }}>
26            去登录页
27          </Button>
28
29          <Button onClick={() => navigate('/canvas/1')} style={{ fontFamily:
    'ZCOOLKuaiLe' }}>
30            去画板页
31          </Button>
32        </div>
33      </div>
34    );
35  };
```

字体

字体使用可以参考组长的设计图，点上面的标注再点到文字上就可以看到是什么字体了。用什么字体不关键，关键是同一个区域同一级别的字体要一致。

https://modao.cc/proto/53W28kGxsvazge2knWRwgD/sharing?view_mode=read_only&screen=rbpUjR6Q3i5Yd2NR8

把设计图用到的四种特殊字体都下载并注册在CSS里面了，需要用什么字体直接参考styles文件夹下的fonts.css文件里面就行，font-face就是把对应字体文件注册了，用的时候直接用font-family写的名字即可，可以参考dashboard页的用法

主题

这里需要注意，因为是多人协作，不统一颜色的话体验会不太好

下面是预设的颜色主题，详见index.css

代码块

```
1   [data-theme='light'] {
2     --bg-primary: var(--orange-100); /* 主背景色（浅橙） */
3     --bg-secondary: var(--orange-50); /* 辅助背景色（更浅） */
4
5     --text-primary: var(--gray-900); /* 主要文字（深色） */
6     --text-secondary: var(--gray-600); /* 次要文字（灰色） */
7
8     --accent-primary: var(--orange-500); /* 主色调（按钮背景） */
9     --accent-hover: var(--orange-600); /* 按钮hover */
10    --accent-secondary: var(--orange-400); /* 边框色 */
11
12    --border-color: var(--orange-100); /* 边框颜色 */
13  }
```

```
14
15  [data-theme='dark'] {
16    --bg-primary: var(--gray-800); /* 主背景色（深灰） */
17    --bg-secondary: var(--gray-900); /* 辅助背景色（更深） */
18
19    --text-primary: var(--gray-50); /* 主要文字（浅色） */
20    --text-secondary: var(--gray-100); /* 次要文字（浅灰） */
21
22    --accent-primary: var(--gray-600); /* 主色调（按钮背景 深色） */
23    --accent-hover: var(--gray-500); /* 按钮hover */
24    --accent-secondary: var(--gray-700); /* 边框色 */
25
26    --border-color: var(--gray-700); /* 边框颜色 */
27  }
```

颜色都使用统一的CSS变量，一样在index.css里定义

代码块

```
1   :root {
2     /* 基础色板 */
3     --orange-50: #fff7ed;
4     --orange-100: #ffedd5;
5     --orange-400: #fb923c;
6     --orange-500: #f59e0b;
7     --orange-600: #fbbf24;
8     --orange-700: #f59e0b;
9
10    --gray-50: #f9fafb;
11    --gray-100: #d1d5db;
12    --gray-500: #6b7280;
13    --gray-600: #64748b;
14    --gray-700: #374151;
15    --gray-800: #1f2937;
16    --gray-900: #111827;
17  }
```

实际写的时候，可以像默认背景设定一样，直接用给的主题变量

代码块

```
1   body {
2     background-color: var(--bg-primary);
3     color: var(--text-primary);
4     transition:
5       background-color 0.3s ease,
```

```
      color 0.3s ease;
    font-family: 'ZCOOLKuaiLe', 'Poppins', 'AlibabaPuHuiTi', 'SourceHanSansSC', sans-serif;
  }
```

也可以使用预定义配置出来的CSS函数

代码块

