

iryan-blog 前后端系统代码质量评估与安全性审查报告

一、审查概述

1.1 项目背景

本项目为 iryan-blog，一个前后端分离的个人博客系统，部署于服务器 `/www/wwwroot/iryan_site`，域名 `iryan.icu`。系统采用 Node.js + Express + TypeScript + MySQL + React 技术栈，以 npm workspaces monorepo 组织为 client、server、shared 三个子包。本次审查针对项目中由 AI 生成的前后端系统代码，进行系统性的质量评估与安全性审查。

1.2 审查范围与方法

审查范围覆盖后端 `server/src` (routes、services、models、middleware)、前端 `client/src` (pages、components、services、stores) 以及共享层 `shared`。审查方法包括：代码静态核查、远程接口实测（服务器 `127.0.0.1:3002`）、依赖安全扫描（npm audit）以及前后端契约一致性比对。

1.3 总体结论

综合各维度审查结果如下表所示。

维度	结论	评级
功能正确性	API 测试 35/35 通过（100%），注册/认证全链路验证通过	良好
代码质量	分层清晰、命名规范、注释完整，但存在若干前后端契约不一致	中等
安全性	认证/访问控制基础扎实，但存在 1 处高危 XSS、验证码用 Math.random 等隐患	需改进
AI 幻觉	识别出约 9 处“自信地犯错”实例	需重视

维度	结论	评级
依赖安全	后端 4 个、前端 3 个已知漏洞，含 2 个高危	需修复

其中需要立即处理（P0）的问题有两项：一是文章内容经 `rehypeRaw` 渲染且无 `rehype-sanitize` 白名单，存在存储型 XSS 风险；二是验证码与临时密码使用 `Math.random()` 生成，随机数可预测，应改用 `crypto.randomInt`。

二、功能性测试

为验证 AI 生成代码的功能正确性，我编写并执行了 API 接口测试与认证全链路测试，并对单元测试、前端组件测试的覆盖情况进行了核查，统计测试通过率。

2.1 API 接口测试

测试脚本 `scripts/api_test.sh` 在服务器 127.0.0.1:3002 上共执行 35 个用例，覆盖健康检查、公开接口、文章详情、访问控制、认证、评论、输入验证、点赞与上传等场景，全部通过。

分类	用例	预期	实际	结果
健康检查	GET /api/health	200	200	通过
公开接口	GET /api/articles、/api/articles?limit=6	200	200	通过
公开接口	GET /api/categories、/api/tags	200	200	通过
公开接口	GET /api/platforms、/api/links	200	200	通过
公开接口	GET /api/home/sections、/api/authors/9	200	200	通过
公开接口	GET /api/comments/article/17	200	200	通过
文章详情	GET /api/articles/{slug}、/api/articles/id/17	200	200	通过
访问控制	GET /api/admin/*（未授权，共 11 项）	401	401	通过
认证	注册无验证码 / 错误验证码	400	400	通过
认证	POST /send-register-code	200	200	通过
认证	登录错误密码	401	401	通过
评论	正常 / 缺 article_id / 非整数 id / 空内容	201/400/400	同	通过

分类	用例	预期	实际	结果
输入验证	弱密码 / 非法邮箱	400	400	通过
点赞	POST /api/likes/toggle 未登录	401	401	通过
上传	/api/attachments、/api/images/upload 未登录	401	401	通过

2.2 认证全链路测试

scripts/register_test.sh 对注册、登录、鉴权、刷新令牌等完整链路进行了验证，结果如下。

步骤	结果
发送注册验证码 → 数据库取码	通过
用验证码注册新用户	通过（返回 user + token + refreshToken）
登录（邮箱 + 密码）	通过
访问受保护接口 /auth/me	通过（200）
普通用户访问 /admin/users	通过（403，权限正确拦截）
刷新令牌 /auth/refresh	通过（返回新 accessToken）
伪造 token / 空 token	通过（401）

