####关于 Login.vue 和 Register.vue
位置：
handleLogin() 方法（<script setup> 内 27 - 47 行左右）
作用：
向后端 登录接口发送邮箱 + 密码；根据返回结果写入前端状态与 localStorage
调用方式：
axios.post("/api/login", { email, password })

---
1. 调用时机与流程
用户点击 “Login” 按钮 → 触发表单 @submit.prevent="handleLogin"

handleLogin() 执行：

前端做空字段校验。

使用 axios 发送 POST /api/login。

若成功：

读取后端 JSON 响应中的 token 与 user 信息。

localStorage.setItem("token", token) —— 后续请求可在 Axios 拦截器里自动带 Authorization: Bearer token。

通过 Pinia userStore.login(user) 把 isAuthenticated=true、user.name 等写入全局状态。

跳转到 /my-center。

若失败：根据后端返回 err.response.data.message 或通用文本更新 errorMsg，在表单下方显示。

前端 → 后端 请求格式：
POST /api/login
Content-Type: application/json

```
{
  "email": "jack@example.com",
  "password": "123456"
}
```

字段　类型　说明
email　string 用户注册邮箱，已去除首尾空格 trim()
password　string 明文或前端二次加密后的密码

后端 → 前端 成功响应预期
HTTP/1.1 200 OK
Content-Type: application/json

```
{
  "token": "<JWT-or-Session-Token>",
  "user": {
    "id": 42,
    "name": "Jack",
    "email": "jack@example.com"
  }
}
```

失败响应预期
HTTP/1.1 401 Unauthorized
Content-Type: application/json

```
{ "message": "Invalid email or password" }
```

后端接口契约：
POST /api/register

```
{
  "username": "Alice",
  "email": "alice@example.com",
  "password": "abc123def4"
}
```

######关于 MyCenter.vue

方法	HTTP 调用	何时触发	期望 请求	期望 成功响应	写入到组件的字段
fetchFavorites()	GET /api/user/favorites	组件 created() 时并行调度	-（只需鉴权 Authorization: Bearer <token>）	Array of objects [{ id, title,	favoriteBooks

方法	HTTP 调用	何时触发	期望 请求	期望 成功响应	写入到组件的字段
				author, cover }]	
fetchRewards()	GET /api/user/rewards	同上	-	Object { records: [{ id, amount, bookTitle, date }], balance: number }	rewards ← records coinBalance ← balance
fetchComments()	GET /api/user/comments	同上	-	Array of objects [{ id, bookTitle, content, date }]	comments
fetchCoins()	GET /api/user/coins *(兜底，若上一步已带 balance 可省略)*	同上	-	Object { balance: number }	coinBalance

---

调用流程 & 代码位置

```
created() {
  this.fetchAll();           // 行 38
},
methods: {
  async fetchAll() {         // 行 44
    await Promise.all([
      this.fetchFavorites(), // 49
      this.fetchRewards(),   // 50
      this.fetchComments(),  // 51
      this.fetchCoins(),     // 52
    ]);
    if (!this.favoriteBooks.length) this.useDemoData();
```

```
  },
  async fetchFavorites() {   // 60
    const { data } = await axios.get("/api/user/favorites");
    this.favoriteBooks = data;
  },
  ... （其余三个 fetch 方法同理）
}
```

典型请求示例

```
GET /api/user/favorites HTTP/1.1
Host: api.novel-heaven.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9…

// 无 QueryString，服务器依据 token 找到当前用户
```

典型响应示例

```
// /api/user/rewards  成功 200
{
  "records": [
    { "id": 1, "amount": 50, "bookTitle": "Martial Universe", "date":
"2025-04-10" },
    { "id": 2, "amount": 20, "bookTitle": "Perfect World",
"date": "2025-04-12" }
  ],
  "balance": 77
}
```

组件如何消费返回值

1. 数据落地

```
this.rewards = data.records;    // v-for 渲染 <li>
this.coinBalance = data.balance;// “My remaining coins:” 行
```

2. 界面渲染

favoriteBooks → 4×grid 卡片（封面 + 书名 + 作者）

rewards → “On DATE, gave X gold coins to "BOOK"”

comments → “In "BOOK", said "TEXT" (DATE)”

coinBalance → 余额行

3. 失败回退

```
catch { this.useDemoData(); }
```

useDemoData() 只调用一次（由 demoUsed 标记），确保不会无限追加导致「无数卡片 / 空行」。

后端侧最小接口契约:

