

结论：不通过（未达到“可上线 / 可维护”标准）

最关键 3 个原因（都属于上线高风险）：

1. **MCP 工具执行错误没有按规范用** `isError: true` **表达**：当业务返回 `{error, code}`（比如 `NO_DATA / USER_NOT_FOUND`）时，MCP Host 会把它当“成功结果”而不是“可自我纠错的工具错误”，导致 LLM 误判、无法正确重试/改参（见 `src/index.ts` CallTool handler；规范要求工具执行错误用 `isError: true`）。([Model Context Protocol](#))
2. **HTTP 传输安全不合规 (P0)**：Streamable HTTP 要求校验 `Origin` 以防 DNS rebinding，但实现里完全没有做 Origin 校验，且 CORS `Access-Control-Allow-Origin: *` + 绑定 `0.0.0.0`，使“浏览器网页/恶意站点”更容易打到本地/内网 MCP 服务，存在数据外泄风险（见 `src/server/http.ts`）。([Model Context Protocol](#))
3. **工具 Schema 与实际校验/能力不一致，契约不稳定**：例如 `group_by` 实际支持 `type`，但 tool definition 的 JSON Schema 没暴露；`list_workloads` 代码支持 `filter_project_id`，但 tool definition schema 未声明。这会直接导致 MCP Host/LLM 生成参数错误、或你后续版本升级不可控（见 `src/tools/userWorkSummary.ts` / `teamWorkSummary.ts` / `listWorkloads.ts`）。

下面按你要求的 1~6 步严格审。

1. 需求对齐与功能覆盖

PRD 我以“可验收行为”方式抽取如下（5~10 条，可直接写测试/验收用）。PRD 文件：

1.1 需求“可验证行为”（验收标准级别复述）

1. **团队汇总**：输入 `time_range.start/end`（≤3个月）且不传 `user_ids`，返回“全员列表”+ 每人 `total_hours` + 每人 Top work items/projeid；若有 `work_item_id`，能解析到 `identifier/title``。
2. **超 3 个月自动分片**：输入跨度 >3个月时，系统透明分片调用 `workloads` 并合并 totals，且在输出里标记 `data_quality.time_sliced=true`。
3. **鉴权/配置**：未提供 PingCode token 时要给“标准错误（401/配置错误）”；对 PingCode API 必须使用 `Authorization: Bearer <token>`。
4. **基础工具齐全**：至少提供 `list_users` / `list_workloads` / `get_work_item`，且返回结构稳定、可被 LLM 组合。
5. **用户汇总工具**：输入人（id 或 name）+ 时间段 + `group_by`（day/week/month/work_item/project/type），输出 `summary`（总工时、TopN、分组）+ `details`（可审计/追问）+ `data_quality`（`workloads_count`、`missing_work_item_count`、`time_sliced`）。
6. **团队汇总工具**：输入时间段（可选 `project_id` / `user_ids` / `group_by` / `matrix`），输出 `summary`（每人汇总）+ 可选 `matrix` + 可追溯明细。
7. **性能**：工作项详情批量拉取+缓存，避免 N+1；`workloads` 分页/限流处理合理。
8. **可靠性**：网络错误重试（指数退避）。但不放大请求；限流不致雪崩。

9. 安全: token 不落日志; 敏感字段脱敏; 建议内网部署/或加鉴权。

10. 可观测性: 请求量、失败率、分片次数、耗时、缓存命中率可观测。

1.2 逐条对齐: 已实现 / 未实现 / 不完整 (带证据)

下面我按 PRD T1~T5 + AC/NFR 对齐 (证据给到文件/函数/调用链)。

T1 `list_users`

- 状态: 已实现 (但分页语义有“二次分页”设计差异)
- 证据
 - MCP 工具: `src/tools/listUsers.ts` `listUsers()` + `listUsersToolDefinition`
 - PingCode API: `src/api/endpoints/users.ts` `listUsers()` 调 `/v1/directory/users`, 自动翻页聚合
 - user 缓存: 无 (代码注释明确不缓存用户列表)
- 备注
 - 工具层参数 `page_index/page_size` 是“本地切片分页”, 不是 PingCode 原生分页 (因为底层已拉全量再 slice)。这不违背 PRD, 但要在文档里明确, 否则用户以为是服务端分页。