2.3 评论接口输入校验测试

审查中发现评论接口声明了 express-validator 校验规则却漏挂 validate 中间件，导致输入校验失效。修复后我对关键边界进行了复测。

用例	修复前	修复后
缺 article_id	500（崩溃）	400（通过）
非整数 article_id	500（NaN in SQL）	400（通过）
空内容	201（错误接受）	400（通过）
超长昵称	一直通过	400（通过）
正常评论	201	201（通过）

2.4 单元测试与前端组件测试

为验证核心业务逻辑的正确性与关键 UI 组件的可用性，本项目引入 Vitest 测试框架，对服务端核心 service 模块补充单元测试，对客户端关键组件补充组件测试，并将测试纳入 GitHub Actions CI 流水线，确保每次提交自动执行测试。

模块/组件	测试文件	用例数	覆盖要点
auth.service	auth.service.test.mts	31	注册（邮箱/用户名冲突、验证码校验与过期）、登录（邮箱/用户名、密码校验）、token 刷新、资料获取与更新、修改密码、临时密码生成、验证码发送冷却、忘记/重置密码
article.service	article.service.test.mts	20	创建文章（slug 生成与冲突、定时/立即发布状态）、文章详情与浏览计数、分页列表（权限过滤、排序白名单、分页边界）、更新（slug 冲突、状态自动调整）、删除、关键词搜索
comment.service	comment.service.test.mts	13	创建评论（登录直接通过、游客待审核、匿名默认昵称）、分页查询、管理员状态筛选、审核通过/拒绝、删除
StatusBadge	StatusBadge.test.tsx	6	各状态文案渲染、未知状态回退
Pagination	Pagination.test.tsx	8	单页不渲染、当前页高亮、页码点击、上一页/下一页、边界禁用、省略号折叠、页码概要
FormInput	FormInput.test.tsx	6	标签与输入框渲染、错误提示、onChange 回调、图标、原生属性透传
ArticleCard	ArticleCard.test.tsx	5	标题/摘要/分类/标签渲染、链接跳转、无摘要占位、阅读量与点赞数、无分类标签降级
CommentItem	CommentItem.test.tsx	6	评论内容与用户名、匿名标识、游客昵称、删除按钮显隐、删除回调
合计	8 个测试文件	95	服务端 64 个 + 客户端 31 个

综合来看，功能正确性验证通过率为 100%。

2.5 测试通过率统计

测试类型	用例数	通过数	通过率
API 接口测试	35	35	100%
认证全链路测试	7	7	100%
评论接口校验测试	5	5	100%
单元测试	64	64	100%
前端组件测试	31	31	100%

综合来看，功能正确性验证通过率为 100%。

三、代码质量评估

3.1 可读性

项目整体遵循 routes → services → models / middleware 的分层架构，各层职责清晰。变量、函数与文件命名语义明确（如 generateUniqueSlug、extractFirstImage），文件头注释与关键函数 JSDoc 齐全，错误处理统一走 try/catch 与业务错误码映射，可读性良好。

检查项	结论
分层架构	清晰：routes → services → models / middleware
命名规范	语义清晰，无歧义缩写
注释完整性	文件头注释、关键函数 JSDoc 齐全
代码块结构	错误处理统一 try/catch + 业务错误码映射

3.2 可维护性

service 层封装了主要业务逻辑，controller 较薄，模块耦合度合理，大部分函数长度控制在合理范围。但存在一定重复代码：getCategories/getTags 在多个页面与服务中重复定义，文章摘要提取前后端各实现一份，后端 getArticles 与 searchArticles 的分页与 include 逻辑重复，未完全遵循 DRY 原则。

检查项	结论
模块耦合度	合理：service 层封装业务，controller 薄

检查项	结论
函数长度	大部分函数长度合理
重复代码	getCategories/getTags 多处重复；摘要提取前后端各一份
DRY 原则	getArticles 与 searchArticles 分页/include 逻辑重复