Endpoint | 必须字段 | 说明

GET /api/user/favorites | Authorization header | 返回收藏列表

GET /api/user/rewards | 同上 | 同时返回余额

GET /api/user/comments | 同上 | 返回评论记录

GET /api/user/coins | 同上 | （可选兜底）返回余额

所有接口统一 200 OK + application/json。错误时返回

{ "message": "Unauthorized" }

并置 HTTP 401 方便前端跳登录页。

#####关于 AuthorDashboard.vue

#	调用位置 & 方法	触发时机	HTTP 请求	前端期望的成功响应	写入到组件的字段 / 后续动作
	openAuthorDialog() 行≈63	用户第一次点击 **Set Author Name** 并通过 prompt 输入合法名称（≤7）	POST /api/author/name Headers: Authorization: Bearer <token> Body (JSON): { "authorName": "MyName" }	200 OK → { "authorName": "MyName" } 或 { success:true }	① 设置 authorName ② authorNameConfirmed = true（按钮、My Center 功能解除禁用）
	refreshWorks() 行≈83	*a)* 组件 mounted 时你可手动调用；*b)* 当 **Create Work** 子组件 @creat	GET /api/author/works	200 OK → **Array:** [{ id, title, cover, synopsis, status, likes, commentsCount }, …]	覆盖 this.works = data，列表 UI 用 v-for 渲染

#	调用位置 & 方法	触发时机	HTTP 请求	前端期望的成功响应	写入到组件的字段 / 后续动作
3	editChapters(workId) 行≈75	ed 事件触发 用户在作品卡片点 **Edit Chapters**	——（只是路由跳转）	—	this.$router.push({ name:'ChapterEditor', params:{workId} })
4	子组件 **CreateWork.vue** 里的调用（父组件只负责打开弹窗）	点击“Publish”	POST /api/author/works Headers: Content-Type: multipart/form-data Body: FormData 包含 title, synopsis, cover 文件	201 Created → { workId: 123, message:"OK" }	弹窗内部 this.$emit("created") → 父组件 refreshWorks() 再抓最新列表

调用流程示意:

```
sequenceDiagram
  participant UI as AuthorDashboard
  participant API as Backend API

  UI->>UI: User clicks "Set Author Name"
  UI->>API: POST /api/author/name {authorName}
  API-->>UI: 200 OK
  UI->>UI: authorNameConfirmed = true

  UI->>UI: User clicks "Create New Work"
  UI-->>UI: show <CreateWork> modal
  UI->>API: POST /api/author/works (FormData)
  API-->>UI: 201 Created
  UI->>UI: emit("created") → refreshWorks()
  UI->>API: GET /api/author/works
```

```
  API-->>UI: [ ...works list... ]
  UI->>UI: this.works = list
--------
```

接口字段约定:

接口	字段	类型	说明
POST /api/author/name	authorName	string	Author name ≤ 7 chars
GET /api/author/works	—	—	JWT 中的 userId 用于查询
Work 对象	id	number	PK
	title	string	书名
	cover	string(URL)	封面 CDN / OSS 地址
	synopsis	string	简介
	status	"Ongoing" | "Finished"	连载状态
	likes	number	点赞数
	commentsCount	number	评论数
POST /api/author/works	FormData 字段: title string ≤10 synopsis string ≤300 cover file(jpg/png ≤2 MB)	—	新建作品

########关于 CreateWork.vue

该文件只有 一次真正的 HTTP 调用——当作者点击 Publish 按钮且所有表单校验通过时。其余逻辑都是前端本地校验、预览与弹窗控制。

#调用位置	触发时机	HTTP 请求	前端期望的 成功响应	写入或后续动作
handleSubmit() 方法（≈41 – 78 行）	点击 **Publish** → 表单校验通过	http POST /api/author/works Content-Type: multipart/form-data FormData 字段: • **title** (string	json { "workId": 123, "message": "Work published successfully."	*on success* → 1. this.$emit("created") 告知父组件刷新作品列表; 2. alert("Work

#调用位置	触发时机	HTTP 请求	前端期望的 成功响应	写入或后续动作
		≤10) • **synopsis** (string ≤300) • **cover** (file jpg/png ≤ 2 MB)	} 或 201 Created	published successfully!") 3. resetAndClose() 清空表单 & 关闭弹窗
Axios 错2误分支 (catch)	后端返回 4xx/5xx、网络超时等	—	json { "message": "Reason of failure..." }	设置 errorMessage → 页面显示红字 Failed to publish work.（若后端无 message 字段）