T2 `list_workloads`

- 状态: 基本实现, 但工具契约不稳定 (Schema 不一致), 且与 PRD“建议最小集”不同
- 证据
 - MCP 工具: `src/tools/listWorkloads.ts` `listWorkloads()` + `listWorkloadsToolDefinition`
 - PingCode API: `src/api/endpoints/workloads.ts` `listWorkloads()` 调 `/v1/workloads`
 - 3 个月分片: `src/api/endpoints/workloads.ts` `isTimeRangeExceedsThreeMonths()` + `splitTimeRangeIntoChunks()` + `seenIds` 去重
- 不完整点
 - **tool definition schema 缺失 `filter_project_id`**: 代码支持 (Zod schema 里有 `filter_project_id`, 并用于 `pilot_id` 过滤), 但 `listWorkloadsToolDefinition.inputSchema.properties` 没有声明它 → Host 侧不会生成/传参。
 - `principal_type/project` 被转成 `pilot_id` 过滤, 而不是原生 `principal_type=project` (PRD 允许 mapping layer, 但要写清“为什么”与“等价性”)。
- 与官方约束符合占

- PingCode `/v1/workloads` start/end 会被转换到“对应日期起止时间点”，且 start/end 需同时存在（代码用 `parseTimeRange()` 强制二者）。(open.pingcode.com)
- page_index 从 0 开始的假设与官方示例一致（代码用 0 起算）。(open.pingcode.com)

T3 `get_work_item`

- 状态：已实现
- 证据
 - MCP 工具： `src/tools/getWorkItem.ts` `getWorkItem()`
 - PingCode API： `src/api/endpoints/workItems.ts` `getWorkItem()` 调 `/v1/project/work_items/{id}`
 - 缓存： `src/api/endpoints/workItems.ts` 使用 `src/cache/memory.ts` (TTL 默认 6h)
- 备注
 - 批量拉取： `getWorkItemsBatch()` 实现了并发控制（默认 10）+ mget/mset 缓存批写。

T4 `user_work_summary`

- 状态：已实现，但 `group_by/type` 的 schema 暴露不完整；错误返回未按 MCP `isError`
- 证据
 - MCP 工具： `src/tools/userWorkSummary.ts` `userWorkSummary()` + `userWorkSummaryToolDefinition`
 - 业务聚合： `src/services/workloadService.ts` `getUserWorkSummary()` → `listUserWorkloads()` → `api/endpoints/workloads.ts:listWorkloads()` → `workItemService.enrichWorkloadsWithWorkItems()` → `aggregateWorkloads()`
- 不完整点
 - 实际支持 `group_by='type'`，但 tool definition 的 enum 里缺失 `type` (Zod 允许，definition 不允许) → 契约不一致。
 - NO_DATA / USER_NOT_FOUND / AMBIGUOUS_USER 等业务错误是“返回对象”，但 `src/index.ts` 对这类结果不会标 `isError`（只在 throw 或 unknown tool 才 `isError`）。
- 加分点
 - 输出结构接近 PRD 建议：summary + details + data_quality（含 time_sliced、pagination_truncated、details_truncated、missing_work_item_count）。

T5 `team_work_summary`

- 状态：实现但关键验收点不完整（“全员列表”语义）+ schema 暴露不完整 + MCP 错误不规范
- 证据

- MCP 工具: `src/tools/teamWorkSummary.ts` `teamWorkSummary()` + `teamWorkSummaryToolDefinition`
- 业务聚合: `src/services/workloadService.ts` `getTeamWorkSummary()`
- 未满足/不完整
 1. “全员列表”未严格满足: `getTeamWorkSummary()` 里只把 `aggregated.totalHours > 0` 的成员 push 到 `members`, 工时为 0 的用户会被丢掉 (文件: `src/services/workloadService.ts`, for-loop 内 if 条件)。这与 PRD AC1 “不指定 `user_ids` → 全员列表、每人 `total_hours`”有偏差 (至少应支持返回 0)。
 2. `group_by='type'` 实际支持但 **tool definition enum** 缺失 (同 `user_work_summary`)。
 3. `data_quality.missing_work_item_count` 始终为 0: `totalMissingWorkItemCount` 定义了但从未累加 (`src/services/workloadService.ts`)。
 4. 业务错误 (NO_DATA) 返回对象但 MCP 层不标 `isError` (同上)。