3.3 性能

数据库查询使用 findAndCountAll + distinct 实现分页，避免全表加载，未发现典型的 N+1 查询问题。但 getArticles 列表在提取每篇文章 first_image 时需要额外解析 content，文章量大时开销线性增长，可考虑落库缓存。前端 ArticleListPage 排序按钮使用内联箭头函数、ArticleCard 未使用 React.memo，存在轻微的不必要重渲染，规模小影响有限。公开接口远程实测响应正常，无明显延迟。

检查项	结论
N+1 查询	未发现典型 N+1；first_image 提取需解析 content，可缓存优化
include 效率	findAndCountAll + distinct，避免全表加载
前端重渲染	ArticleCard 未 memo、排序按钮内联箭头函数，影响低
API 响应时间	公开接口远程实测正常，无明显延迟

良性确认：HomePage 使用 Promise.allSettled 避免单接口失败拖垮整页，各页面均有 loading / error / cancelled 守卫。

3.4 前后端一致性

前后端一致性是本次审查发现问题的集中区域，共识别出 4 处契约不一致或实现偏差。

(1) 问题 C1: updateProfile 接口契约错误 + 死代码

client/src/services/authApi.ts 中定义的 UpdateProfileData 类型为 {username, avatar, bio}，而后端 /auth/profile 接口实际只接收 display_name 与 bio，username、avatar 均不被后端接受。该函数全项目无人调用，属于死代码；真实资料保存走 ProfilePage 的 /auth/profile/full 接口。

(2) 问题 C2: 路由顺序 Bug 导致分类排序失效

`admin.routes.ts` 中 `PUT /categories/:id` 注册在 `PUT /categories/sort-order` 之前，Express 按顺序匹配会把 "sort-order" 当作 `:id`，`parseInt` 后得到 `NaN`，导致排序接口返回 404。姊妹文件 `category.routes.ts` 顺序正确，说明是 AI 生成时的遗漏，本批次已修复。

(3) 问题 C3：文章状态枚举不一致

前端 `AdminArticle.status` 与创建/更新参数包含 "scheduled"，而后端 `admin.routes.ts` 校验仅允许 `draft`、`published`、`archived`，提交 `scheduled` 会被后端 400 拒绝。

(4) 问题 C4：匿名评论先显示后消失

后端未登录评论状态为 `pending`，列表接口只返回 `approved`；前端 `CommentSection` 无条件将新评论 `prepend` 进列表，导致匿名评论短暂显示、刷新后消失。

四、Web 安全性审查

Web 安全性审查为本报告的重点，从身份认证、访问控制、输入验证、文件上传、敏感信息与依赖安全六个方面逐项核查。

4.1 身份认证

系统采用 JWT 双 token 机制（`access` + `refresh`），密码使用 `bcryptjs` 加密存储（10 轮 `salt`），登录时用 `bcrypt.compare` 校验。JWT 密钥与过期时间由环境变量控制，刷新机制经实测正常。但存在两处安全隐患：代码中留有 JWT 默认密钥硬编码（生产 `.env` 已覆盖，长度偏短建议加长）；验证码与临时密码使用 `Math.random()` 生成，随机数可预测。

检查项	结论	说明
密码存储	通过	<code>bcryptjs.hash</code> （10 轮 <code>salt</code> ）， <code>bcrypt.compare</code> 校验
JWT 密钥管理	需改进	存在硬编码默认值；生产 <code>.env</code> 已覆盖，建议 ≥ 32 随机字符
JWT 过期策略	通过	<code>access</code> / <code>refresh</code> 双 token， <code>expiresIn</code> 由 env 控制
刷新机制	通过	<code>/auth/refresh</code> 验证 <code>refreshToken</code> 签发新 <code>accessToken</code> （实测）
验证码生成	未通过	<code>Math.random()</code> 可预测，应改用 <code>crypto.randomInt</code>