代码关键段（摘自 handleSubmit）：

```
// 构建 FormData
const formData = new FormData();
formData.append("title", this.title);
formData.append("synopsis", this.synopsis);
formData.append("cover", this.coverFile);

// 调用后端
await axios.post("/api/author/works", formData, {
  headers: { "Content-Type": "multipart/form-data" },
});
```

客户端校验顺序

1. **空值检查**：title / synopsis 不可为空
2. **长度限制**：title ≤ 10，synopsis ≤ 300
3. **文件类型**：仅 image/png、image/jpeg
4. **文件大小**：≤ 2 MB

若任何一步失败，前端直接在 errorMessage 中提示，不会向后端发送请求。

后端端点契约

字段	类型	约束	说明
title	string	≤ 10 UTF-8 chars	作品标题

字段	类型	约束	说明
synopsis	string	≤ 300 chars	简介 \
cover	file	jpg / png ≤ 2 MB	封面文件

与父组件（AuthorDashboard.vue）的配合

- 父组件在 <CreateWork v-model:visible="showCreateModal" @created="refreshWorks" />
- 当 CreateWork.vue 成功发布后 emit("created") → 父组件调用 GET /api/author/works 刷新作品列表，以看到新作品即时出现。

######关于 ChapterEditor.vue

#	调用方法（行号≈）	触发时机	HTTP 请求	成功响应 (示例)	前端后续动作
1	updateWorkStatus() (~115)	作者在页面顶部 **Status** 下拉框切换为 *Ongoing* / *Finished*	PUT /api/author/works/{workId}/status Headers: Authorization: Bearer <token> Body: { "status": "Ongoing" }	200 OK { message:"updated" } (或 204)	仅弹窗 alert 提示（当前实现）；可扩展为 Toast
2	publishChapter() (~130)	点击 **Publish Chapter** 按钮，且标题/内容符合校验	POST /api/author/works/{workId}/chapters JSON: { "title": "Chap 1", "content": "..." }	201 Created { chapterId: 7 }	将章节推入 chapters 列表；清空表单
3	saveDraft() (~155)	点击 **Save Draft**	POST /api/author/works/{workId}/drafts 同上 payload	201 Created { draftId: 12 }	将条目 (title) (draft) 推入 chapters 列表；保

#	调用方法（行号≈）	触发时机	HTTP 请求	成功响应 (示例)	前端后续动作
4	continueWithAI() (~172)	点击 AI Continued Writing 按钮	POST /api/llm/continue JSON: { "workTitle": "...", "chapterTitle": "...", "chapterContent": "..." }	200 OK { continuedText: "AI-generated ..." }	把 continuedText 直接追加到 textarea 留后台草稿

若任一调用 catch 抛错（4xx/5xx/网络）：

• 控制台打印 console.warn("API failed – using local add (dev mode)")

• 发布 / 草稿 保存会改用本地模拟以保证 UI 可测试；AI 续写追加占位文本 “(AI generated continuation...)”。

每个接口的字段约定

1. 修改作品状态:

```
PUT /api/author/works/{workId}/status
Authorization: Bearer <JWT>
Content-Type: application/json

{ "status": "Ongoing" }   // or "Finished"
```

成功 → HTTP 204 No Content 或:

```
{ "message": "Status updated" }
```

2. 发布章节:

```
POST /api/author/works/{workId}/chapters
Authorization: Bearer <JWT>
Content-Type: application/json

{
  "title":  "Chapter 3",
  "content": "Long chapter body …"
}
```

成功 → 201 Created

```
{ "chapterId": 123 }
```

3. 保存草稿

同路径但 .../drafts，payload 与章节相同；返回

```
{ "draftId": 55 }
```

4. AI 续写
POST /api/llm/continue
Authorization: Bearer <JWT>
Content-Type: application/json

```
{
  "workTitle":      "Moonlit Tales",
  "chapterTitle":   "Chapter 3",
  "chapterContent": "Text before invoking AI ..."
}
```

```
{ "continuedText": "AI generated additional paragraphs ..." }
```

依赖的全局状态

- workId 来自路由 param：/author-dashboard/chapter-editor?workId=123 或 params:{workId}。
- Authorization header 建议在 Axios 请求拦截器中统一注入：

```
axios.interceptors.request.use(cfg=>{
  const t = localStorage.getItem("token");
  if (t) cfg.headers.Authorization = `Bearer ${t}`;
  return cfg;
});
```