NFR / 其他

- 可观测性: 部分实现
 - 指标: `src/utils/metrics.ts` + builtin tool `get_metrics` (定义在 `src/tools/registry.ts`)
 - HTTP 额外 endpoint: `src/server/http.ts` 提供 `/metrics` (JSON)
- 可靠性: 部分实现
 - 重试/退避: `src/api/client.ts` 对 5xx/网络错误做指数退避
 - 限流: `src/api/client.ts` `RateLimiter` (按分钟窗口)
 - 但: 缺少 **超时/取消**, 可能出现长时间挂起 (见第 2/3 节)。
- 安全: 部分实现
 - token 不落日志: `pino redact` (`src/utils/logger.ts`)
 - HTTP 模式强制 API key: `src/index.ts` `main()` 检查 `MCP_API_KEY`
 - 但: Streamable HTTP 的 Origin 校验缺失 (PO, 见第 2 节)。

1.3 需求歧义与实现假设

1. “全员列表”是否包含 0 工时用户?
 - PRD 语义更像“list_users 的全员”都应列出 (哪怕 0)。实现选择只输出有工时的人 (`aggregated.totalHours > 0`)。
 - 这会导致“谁没填工时”无法从输出中直接看出, 管理场景会踩坑 (需求层面更偏“应包含 0”)。建议: 默认包含 0, 提供 `exclude_zero_users` 开关。
2. `principal_type=project` 的等价实现
 - PRD 建议 `principal type=project + principal id`, 实现改用 `pilot id`。这可能

是基于 PingCode API 实际能力的 mapping（合理），但需要在 tool 描述里明确“project 查询走 pilot_id”。

3. 时间分片边界（是否 end_at inclusive）

- PRD/官方描述强调会转换到日期结束时间点，但未明确 API 对 end_at 边界是否 inclusive。实现按 chunk 边界直接拼接，并用 workload_id 去重，规避了“边界重复”风险；但仍建议明确 half-open 区间策略并写测试。

2. MCP 协议合规性检查（按 MCP 实际机制审）

2.1 传输方式（stdio / http）

stdio：基本合规

- 使用 StdioServerTransport（src/index.ts），符合 MCP stdio 规范。
- 日志走 stderr：pino destination: 2（src/utils/logger.ts），符合“server 可写 stderr，但 stdout 只能是 MCP 消息”的要求。（[Model Context Protocol](#)）

HTTP（Streamable HTTP）：功能上可用，但安全与协议细节不合规（P0/P1）

- 使用 StreamableHTTPServerTransport（src/server/http.ts）是正确方向。
- P0：缺失 Origin 校验**
MCP Streamable HTTP 规范明确要求校验 Origin 以防 DNS rebinding；实现未校验，且 CORS 放开 *。（[Model Context Protocol](#)）
- P1：绑定 0.0.0.0**
规范建议本地运行绑定 localhost（SHOULD），这里固定监听 0.0.0.0（httpServer.listen(port, '0.0.0.0'）），如果用户在笔记本上跑 HTTP 模式，会把 MCP 暴露到同网段。（[Model Context Protocol](#)）
- P1：DELETE 终止 session 未处理**
规范提到 client 可能发 DELETE 结束 session，server 可返回 405 + Allow；当前会落到 404。（[Model Context Protocol](#)）
- P2：CORS allow headers 不包含 x-API-Key / MCP-Protocol-Version**
你验证支持 x-API-Key，但未加到 Access-Control-Allow-Headers，浏览器环境会失败；同理 MCP-Protocol-Version 可能需要放行。

2.2 tools 定义清晰度（名称/描述/schema/输出/错误）

工具命名

- PRD 建议 `pingcode.xxx` 命名；实现用 `list_users` / `team_work_summary` 等无命名空间。PRD 允许调整 → 可接受，但建议加 namespace 或在 README 强说明，避免与其他 MCP 工具冲突。

输入 schema：不合规（P1）

- 同一工具存在两套 schema：
 - 实际校验：Zod (`toolRegistry.callTool()` 里 `toolVersion.inputSchema.parse(args)`)
 - 对外声明：`toolDefinition.inputSchema` (JSON Schema, `ListTools` 返回给 Host)
- 多处出现声明与校验不一致：
 - `user_work_summary` / `team_work_summary`：Zod enum 包含 `type`，但 `toolDefinition` enum 不含 (`src/tools/userWorkSummary.ts` / `teamWorkSummary.ts`)。
 - `list_workloads`：Zod 支持 `filter_project_id`，`toolDefinition` 没有 (`src/tools/listWorkloads.ts`)。
- 这类不一致会让 Host/LLM “按 schema 生成参数”时无法覆盖真实能力，或未来升级时产生契约破坏。