检查项	结论	说明
临时密码生成	未通过	Math.random(), 同上

4.2 访问控制

管理接口统一挂载 authenticate + requireAdmin 中间件，实测 11 个 /api/admin/* 未授权均返回 401，普通用户访问返回 403，垂直权限控制有效。评论删除校验 user_id 归属，水平权限控制正确。前端 /editor 路由已补加权限守卫。需注意前端 isAuthenticated 仅凭 localStorage token 存在性判断、role 信任 localStorage，可被篡改看到管理 UI，但权限由后端兜底，风险可控。

检查项	结论	说明
管理接口保护	通过	11 个 /api/admin/* 未授权全部 401（实测）
垂直权限	通过	普通用户访问 admin 返回 403（实测）
水平权限	通过	评论删除校验 user_id === 评论作者 admin
错误页面守卫	通过	/editor 已加 AuthorRoute 守卫（本批次修复）
前端 UI 门控	需改进	仅凭 localStorage token/role，可篡改看到管理 UI，后端兜底

4.3 输入验证

数据库操作全部使用 Sequelize 参数化查询，未发现 SQL 注入。评论与昵称通过 React `{}` 插值渲染，React 自动转义，且全项目未使用 dangerouslySetInnerHTML，XSS 风险较低。但文章渲染链路使用 rehypeRaw 渲染原始 HTML 且未配置 rehype-sanitize 白名单，存在存储型 XSS 向量（如 img onerror、svg/onload）。CSRF 方面，JWT 通过 Authorization header 传递（非 cookie），天然免疫经典 CSRF，且已启用 helmet。

检查项	结论	说明
SQL 注入	通过	全部 Sequelize 参数化查询，无字符串拼接 SQL
XSS - 文章渲染	未通过	rehypeRaw 渲染原始 HTML 无 rehype-sanitize 白名单
XSS - 评论/昵称	通过	React <code>{}</code> 插值自动转义

检查项	结论	说明
XSS – dangerouslySetInnerHTML	通过	全项目未使用
CSRF	通过	JWT 走 Authorization header + helmet
后端输入清洗	需改进	sanitize.ts 用 stripTags 正则全局清洗，会误删合法 HTML

4.4 文件上传

上传功能通过 multer 配置了文件类型白名单（imageMimeTypes / attachmentMimeTypes）与大小限制（头像 5MB、图片/附件 10MB），文件名使用时间戳 + 随机串重写，不使用原始文件名，可防止路径遍历。但仅信任客户端 Content-Type，未校验文件魔数（magic bytes），可被伪造为合法图片 MIME 上传任意内容；文件名随机串使用 Math.random() 生成，熵较低。

检查项	结论	说明
文件类型限制	通过	fileFilter 白名单 imageMimeTypes / attachmentMimeTypes
文件大小限制	通过	avatar 5MB, image 10MB, attachment 10MB
文件头（魔数）校验	未通过	缺失，仅信任客户端 Content-Type
路径遍历	通过	文件名重写，不使用原始文件名
文件名熵	需改进	Math.random().toString(36) 熵较低

4.5 敏感信息

SMTP 与数据库凭据均通过 .env 配置，未硬编码。接口统一返回通用错误信息（如“分类不存在”），堆栈仅 console.error 打日志，未回传给客户端。CORS 从 .env 读取白名单，仅含 iryan.icu 相关域名与本地开发地址，配置正确。

检查项	结论	说明
密钥硬编码	需改进	JWT 有默认值硬编码（生产已覆盖）；SMTP/DB 凭据走 .env
错误信息泄露	通过	统一通用错误信息，堆栈仅打日志不回传

检查项	结论	说明
CORS 配置	通过	从 .env 读取，白名单仅含 iryan.icu 相关域名 + 本地开发

4.6 依赖安全

使用 npm audit（官方源）对前后端依赖进行扫描，结果如下。

位置	包	严重度	类型	说明
server	nodemailer	高危	direct	SMTP 命令注入 / 任意文件读取
server	dottie	中危	transitive	原型污染（经 sequelize）
server	sequelize	中危	direct	经 uuid 引发的原型污染链
server	uuid	中危	transitive	<11.1.1 原型污染
client	nanoid	高危	transitive	<3.3.18 DoS
client	react-router-dom	中危	direct	链到 react-router 6-7.17
client	react-router	中危	transitive	同上