```
    background-color: light-dark(var(--orange-400), var(--gray-700));
    color: light-dark(var(--gray-900), var(--gray-50));
    border: 1px solid light-dark(var(--orange-100), var(--gray-600));
```

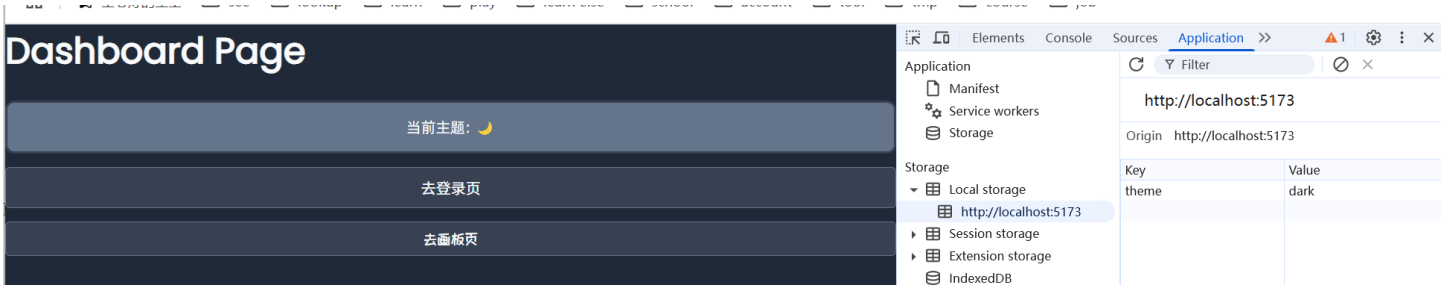
第一个参数是亮值，第二个是暗值，编译出来会是这样

```
[data-theme="light"] ._button_1i1rp_1 {
    background-color: var(--orange-400);
    color: var(--gray-900);
    border: 1px solid var(--orange-100);
}
```

```
[data-theme="dark"] ._button_1i1rp_1 {
    background-color: var(--gray-700);
    color: var(--gray-50);
    border: 1px solid var(--gray-600);
}
```

明暗切换原理，就是在全局变换dark和light，然后颜色值也会随之变换，内部通过React的状态管理结合localStorage实现，localStorage作用就是记住用户最后一次的选择，下次访问还是上次的主题

谷歌浏览器可以在这里看见



记住写颜色的时候选择上面两种之一即可，颜色值可以自行调整，主要是保持风格统一。

注意事项汇总和分工

分支管理与提交

参考GPT的分支管理方式

main分支是正式版

dev是开发版，主要变动的一个

dev-xxx，按姓名的开发分支

开发拉取流程

代码块