输出结构：部分合规但工程成熟度不足

- 目前所有工具输出都塞在 `content[0].text = JSON.stringify(...)`，没有 `structuredContent` / `outputSchema`。
规范允许只返回文本，但建议同时提供 `structuredContent` 以便 Host 稳定解析。([Model Context Protocol](#))
- 错误结构：工具内部返回 `{error, code}` 很清晰；问题在于 MCP 外层没有 `isError`（见下一条）。

2.3 初始化/能力声明/版本兼容性

- 初始化：使用 SDK `Server({name,version},{capabilities:{tools:{}}})`，OK。
- 版本兼容：实现了工具版本系统 (`src/tools/versions.ts`) + builtin `get_tool_versions`，这是加分项（工程成熟度不错）。
- 但：版本系统描述里替换 `[Version: v1]` 的做法依赖字符串替换，容易被后续文案改动破坏；建议结构化字段 (`title/version`)。

2.4 JSON-RPC/消息规范（请求响应对应、id、批量/并发）

- 依赖 SDK，stdio 的 JSON-RPC framing 大概率正确。
- HTTP `parseRequestBody()` 解析失败直接 `resolve(undefined)`，可能把“非法 JSON”吞掉，导致 SDK 处理其他错误（建议返回 400：JSON RPC 不清晰）

导致 SDK 处理其他错误（建议返回 400 + JSON-RPC parse error 更清晰）。

- 并发：HTTP 模式通过 `Map<sessionId, transport>` 复用会话，但没有上限/TTL，会话泄漏风险（P1）。

2.5 失败与异常：超时、取消、重试、部分失败

- **P0：工具级“业务错误”不标 `isError`**
MCP Tools 规范明确：工具执行错误要在 result 里 `isError:true` 返回，方便 LLM 自我修复。你现在只有 unknown tool / throw 才 `isError:true`，而 NO_DATA / USER_NOT_FOUND / AMBIGUOUS_USER 等是“成功结果里包含 error 字段”，Host 很可能当成功处理。（[Model Context Protocol](#)）
- 超时/取消（cancellation）：没有任何 AbortSignal 传递到 PingCode fetch，也没有对 CancelledNotification 做显式支持；长查询/限流等待会把 Host 卡住（P0/P1）。
- 部分失败：work_item 批量获取失败会记 missingCount，但 team 汇总没把 missingCount 汇总进 data_quality（bug, P1）。

2.6 可观测性：日志/追踪是否污染 MCP 通道

- stdio：日志走 stderr，OK。（[Model Context Protocol](#)）
- HTTP：额外 `console.log()` 输出到 stdout，不影响 MCP（HTTP 模式不走 stdout），但建议也走 logger。
- 缺少 trace id / request id 的贯穿（只有少量日志字段），排障一般。

3. PingCode 集成质量（外部服务/API 视角）

3.1 鉴权安全

- 做得对
 - `PINGCODE_TOKEN` 必填校验（`src/config/index.ts` zod min(1)）。
 - 请求使用 Bearer token（`src/api/client.ts` headers `Authorization: Bearer ${token}`），符合 PRD/官方。（[open.pingcode.com](#)）
 - 日志脱敏：`src/utils/logger.ts` `redact` 包含 token/authorization。
- 不足
 - `TOKEN_MODE` 配置存在但未参与任何权限/数据策略（目前只是“摆设字段”，P2）。
 - HTTP 模式 API key 校验是“静态共享密钥”，没有过期/轮换/多 key 支持（可接受 MVP，但上线建议补）。