后端合计 4 个（1 高 3 中），前端合计 3 个（1 高 2 中），无 Critical 级漏洞。

五、AI 幻觉问题识别

5.1 幻觉案例

#	严重度	实例	位置	本质	状态
1	高危	updateProfile 声明类型 {username,avatar,bio}, 后端实际要 {display_name,bio}, 且无人调用	authApi.ts:25,85	虚构 API 契约 + 死代码	建议删除/修正
2	高危	rehypeRaw 渲染原始 HTML 却不配 rehype-sanitize	ArticleDetailPage.tsx:166、MarkdownEditor.tsx:648	安全实现错误	需修复

#	严重程度	实例	位置	本质	状态
3	高危	验证码 / 临时密码用 Math.random()	auth.service.ts:57,547	安全实现错误（可预测随机数）	需修复
4	中危	声明校验规则却漏挂 validate 中间件	comment.routes.ts	代码遗漏（规则写了没生效）	本批次修复
5	中危	/categories/sort-order 注册在 /:id 之后	admin.routes.ts	Express 配序误区	本批次修复
6	中危	前端虚构 /admin/categories/reorder 路径	client adminApi.ts	虚构接口路径	前序修复
7	中危	AdminArticle.status 枚举含后端不接受的 scheduled	client adminApi.ts	契约漂移	需统一
8	低危	User 类型同时含 avatar? 与 avatar_url?	types/index.ts	臆测冗余字段	需清理
9	低危	全局 stripTags 正则清洗所有字符串	server/src/middleware/sanitize.ts	自相矛盾的安全设计	需重构

5.2 原因分析

综合上述案例，AI 幻觉的产生主要有以下原因。

1. 缺乏契约校验：前端类型与后端字段未用单一数据源（OpenAPI / shared types）约束，各写各的，漂移无从察觉。
2. 自定义安全边界：AI 为省事使用 Math.random、stripTags、rehypeRaw 等“看似可行”的方案，未考虑其安全与语义后果。
3. 框架 API 误记：rehypeRaw 默认不做净化；Express 路由顺序、Sequelize 类字段 declare 遮蔽 getter 等均为常见坑。
4. 规则不统一：同类路由（category.routes.ts 与 admin.routes.ts）一个对、一个错，说明生成样本不一致。

5. 测试缺失：开发周期内无单元/组件测试，死代码与错误契约长期未被发现。

5.3 防范策略

- 1. 建立单一 API 契约：用 OpenAPI 或 shared/ 类型作为前后端唯一字段来源，并加 CI 校验请求/响应结构。
- 2. 安全默认值：验证码一律 `crypto.randomInt`；富文本渲染一律白名单 `sanitize`；上传校验文件魔数。
- 3. 补充测试：引入 vitest 与关键 service/组件单测及契约测试，让幻觉在 CI 暴露。
- 4. 路由顺序规范：固定参数的具名路由（/sort-order）一律在 /:param 之前注册，并加路由存在性测试。
- 5. 代码审查清单：将上述高频幻觉点写入审查 checklist。

六、安全审查清单与修复记录

6.1 安全审查清单与修复状态

类别	检查项	修复前状态	当前状态	修复说明
认证	验证码不可预测	未通过 (Math.random)	已修复	auth.service.ts 改用 <code>crypto.randomInt(100000, 1000000)</code> 生成 6 位数字验证码
认证	临时密码不可预测	未通过 (Math.random)	已修复	临时密码各字符位改用 <code>crypto.randomInt</code> 从字符池中选取
认证	上传文件名随机串	需改进 (Math.random)	已修复	文件名随机串改用 <code>crypto.randomBytes(4).toString("hex")</code>
认证	JWT 密钥管理	需改进（硬编码默认值）	已修复	移除硬编码默认值，改为 <code>process.env.JWT_SECRET crypto.randomBytes(32)</code> ；生产 .env 已配置 48 字符独立密钥
访问控制	前端 UI 门控	需改进（可篡改看到管理 UI）	维持现状	权限由后端 <code>authenticate + requireAdmin</code> 中间件兜底，前端门控仅作体验优化，篡改不影响实际权限，风险可控