```
git checkout dev
git pull origin dev

// 后续再切到自己分支不用-b
git checkout -b dev-xxx
```

开发推送流程

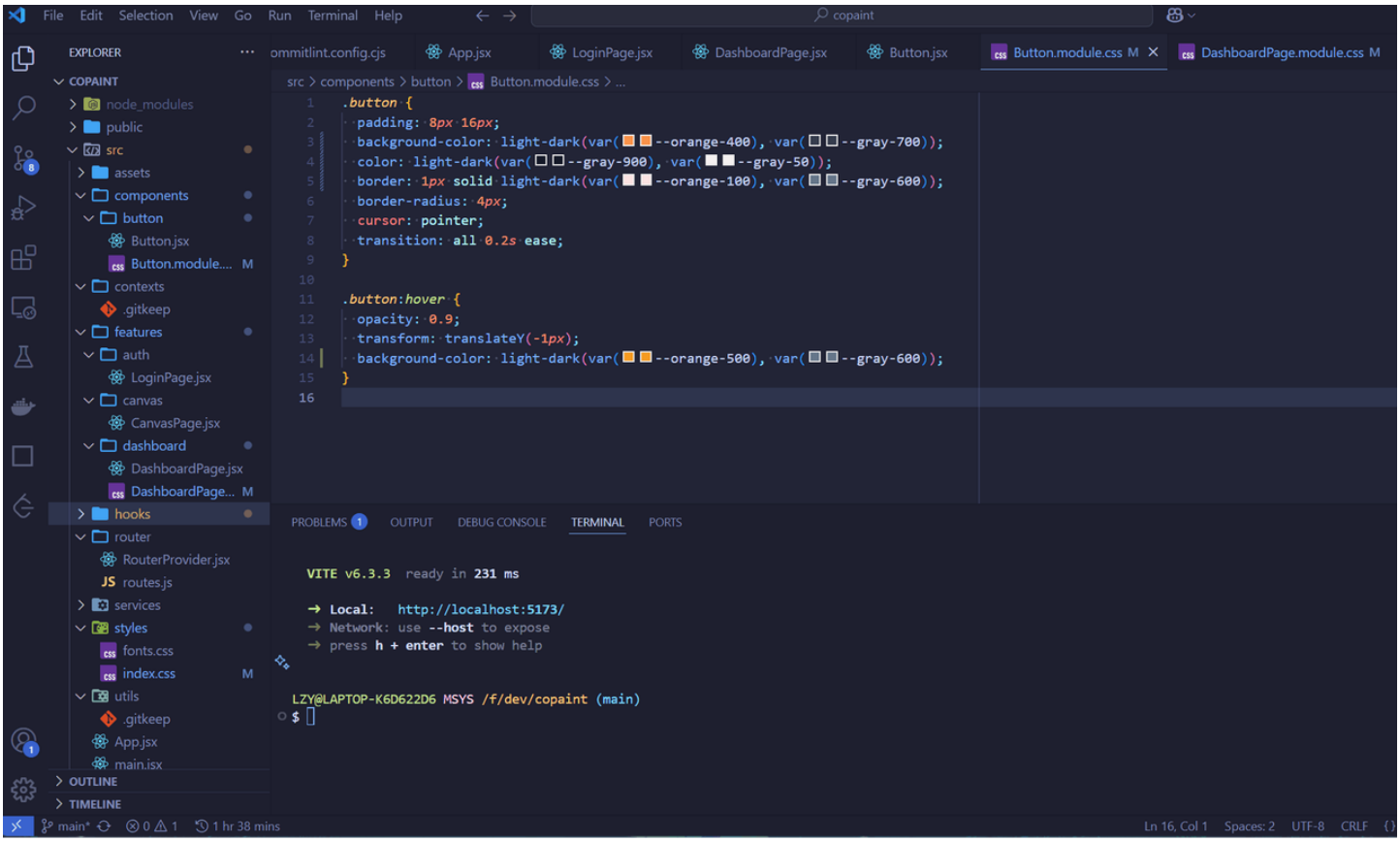
代码块