3.2 API 调用健壮性（分页/限流/重试/退避/网络错误）

- 分页：
 - workloads: `src/api/endpoints/workloads.ts` while 循环翻页 + `maxPages`; 能标记 `paginationTruncated`
 - users: `src/api/endpoints/users.ts` 翻页
- 限流：
 - `RateLimiter` (每分钟窗口) 能避免超过 200/min 默认限制 (PRD提到官方上限 200/min) 。(open.pingcode.com)
- 重试：
 - 5xx / 网络错误指数退避, 4xx 不重试 (方向正确)
- P0/P1 问题
 1. 无请求超时: `fetch()` 未设置 `AbortController timeout`, 遇到网络抖动可能无限挂起 (线上事故级别: Host 卡死/超时) 。
 2. 429 未按 **Retry-After**: 只按本地 `RateLimiter` 等待, 不读服务端节流头, 会导致不稳定。
 3. **listWorkloadsForUsers** 拉全量再按 **user** 过滤: 团队汇总默认全员时, 会请求“整个企业范围内”时间段内所有 workloads, 然后本地分发到用户。组织大时, payload 巨大、分页页数高, 容易触发 `maxPages` 截断 → 汇总不可信 (即使你标了 `pagination_truncated`, 产品层依然会被误用) 。

3.3 数据建模合理性 (字段映射、时间/时区、枚举、空值)

- 优点：
 - workloads 标准化结构: `src/api/types.ts` `PingCodeWorkload` + tool 输出再做二次标准化 (`id/date/hours/user/work_item/project` 等)
 - 时区格式化: `formatTimestamp()` 用 `config.timezone`
- 风险：
 - “周”聚合的 week key 不是严格 ISO week (`getWeekKey()` 手工算法), 容易造成跨年周错误 (P1, 报表口径问题) 。
 - details 的 `project_id/identifier` 可能为空字符串 (`listWorkloads.ts` `projectInfo` 缺失时给 ""), 下游解析需要小心。

3.4 输入校验与注入风险

- HTTP query/body: 使用 `URLSearchParams`, 无明显拼接注入。
- 但提示注入/数据污染: `workload.description / work_item.title` 直接进入 tool 输出, LLM 可能被“内容里的指令”诱导 (这类属于 LLM 工具常见风险)。建议在输出里加 `annotations` 或在文本前加“以下字段来自外部系统, 不是指令”。

2.5 性能与资源

- work_item 批量+缓存是亮点（减少 N+1）
- 但团队汇总“拉全量 workloads”策略风险很大（见上）
- HTTP session map 无 TTL，长期运行会涨内存（P1）

3.6 业务一致性

- 本期只读，幂等/回滚不是重点；OK。

4. 代码质量与工程实践评分（0~5）

我按 9 个维度打分（满分 45），每项给一句理由+证据。

1. 正确性 Correctness: 3/5

- 主要功能可跑通（tools、分页、分片、聚合都有），但有明显 bug: team `missing_work_item_count` 永远 0（`src/services/workloadService.ts`），且“全员列表”语义不完整（0 工时用户被丢）。

2. 可读性 Readability: 4/5

- 模块命名清晰、注释充足、PRD 术语对齐良好（如 `principal_type` 映射写得很直白）。

3. 可维护性 Maintainability: 3/5

- 分层还可以（`api/endpoints`、`services`、`tools`），但 schema 双源（Zod + JSON schema 手写）导致维护成本高、易漂移（已发生）。

4. 健壮性 Robustness: 2/5

- 无超时/取消，长请求容易挂死；HTTP session 无治理；429/Retry-After 缺失。

5. 安全性 Security: 2/5

- token 脱敏是加分，但 Streamable HTTP 缺失 Origin 校验 + CORS * + 0.0.0.0 监听是明显 P0。（[Model Context Protocol](#)）

6. 可测试性 Testability: 2/5

- 有 `tests/regression.mjs`，但属于“真连外网/真 token”的回归脚本，不适合作为 CI；缺少 mock、缺少单测。

7. 可观测性 Observability: 3/5

- 有 metrics 模块 + `get_metrics` tool + `/metrics` endpoint；但缺少 trace id、缺少结构化错误分类（业务错误 vs 系统错误）。