类别	检查项	修复前状态	当前状态	修复说明
输入	XSS（文章渲染）	未通过（rehypeRaw 无 sanitize）	已修复	ArticleDetailPage.tsx、MarkdownEditor.tsx 均接入 rehype-sanitize + defaultSchema 白名单，仅允许安全的 HTML 标签与属性
输入	后端全局 stripTags	需改进（正则误删合法 HTML）	已修复	sanitize.ts 中间件已删除，不再对请求体做正则清洗
输入	评论接口 validate 中间件漏挂	未通过（校验规则未生效）	已修复	comment.routes.ts 补挂 validate 中间件，缺 article_id、非整数 id、空内容等边界用例复测通过（见 2.3 节）
上传	文件魔数校验	未通过（缺失）	已修复	upload.ts 新增 MAGIC_SIGNATURES 签名表（JPEG/PNG/GIF/WebP/SVG/PDF/ZIP/DOC）与 verifyFileMagic 中间件，校验失败删除文件并返回 400
契约	updateProfile 死代码	需修复（虚构 API 契约）	已修复	authApi.ts 中 updateProfile 函数及其类型定义已删除（全项目无人调用）
契约	虚构 /admin/categories/reorder 路径	需修复（前端调用不存在的接口）	已修复	前端 adminApi.ts 移除虚构路径，改用真实接口
契约	文章状态枚举不一致	需统一（scheduled 被后端 400 拒绝）	已修复	admin.routes.ts 状态校验数组接受 scheduled；admin.service.ts 新增 scheduled 状态支持
契约	路由顺序 Bug	需修复（sort-order 被 :id 捕获）	已修复	admin.routes.ts 中 PUT /categories/sort-order 已前置至 PUT /categories/:id 之前注册
契约	avatar 冗余字段	需清理（avatar? 与 avatar_url? 并存）	已修复	types/index.ts 统一保留 avatar_url，删除冗余的 avatar 字段声明
体验	匿名评论先显示后消失	需优化（pending 评论插入列表后刷新消失）	已修复	CommentSection.tsx 对未登录用户提交的 pending 评论不再插入列表，改为显示"您的评论已提交，需经过审核后才能显示"提示

类别	检查项	修复前状态	当前状态	修复说明
依赖	nodemailer 高危漏洞	需修复（SMTP 命令注入）	已修复	升级至 ^9.0.5，npm audit 不再报告
依赖	nanoid 高危漏洞	需修复（<3.3.18 DoS）	已修复	升级至 3.3.18，npm audit 不再报告
依赖	uuid / sequelize 中危漏洞	需修复（原型污染链）	已修复	通过 package.json overrides 将 uuid 固定至 ^11.1.1，npm audit 不再报告
依赖	react-router-dom / react-router 中危漏洞	需修复（开放重定向）	已修复	升级至 7.18.2，npm audit 不再报告
测试	单元测试 / 组件测试	未配置（覆盖率 0%）	已补充	引入 Vitest，8 个测试文件 95 个用例全部通过（详见 2.4 节）
测试	测试纳入 CI	未配置	已配置	.github/workflows/ci.yml 在 push / pull_request 时自动运行两端测试与客户端构建检查

当前 `npm audit` 前后端均为 0 漏洞，无遗留待修复安全条目。

七、项目优化建议

审查发现的高优先级问题已全部修复，代码已同步至本地、GitHub、服务器三端。本节从架构契约、安全运营、性能体验与工程化四个维度，对后续优化方向给出建议。

7.1 架构与契约

1. 建立单一契约源。当前 `updateProfile` 死代码、`scheduled` 状态枚举不一致等问题均源于前后端各写各的类型定义，建议将公共类型收敛至 `shared/` 包，并以 OpenAPI 或 TypeScript 类型作为唯一字段来源，在 CI 中增加契约一致性校验。