Home.vue

fetchHotBooks(limit = 27)

作用

- 向后端接口拉取热门小说列表，用于首页 <RankingCarousel> 组件展示。

调用时机

- 组件挂载时由 onMounted() 自动触发。

HTTP 请求

bash
复制编辑
GET /api/books/hot?limit={limit}
Host: <后端域名>

请求参数

参数	类型	必填	说明
limit	Number	否	要获取的小说数量，默认 27

成功响应示例

- 状态码：200 OK
- Content-Type：application/json

```
[
  {
    "id": 1,
    "rank": "01",
    "title": "惜花芷",
    "category": "宫斗",
    "cover": "src/assets/covers/book1.jpeg"
  },
  {
    "id": 2,
    "rank": "02",
    "title": "我不是戏神",
    "category": "都市",
    "cover": "src/assets/covers/book2.jpeg"
  },
  ...
]
```

失败响应示例

- 状态码：4xx / 5xx

- 响应体：

```
{ "message": "错误描述" }
```

前端处理

1. books.value = data;
2. <RankingCarousel :books="books" /> 自动更新渲染。

NovelDetail.vue

loadNovelDetail()

作用

- 从后端获取该小说的基本信息（标题、作者、封面、简介、更新时间等），用于页面顶部展示。

调用时机

- 组件挂载时由 onMounted() 自动调用一次。

HTTP 请求

```
GET /api/novel/{novelId}
Host: <后端域名>
Authorization: Bearer <token>   // 如需鉴权
```

成功响应示例

- 状态码：200 OK

```
{
  "id": 123,
  "title": "惜花芷",
  "author": "空留",
  "category": "宫斗",
  "updateTime": "2025-04-15 11:45",
  "cover": "/assets/covers/book1.jpeg",
  "description": "双生妹妹嫁入皇宫前夕…"
}
```

失败响应示例

```
{ "message": "小说不存在或服务器错误" }
```

前端处理

1. novel.value = data;
2. 捕获错误后 console.error，可加 UI 提示或占位数据。

---

loadChaptersList()

作用

- 获取该小说的章节列表，用于“目录”版块渲染。

调用时机

- 与 loadNovelDetail() 并行，在 onMounted() 中调用。

HTTP 请求

```
GET /api/novel/{novelId}/chapters
Host: <后端域名>
Authorization: Bearer <token>
```

成功响应示例

- 状态码：200 OK

```
[
  { "id": 1, "title": "第一章 替嫁" },
  { "id": 2, "title": "第二章 拿惯银枪的手" },
  …
]
```

失败响应示例

```
{ "message": "获取章节列表失败" }
```

前端处理

1. chaptersList.value = data;
2. 目录由 <router-link> + v-for 自动渲染。

---

fetchComments()

作用

- 拉取该小说的所有评论，用于“评论区”展示。

调用时机

- 组件挂载时 onMounted() 调用一次；
- 用户提交新评论后再次调用以刷新列表。

HTTP 请求

```
GET /api/novel/{novelId}/comments
Host: <后端域名>
Authorization: Bearer <token>
```

成功响应示例

- 状态码：200 OK

```
[
  {
    "id": 1001,
    "author": "Alice",
    "content": "太好看了！",
    "createdAt": "2025-04-26T10:30:00Z",
    "likes": 5
  },
  …
]
```

失败响应示例

```
{ "message": "加载评论失败" }
```

前端处理

1. comments.value = data;
2. 通过计算属性 sortedComments 排序并渲染。

---

submitComment()

作用

- 向后端提交用户新评论，提交成功后刷新评论列表。

触发方式

- 用户点击“发表评论”按钮。

HTTP 请求

```
POST /api/novel/{novelId}/comments
Host: <后端域名>
Authorization: Bearer <token>
Content-Type: application/json

{
  "content": "写下你的评论…"
}
```

成功响应示例

- 状态码：201 Created

```
{
  "id": 1007,
  "author": "当前用户",
  "content": "写下你的评论…",
  "createdAt": "2025-04-27T13:00:00Z",
  "likes": 0
}
```

失败响应示例

```
{ "message": "发布失败，请重试" }
```

前端处理

1. newComment.value = '';
2. 调用 fetchComments() 刷新列表；
3. 捕获错误后 console.error 并可提示用户。