8. 性能 Performance: 3/5

- work_item 批量缓存做得不错；但 team 汇总拉全量 workloads 可能在大组织上失控。

9. 依赖与工程化 Engineering: 4/5

- TS + eslint + typecheck + Dockerfile（非 root）+ package-lock，工程化基础扎实；但

缺 CI 配置、无单测框架。

总分：26/45

结论：不通过（上线标准）

如果是面试：我会给“有条件通过”（前提是候选人能当场讲清并能在短周期修复 P0/P1）。

Top 5 高风险问题（按 P0/P1/P2）

- **P0-1: MCP 工具业务错误不返回 `isError: true`** (`src/index.ts` CallTool handler) → LLM/Host 无法正确纠错重试，线上表现“看似成功但内容是 error”。([Model Context Protocol](#))
- **P0-2: HTTP Streamable transport 未校验 Origin + CORS * + 0.0.0.0** (`src/server/http.ts`) → DNS rebinding/跨站调用风险，可能数据外泄。([Model Context Protocol](#))
- **P0-3: 无 timeout/取消机制** (`src/api/client.ts` fetch 无 Abort; RateLimiter 可能长等待) → Host 超时、服务线程堆积、不可控。
- **P1-1: 工具 schema 漂移 (Zod vs JSON Schema)** (`userWorkSummary.ts` / `teamWorkSummary.ts` / `listWorkloads.ts`) → 契约不稳定、升级易炸。
- **P1-2: team_work_summary “全员列表”不完整 + data_quality bug** (`src/services/workloadService.ts`) → 报表口径不可信，管理场景直接踩坑。

5. 具体可执行改进建议

5.1 最小重构计划（3~5 步，每步 0.5~1 天）

Step 1（0.5~1 天）：修复 MCP 错误语义（P0）

- 统一：只要工具返回 `{error, code}`（或其他业务错误结构），MCP result 必须 `isError:true`。
- 同时建议提供 `structuredContent`，让 Host 不用 parse text。

Step 2（0.5~1 天）：消除 schema 双源漂移（P1）

- 选一个作为“单一真源”：
 - 要么用 `zod-to-json-schema` 从 Zod 生成 JSON Schema；
 - 要么把 JSON Schema 写成常量，再由它生成 Zod（更少见）。
- 先修复现有漂移点：补齐 `group_by.type`、补齐 `filter_project_id` 等。

Step 3（1 天）：HTTP transport 安全加固（P0）

- Origin 校验（可配置 allowlist），不满足就 403。
- 默认绑定 `127.0.0.1`（除非显式配置暴露）。

- 收紧 CORS：只允许受信 Origin，或直接关闭（多数 MCP client 不需要浏览器 CORS）。

Step 4 (0.5~1 天)：API client 增强健壮性 (P0/P1)

- 加超时 (AbortController)，并把取消信号从 MCP request 透传到 fetch。
- 429 处理：尊重 Retry-After（若有）+ jitter。
- 为长查询增加“最大总耗时/最大页数/最大记录数”的硬熔断，并把原因写入 data_quality。

Step 5 (0.5~1 天)：修复 team 汇总口径 (P1)

- 默认包含 0 工时用户（让管理者看到谁没填）；新增开关可过滤 0。

5.2 关键问题示例代码/伪代码

(A) MCP tool 结果统一封装：自动识别业务错误并 `isError:true`

在 `src/index.ts` 的 CallTool handler 里引入：

```
type BusinessError = { error: string; code: string };

function isBusinessError(x: unknown): x is BusinessError {
  return !!x
    && typeof x === 'object'
    && typeof (x as any).error === 'string'
    && typeof (x as any).code === 'string';
}

function toMcpToolResult(payload: unknown) {
  // 同时返回 text + structuredContent (推荐)
  const structuredContent = payload;
  const text = typeof payload === 'string' ? payload : JSON.stringify(payload,
    null, 2);

  if (isBusinessError(payload)) {
    return {
      content: [{ type: 'text', text }],
      structuredContent,
      isError: true,
    };
  }
  return {
    content: [{ type: 'text', text }],
    structuredContent,
  };
}
```

然后把原来的 return 替换为 `return toMcpToolResult(result) / return toMcpToolResult(builtinResult.result)`。

这一步能直接解决 PRD 中 NO_DATA / USER_NOT_FOUND 等可纠错错误的“协议层语义”。

(B) Origin 校验 (HTTP Streamable transport 必做项)

在 `src/server/http.ts` 的 `createServer` 回调开头：

```
const allowedOrigins = new Set((process.env.ALLOWED_ORIGINS ??
"").split(",").filter(Boolean));

function isOriginAllowed(origin?: string) {
  if (!origin) return true; // 非浏览器/无 Origin 情况可放行
  if (allowedOrigins.size === 0) return false; // 默认拒绝, 强制显式配置
  return allowedOrigins.has(origin);
}

const origin = req.headers["origin"] as string | undefined;
if (!isOriginAllowed(origin)) {
  res.writeHead(403, { "Content-Type": "application/json" });
  res.end(JSON.stringify({ error: "Forbidden", message: "Invalid Origin" }));
  return;
}
```

