软件配置与运维文档

本项目采用现代化的软件配置管理、版本控制、持续集成、部署与运维手段,确保系统的高可用性、可维护性和可扩展性.以下为详细说明.

一、配置管理

1. 依赖与环境配置

- **依赖管理**:所有依赖通过 `package.json` 统一声明, `package-lock.json` 锁定依赖版本,确保开发、测试、生产环境一致.
- **Node 版本约束**: `.npmrc` 文件中设置 `engine-strict=true` ,强制团队成员使用指定 Node 版本,避免环境不一致问题.
- **环境变量管理**:敏感信息和环境相关配置通过 `.env` 文件管理, `.gitignore` 已忽略 `.env` ,防止泄露.

2. 配置文件

- **TypeScript 配置**: `tsconfig.json` 统一管理 TypeScript 编译选项,保证类型安全和一致性.
- **SvelteKit 配置**: `svelte.config.js` 、 `vite.config.ts` 配置前端构建与开发环境.
- **测试配置**: `vitest.config.ts` 配置测试环境,支持自动化测试和测试数据库隔离.
- **文档生成配置**: `typedoc.json` 配置自动生成 API 文档.

二、版本控制

- **Git 管理**:项目采用 Git 进行版本控制,所有源代码、配置文件、文档等均纳入 Git 管理.
- **忽略文件**: `.gitignore` 文件已配置,忽略 node_modules、环境变量、构建产物等无关文件,保持仓库整洁.

三、持续集成(CI)

- **自动化测试**:通过 `vitest` 实现单元测试,确保每次提交和合并都不会破坏现有功能.
- **自动化构建**:集成持续集成工具,实现自动化构建、测试和文档生成.

四、部署

1. 构建与发布

- **前端构建**:使用 Vite 进行前端打包,执行 `npm run build` 生成生产环境静态资源.
- **后端部署**:如有后端服务,可通过 Node.js 运行,或部署到云平台(Vercel).

2. 部署流程

1. 拉取最新代码: `git pull`
2. 安装依赖: `npm install`
3. 配置环境变量:根据实际环境设置 `.env` 文件
4. 构建项目: `npm run build`
5. 启动服务: `npm run preview` 或自定义启动命令

五、未来运维计划

1. 日常运维

- **日志管理**:集成日志系统,记录访问、错误、异常等信息,便于问题追踪和定位.
- **备份策略**:定期备份数据库和重要配置文件,防止数据丢失.
- **监控与告警**:集成监控工具,实时监控服务状态和性能,异常时自动告警.

2. 安全与权限

- **权限控制**:敏感操作需权限校验,防止未授权访问.
- **环境隔离**:开发、测试、生产环境严格隔离,避免数据串扰和安全隐患.
- **依赖安全**:定期检查依赖库安全性,及时修复漏洞.

3. 升级与维护

- **依赖升级**:定期升级依赖,保持技术栈安全和现代化.
- **文档维护**:及时更新配置、部署、运维相关文档,便于新成员快速上手和交接.

六、总结

本项目通过规范的配置管理、严格的版本控制、自动化的持续集成、标准化的部署流程和完善的运维计划,保障了系统的高可用性和可维护性.团队成员应共同遵循上述规范,持续优化运维体系,确保项目稳定健康运行.