2. 治理重复代码。`getCategories` / `getTags` 在多个页面与后端服务中重复定义，文章摘要提取前后端各实现一份，后端 `getArticles` 与 `searchArticles` 的分页与 `include` 逻辑重复。建议收敛到单一实现，减少维护成本。

7.2 安全运营

1. 建立依赖漏洞主动监控机制。当前已修复全部已知漏洞，但依赖库持续更新，建议将 `npm audit` 纳入 CI 门禁或定期手动执行，并在依赖升级后执行回归验证。
2. JWT 密钥定期轮换。生产环境密钥已加长至 48 字符，建议建立季度轮换机制，并确保密钥仅存储在 `.env` 中、不进入版本控制。
3. 关注核心依赖安全公告。Express、Sequelize、MySQL2 等关键依赖的安全性直接影响系统，建议订阅官方安全公告渠道。

7.3 性能与体验

1. 文章列表 `first_image` 落库缓存。当前 `getArticles` 在提取每篇文章首图时需解析 `content` 字段，文章量大时开销线性增长，建议将首图路径落库缓存或延迟计算。→未实施
2. 列表项使用 `React.memo` + 稳定 key。`ArticleCard` 未使用 `React.memo`，排序按钮使用内联箭头函数，存在轻微不必要重渲染。当前规模下影响有限，可作为后续优化项。
3. 路由顺序规范制度化。`sort-order` 被 `:id` 捕获的问题已修复，但根因在于缺少规范约束。建议将“具名路由前置”写入代码规范，并增加路由存在性测试，防止同类问题复现。

7.4 工程化

1. 补充契约测试与覆盖率门禁。当前测试覆盖核心 `service` 与关键组件，但未设置覆盖率阈值。建议在 CI 中配置覆盖率门禁（如 80%），并补充前后端接口契约测试。
2. 前端构建体积优化。当前 `rollup` 构建报 `chunk size > 500 kB` 警告，建议对大体积依赖（如 `react-markdown` 及其插件链）启用动态导入或代码分割。

3. 持续完善测试覆盖。当前 95 个用例覆盖了 auth、article、comment 三大核心服务与 5 个关键组件，建议后续为 admin 服务、上传服务、store 层补充测试。

八、结论

本次审查对 iryan-blog 项目进行了系统性的质量评估与安全性审查，并完成了大量修复。

功能性测试方面，API 接口测试 35/35 全部通过，认证全链路验证正常。引入 Vitest 后，服务端核心 service 单元测试 64 个用例、客户端关键组件测试 31 个用例全部通过，覆盖 auth、article、comment 三大核心服务与 StatusBadge、Pagination、FormInput、ArticleCard、CommentItem 五个关键组件，均已纳入 GitHub Actions CI 流水线。

安全性方面，审查发现的所有安全问题已修复：文章渲染链路接入 rehype-sanitize 白名单（消除存储型 XSS 风险）、验证码与临时密码改用 crypto.randomInt（消除随机数可预测问题）、上传层新增文件魔数校验、全局 stripTags 正则清洗删除。依赖漏洞方面，nodemailer、nanoid、uuid、react-router-dom 等全部升级，npm audit 前后端均为 0 漏洞。JWT 密钥已加长至 48 字符随机串。

代码质量方面，项目分层清晰、命名规范、注释完整。前后端契约不一致问题（updateProfile 死代码、scheduled 状态枚举、avatar 冗余字段）已清理，路由顺序 Bug 已修复，匿名评论用户体验已优化。认证与访问控制基础扎实，SQL 注入与 CSRF 防护到位。

当前项目代码已同步至本地、GitHub（Ryan-alpaca/iryan-blog）与服务器（/www/wwwroot/iryan_site）三端，最新提交 c99bfaa，生产环境运行正常。后续建议按第七部分推进架构契约统一、安全运营常态化、性能优化与工程化提升，持续完善 AI 生成代码的质量与安全性。