并将 CORS 从 `*` 改为 `allowlist` 回显（或直接不发 CORS 头）。

(C) fetch 超时 + MCP 取消透传（避免挂死）

在 `apiClient.request()` 里加入：

```
const controller = new AbortController();
const timeout = setTimeout(() => controller.abort(), 15_000); // 15s 示例
try {
  const response = await fetch(url, { ...options, signal: controller.signal });
  ...
} finally {
  clearTimeout(timeout);
}
```

更理想：从 MCP SDK 的 handler context 获取 `signal`（如果 SDK 暴露），合并到 `AbortSignal.any([ctx.signal, controller.signal])`。

(D) team 汇总包含 0 工时用户

在 `getTeamWorkSummary()` 里不要 `if (aggregated.totalHours > 0)` 才 push，而是总是 push（或可配置）：

```
const includeZero = options.includeZeroUsers ?? true;
if (includeZero || aggregated.totalHours > 0) {
  members.push({ user, total_hours: aggregated.totalHours, ... });
}
```

```
}
```

并保证 `user_count` 语义清晰：是“返回成员数”还是“有工时成员数”。

5.3 更合理的模块划分建议

当前结构已经不错，但为减少“schema 漂移”和增强可测试性，建议：

```
src/
  mcp/
    server.ts           // stdio/http 启动、origin 校验、auth、session
    handlers.ts         // listTools/callTool + toMcpToolResult
    schemas/            // 从 zod 生成 json schema (或反过来)
  pingcode/
    client.ts           // timeout/retry/ratelimit
    endpoints/
      users.ts
      workloads.ts
      workItems.ts
    types.ts
  domain/
    workloadService.ts // 纯业务聚合，尽量不直接依赖 mcp
  observability/
    logger.ts
    metrics.ts
  cache/
```

6. 测试用例与面试追问

A) 测试清单

单元测试（至少 8 条）

1. **parseTimeRange**: `start/end` 日期字符串 → 秒级时间戳，且 end 为 endOfDay；非法输入抛 `INVALID_TIME_RANGE`。
2. **时间别名**: `上周/本月/昨天` 等别名解析正确，且时区一致。
3. **workloads** 分片: 跨度 4 个月 → 生成多个 chunk，且 `timeSliced=true`；chunk 边界无重复（或通过 `seenIds` 去重）。
4. **/v1/workloads** 分页终止条件: 模拟 `total/page_size/page_index`，能正确停止；超过 `maxPages` 设置 `paginationTruncated`。
5. **RateLimiter**: 超过阈值时会等待；窗口重置后可继续；并发多请求不会超发。
6. **work_item** 批量缓存: `mget` 命中/未命中路径，`missingCount` 正确；并发控制生效。

7. **team** 汇总包含 0 工时用户（修复后）：无工时用户仍出现在 members, total_hours=0。
8. **business error** → **MCP isError**（修复后）：工具返回 NO_DATA 时, CallToolResult.isError=true。

集成测试（至少 4 条：mock PingCode + MCP host 调用路径）

推荐用 `nock` 或 `msw` mock PingCode API；用 MCP SDK client（或子进程 stdio）做 end-to-end。

1. **stdio 模式 E2E**：启动 server 子进程 → Initialize → listTools → callTool(team_work_summary)；断言输出含 members/details。
2. **http 模式 E2E**：启动 HTTP server → POST /mcp 完整调用链；断言必须带 API key，否则 401。
3. **429 场景**：mock PingCode 返回 429 + Retry-After；断言 client 退避并最终成功或返回可解释错误。
4. **work_item enrichment**：mock workloads 返回 work_item_id；mock work_items 返回 identifier/title；断言 summary/details 中 work_item 字段被填充。

契约测试（至少 4 条：工具输入输出 schema 稳定性）

1. `listTools` 返回的 inputSchema 与实际 Zod 校验一致（自动对比 enum、required 字段）。
2. `group_by` enum 必须包含 `type`（修复后），并且文档/description 一致。
3. `list_workloads` schema 必须包含 `filter_project_id`（修复后），且传入后确实影响输出。
4. `user_work_summary/team_work_summary` 输出必须包含 `summary/details/data_quality` 且字段类型稳定（可用 JSON Schema snapshot 测试）。