```
git checkout dev
git pull origin dev
git merge dev-xxx
git push origin dev

// 可以把自己的分支也推上去留痕
git push origin dev-xxx

// 本地后续可以切回自己的分支继续开发，然后重复上面的流程合并提交
git checkout dev-xxx
```

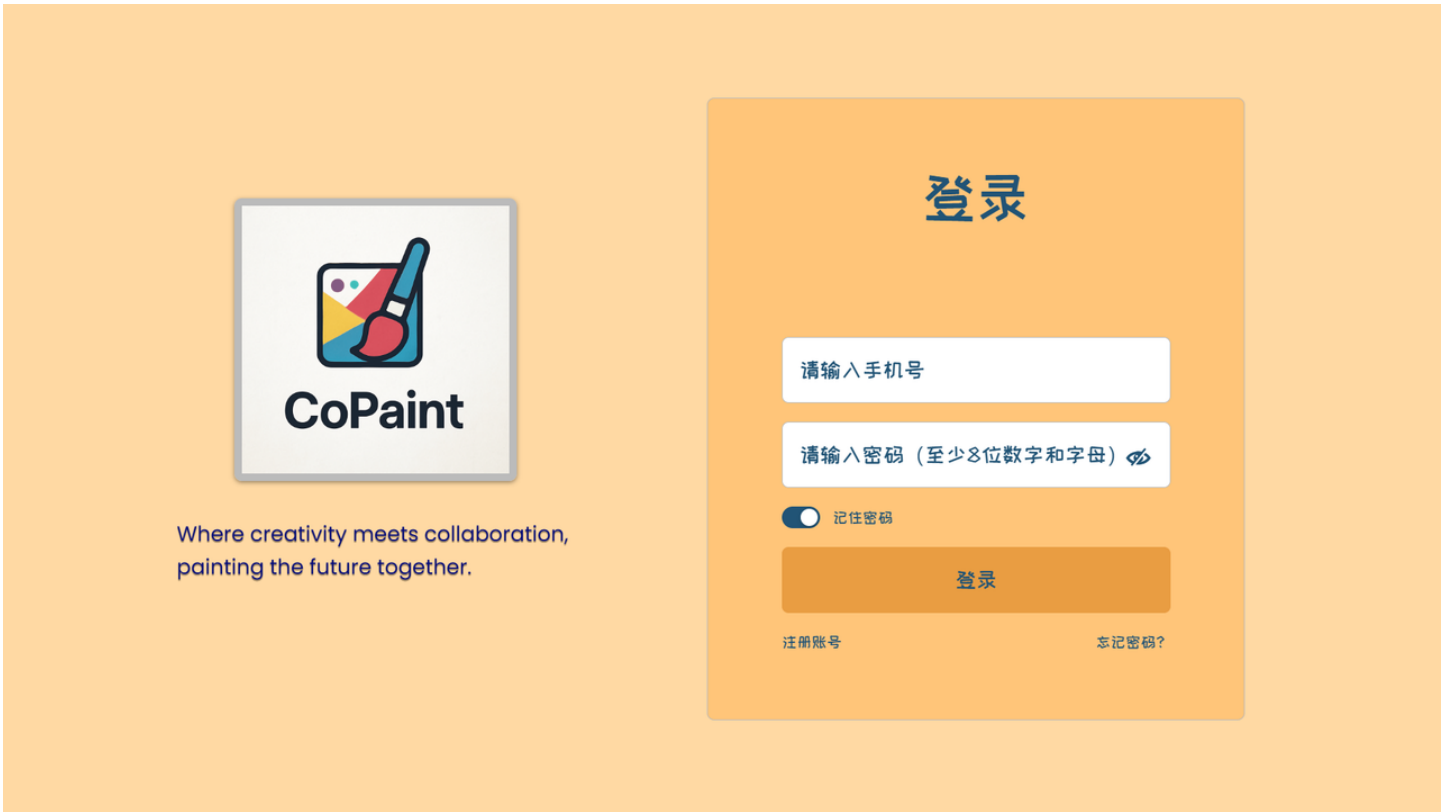
为了让提交信息好看些，方便展示，commit时建议采用配置好的脚本进行commit，流程如下



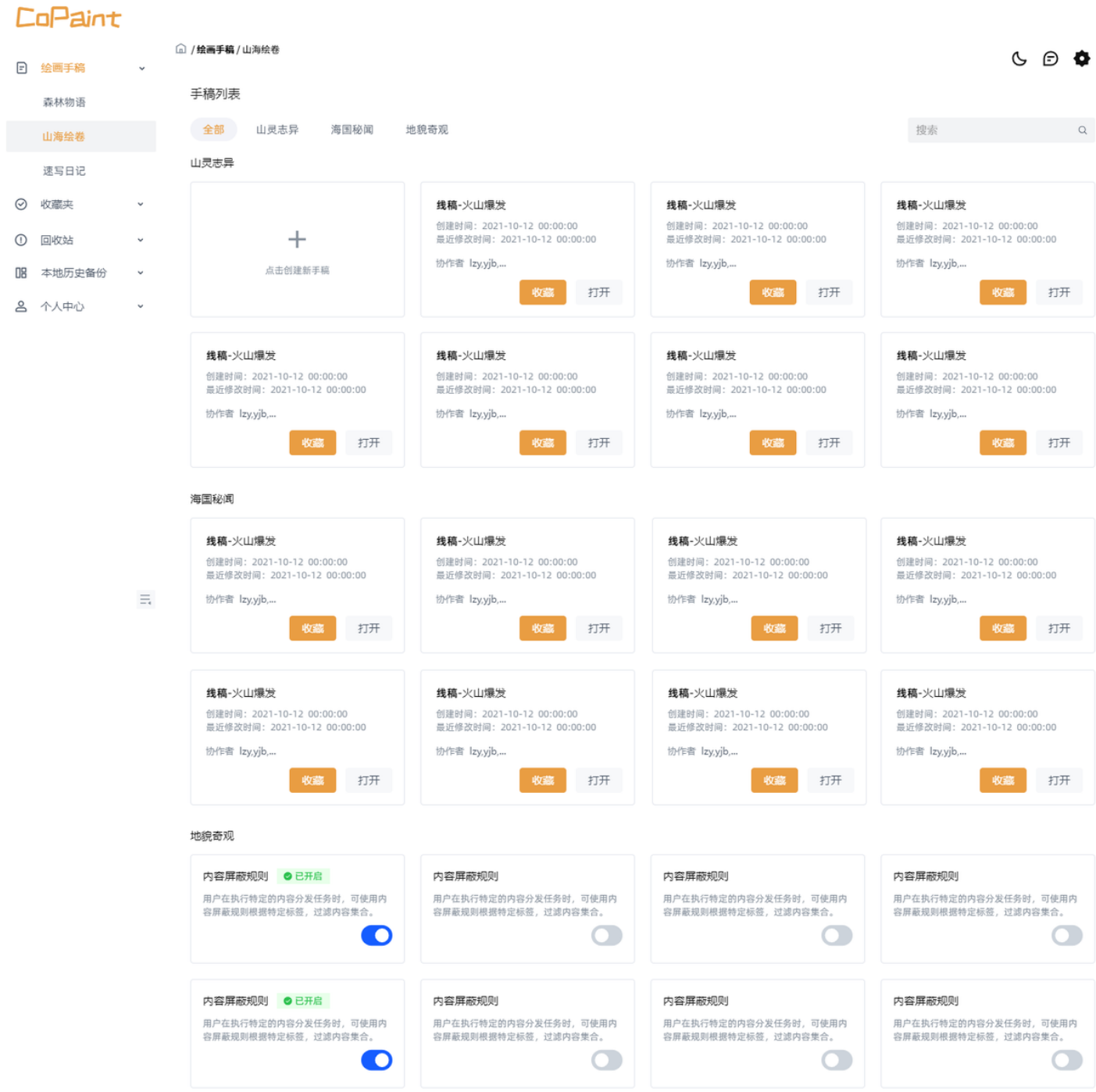
开发内容与分工

先把页面搭起来，暂且忽略页面间的跳转和身份认证和同步等功能

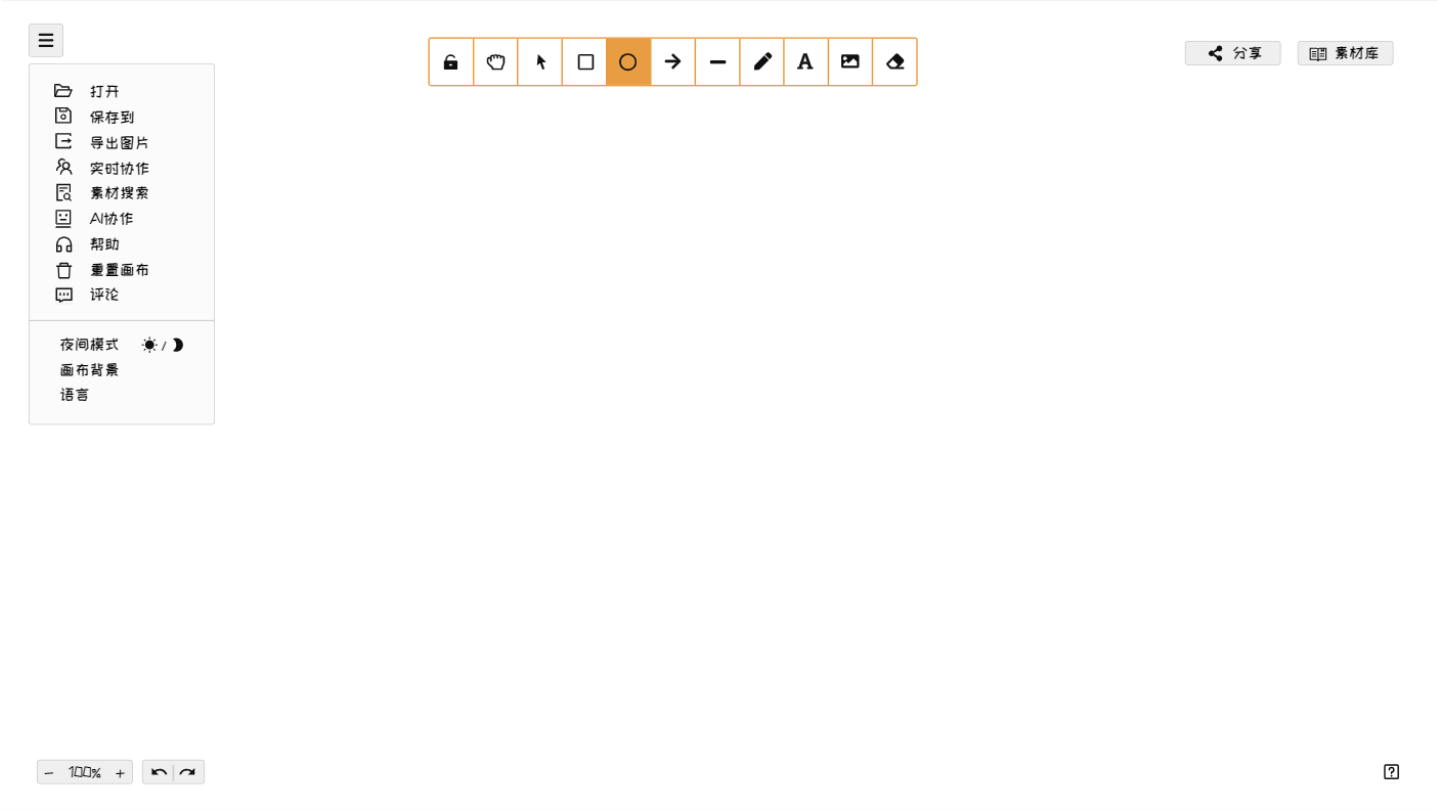
