软件工程化说明文档

本项目在开发过程中,尝试采用了现代软件工程化手段,提升了开发效率、协作能力和代码质量.以下从自动化与协作化两个维度,详细说明本项目的软件工程化实践.

一、自动化手段

1. 依赖与环境管理

- **依赖锁定**:使用 `package-lock.json` 锁定依赖版本,确保团队成员和 CI/CD 环境下依赖一致,避免“在我电脑上能跑”的问题.
- **Node 版本约束**: `.npmrc` 中 `engine-strict=true` ,强制使用指定 Node 版本,保证运行环境一致.

2. 构建与开发自动化

- **脚本自动化**: `package.json` 中定义了常用脚本（如 `dev` 、 `build` 、 `preview` 、 `check` 、 `test` 等）,一键完成开发、构建、预览、类型检查、测试等流程.
- **现代构建工具**:采用 Vite（ `vite.config.ts` ）和 SvelteKit（ `svelte.config.js` ）作为前端构建与开发工具,支持热更新、快速打包和高效开发体验.

3. 类型与静态检查

- **TypeScript 全面接入**:通过 `tsconfig.json` 配置严格的类型检查,提升代码健壮性和可维护性.
- **Svelte Check**: `svelte-check` 集成于脚本,自动检查 Svelte 组件和 TypeScript 类型.

4. 自动化测试

- **测试框架**:使用 Vitest（ `vitest.config.ts` ）进行单元测试, `tests/` 目录下有数据库相关的测试用例,保障核心功能正确性.
- **测试环境隔离**:通过 `mongodb-memory-server` 实现测试数据库的自动化、隔离,避免污染生产数据.

5. 自动化文档生成

- **Typedoc**: `typedoc.json` 配置自动生成 API 文档, `package.json` 脚本 `docs` 一键生成,输出到 `docs/` 目录,便于团队成员查阅和维护.

二、协作化手段

1. 版本控制与协作

- **Git 管理**:项目采用 Git 进行版本控制,`.gitignore` 文件规范忽略无关文件,保证仓库整洁.
- **分支管理**:通过分支（如 `logoff`）和 Pull Request 流程,支持多人协作开发和代码审核.

2. 目录结构与模块化

- **分层目录结构**:`src/` 下细分为 `lib`（组件、工具、类型、schema、server）、`routes`（页面路由）、`tests`（测试）等,前后端类型、接口、数据库操作、UI 组件等均有独立目录,职责清晰,便于多人协作和维护.
- **类型共享**:`types/` 目录下细分 client/server/share,前后端类型共享,减少重复定义,提升协作效率.

3. 配置与环境隔离

- **环境变量管理**:通过 `.env` 文件（已在 `.gitignore` 忽略）,实现开发、测试、生产环境的配置隔离,保障安全与灵活性.

4. 文档与规范

- **README.md**:详细说明项目目标、功能、技术选型、开发流程、待完成文档等,便于新成员快速上手.
- **自动化 API 文档**:Typedoc 自动生成的文档,提升代码可读性和团队协作效率.

5. 实时面对面沟通

- **面对面高效沟通**:本项目团队三人同住一个宿舍,日常开发中通过实时面对面沟通,快速讨论需求、设计与实现细节,极大提升了协作效率和问题响应速度.

三、总结

本项目通过依赖锁定、类型检查、自动化测试、文档生成、分层目录、类型共享、Git 管理、实时面对面沟通等多项软件工程化手段,实现了高效的自动化与协作化开发流程.团队成员持续遵循上述规范,共同维护高质量的工程实践.