B) 面试追问题（至少 10 题，分类 + 优秀回答要点 + 常见坑）

协议理解（MCP）

1. 为什么 MCP 工具的“业务错误”要用 `isError:true`，而不是在 payload 里塞 `{error:...}`？
 - 优秀要点：区分 JSON-RPC 协议错误 vs 工具执行错误；`isError` 让 LLM 可自我修正重试；Host 也能正确显示错误。（[Model Context Protocol](#)）
 - 常见坑：把所有错误都当 JSON-RPC error；或完全不标 isError，导致 LLM 继续使用错误结果。
2. **stdio transport** 为什么要求 **stdout** 只能输出 MCP 消息？日志应该去哪？
 - 优秀要点：framing/newline-delimited JSON-RPC；stdout 污染会导致 Host 解析失败；日志走 stderr。（[Model Context Protocol](#)）

- 常见坑：console.log 到 stdout；或把 JSON 输出里带换行破坏协议。

3. Streamable HTTP 下 session、MCP-Session-Id、MCP-Protocol-Version 你如何处理？

- 优秀要点：初始化返回 session id；后续请求带 header；协议版本协商与兼容默认版本；DELETE 终止会话处理。([Model Context Protocol](#))
- 常见坑：自己维护 session map 不清理；不处理协议版本头。

系统设计

1. team_work_summary 你为什么选择“拉全量 workloads 再按 user 过滤”？规模变大怎么办？

- 优秀要点：权衡请求次数 vs payload；当组织大时应改成“按 user 分批拉”或“按项目/组织维度分页拉并流式聚合”；需要限制 max records + 早停。
- 常见坑：只考虑小数据；忽略 maxPages 截断导致数据不可信。

2. 你会如何设计 tool 的 schema/versioning，避免 Zod 和 JSON schema 漂移？

- 优秀要点：单一真源、自动生成、snapshot/契约测试；版本策略 (deprecated/removalDate/migrationGuide) 。
- 常见坑：手写两份 schema；靠人工 review 保持一致。

安全

1. Streamable HTTP 为什么要校验 Origin？DNS rebinding 是什么攻击？你会怎么实现 allowlist？

- 优秀要点：浏览器可携带用户网络权限访问本地服务；Origin 校验+仅绑定 localhost+鉴权；默认拒绝策略。([Model Context Protocol](#))
- 常见坑：CORS * 当“方便调试”；不理解本地 MCP 的威胁模型。

2. 日志脱敏你怎么做？有哪些“看起来不是 secret 但也敏感”的字段？

- 优秀要点：token/authorization 必脱敏；项目名/工作项标题/描述可能含敏感信息；errorBody 也可能含 PII；分级日志。
- 常见坑：只脱敏 token，忽略 errorBody 与业务字段。

稳定性

1. PingCode API 429/5xx 你如何区分处理？退避策略怎么避免放大请求？

- 优秀要点：429 尊重 Retry-After + 全局限流；5xx 指数退避+抖动；最大重试次数/最大总耗时；区分幂等 GET。
- 常见坑：所有错误都重试；无 jitter 导致惊群。

2. 你会如何做“超时 + 取消 (cancellation) ”的端到端传递？

- 优秀要点：MCP request context signal → fetch AbortController；及时释放资源；返回 isError 的可解释错误。
- 常见坑：只在 client 侧超时，server 继续跑；导致资源泄漏。

测试与发布

1. 为什么你现在的 `regression.mjs` 不适合 CI? 你会怎么改成可重复的自动化测试?

- 优秀要点: 外部依赖/token/数据不稳定; 应 mock PingCode; 引入单测框架; 契约测试+E2E 通过容器化 mock server。
- 常见坑: 把“真连外网脚本”当测试体系。

1. 上线后如何监控“数据截断”(`pagination_truncated`)和“`time_sliced`”对准确性的影响?

- 优秀要点: metrics 维度化; 报警 (截断率、分片次数激增); 在输出里强提示; 必要时拒绝返回不可信数据。
- 常见坑: 只记录总请求量, 不记录质量标志。

如果你希望我进一步“更狠一点”做上线门禁, 我会额外要求:

- 修完 P0 后补一套最小 CI (lint/typecheck/unit tests)
- 以及一个真实 MCP Host (例如 Claude Desktop/Cursor) 回放用的契约用例, 确保 `listTools` schema 与行为一致。