这个页面可以一个人写



这个页面分三个人


CoPaint
/ 绘画手稿 / 山海绘卷
绘画手稿
森林物语
山海绘卷
速写日记
收藏夹
回收站
本地历史备份
个人中心
手稿列表
全部
山灵志异
海国秘闻
地貌奇观
搜索
山灵志异
点击创建新手稿
线稿-火山爆发
创建时间：2021-10-12 00:00:00
最近修改时间：2021-10-12 00:00:00
协作者 lzy,yjb,...
收藏
打开
海国秘闻
地貌奇观
内容屏蔽规则
已开启
用户在执行特定的内容分发任务时，可使用内容屏蔽规则根据特定标签，过滤内容集合。

这个页面分一个人



然后需要一个人来处理firebase相关的内容

只是大概分个工，这个东西感觉确实没办法分得很清楚。

看板页会有很多内容要写，因为侧边栏有很多个选项，代表这个页面有很多个子页面，虽然设计图只画了一张，但收藏夹，回收站，个人中心这些都没画，所以实际任务量不小。所以分三个人。可以先把外围骨架搭建起来，然后考虑怎么分工写各个子页面核心部分的内容。

画板页只分一个人主要考虑的是能找到现有的或者第三方的实现，应该没有特别难。

登录相对简单，额外分配了通用性任务。

负责firebase需要随时和其他5人对接，提供服务接口。

写的时候只需要注意样式里面的颜色按前面说的来就行，因为用AI写代码它可能不会管颜色，别的没什么需要注意的，AI写的能跑就行

在下面填上名字

- 登录页，并且负责配置路由，写公共组件，工具函数等公用的部分
  - 栗智勇
- 看板页
  - 杨锦波
  - 王潘羽

    - xxx
- 画板页
    - 黄权
- firebase，服务端相关内容
    - xxx

反馈问题