---

toggleFavorite()

作用

- 切换用户对该小说的“收藏”状态，并同步到后端。

触发方式

- 用户点击“收藏”按钮时调用。

HTTP 请求

```
POST /api/novel/{novelId}/favorite
Host: <后端域名>
Authorization: Bearer <token>
Content-Type: application/json

{
  "favorite": true   // 或 false
}
```

成功响应示例

- 状态码：200 OK

```
{ "favorite": true }
```

失败响应示例

```
{ "message": "同步收藏状态失败" }
```

前端处理

1. 立即切换 isFavorited.value;
2. 后端失败时回滚并可提示用户。

handleCoinClick()

作用 • 切换用户对该小说的“投币”操作，并同步到后端，从用户余额扣除相应硬币。

触发方式 • 用户点击“投币”按钮时调用。

```
HTTP 请求 POST /api/novel/{novelId}/coin
Host: <后端域名>
Authorization: Bearer <token>
Content-Type: application/json

请求体 {}

成功响应示例 200 OK
{
"newBalance": 99
}

失败响应示例 400 Bad Request，余额不足
{
"message": "余额不足"
}
```

前端处理

1. 立即进行登录状态校验，未登录时跳转到登录页；
2. 接口调用成功后，更新用户余额或显示“投币成功”提示；
3. 后端返回余额不足时，提示“余额不足”；
4. 其他错误时，打印错误并可展示“投币失败，请重试”。

NovelContent.vue

loadNovelMeta()

作用

- 从后端获取小说元信息（主要是标题），用于页面顶部展示。

触发方式

- 组件挂载时 onMounted() 自动调用一次。

HTTP 请求

```
GET /api/novel/{novelId}
Host: <后端域名>
Authorization: Bearer <token>
```

成功响应示例

- 状态码：200 OK

```
{
  "id": 123,
  "title": "惜花芷",
  "author": "空留",
  "updateTime": "2025-04-15T11:45:00Z",
  …
}
```

失败响应示例

```
{ "message": "加载小说信息失败" }
```

前端处理

- novelTitle.value = data.title;
- 错误时 console.error 并可提示用户。

---

loadChaptersList()

作用

- 获取小说的章节列表，用于“上一章/下一章”导航。

触发方式

- 组件挂载时 onMounted() 自动调用一次。

HTTP 请求

```
GET /api/novel/{novelId}/chapters
Host: <后端域名>
Authorization: Bearer <token>
```

成功响应示例

- 状态码：200 OK

```
[
  { "id": 1, "title": "第一章 替嫁" },
  …
```

]

失败响应示例

```
{ "message": "加载章节列表失败" }
```

前端处理

- chapters.value = data;
- 计算 currentChapter、nextChapter、prevChapter。

---

loadChapterContent()

作用

- 拉取当前章节的完整文本内容，用于正文按行渲染。

触发方式

- 组件挂载时调用；
- 路由参数变化时再次调用。

HTTP 请求

```
GET /api/novel/{novelId}/chapters/{chapterId}/content
Host: <后端域名>
Authorization: Bearer <token>
```

成功响应示例

- 状态码：200 OK

```
{
  "id": 2,
  "title": "第二章 拿惯银枪的手",
  "content": "第一段…\n 第二段…\n…"
}
```

失败响应示例

```
{ "message": "加载章节内容失败" }
```

前端处理

- currentChapter.value = data;
- 由 contentLines 计算属性拆行渲染;
- 错误时 console.error 并可提示用户。

Search.vue

searchNovels(q)

作用

- 向后端发起搜索请求，根据关键词获取匹配的小说列表。

触发方式

- doSearch() 点击搜索按钮或回车时调用;
- 页面加载或路由 ?q=xxx 变化时调用。

HTTP 请求

```
GET /api/novel/search?q={q}
Host: <后端域名>
Authorization: Bearer <token>
```

请求参数

参数	类型	必填	说明
q	String	是	搜索关键词

成功响应示例

- 状态码：200 OK

```
[
  {
    "id": 101,
    "title": "惜花芷",
    "author": "空留",
    "status": "连载中",
    …
  },
  …
]
```

失败响应示例

```
{ "message": "搜索失败，请稍后重试" }
```

前端处理

1. searchResults.value = data;
2. hasSearched.value = true;
3. 错误时 console.error 并可提示用户;
4. 结果由 filtered 计算属性渲染。