Tools & Registry

Tool inventory and usage guides

- HiveMind Conventions
- Voice Channel — Google Home TTS Integration
- MCP Setup — Model Context Protocol
- ac-grind.js — $0 Local-Model Code Grinder (per-finding)

HiveMind Conventions

HiveMind Convention

Odobreno: Alem, 2026-02-06 **Updated:** 2026-02-12 (Edita archived)

Arhitektura

```
SHARED BUS (svi agenti čitaju/pišu)
  ~/system/agents/hivemind/hivemind.db

CROSS-SESSION TASKS (Alem vidi)
  ~/system/databases/mission-control.db

PER-CLIENT DATA (izolirano po projektu)
  ~/projects/<klijent>/client.db
```

Hijerarhija

Alem → John (direktno)

- Edita arhivirana 2026-02-12 (backup: ~/system/archive/edita-backup-2026-02-12/)
- John radi direktno sa Alemom
- Subagent teamovi (Builder, Validator) po potrebi

HiveMind Type Konvencija

agent	type	Značenje
john	task	John loguje task
john	response	John odgovara
john	update	John javlja status
john	discovery	John pronašao korisnu informaciju

agent	type	Značenje
builder	task-update	Builder javlja napredak na tasku
validator	validation	Validator javlja rezultat provjere
*	alert	Hitno — treba pažnja odmah
*	learning	Naučeno nešto novo
*	error	Nešto puklo

Historijski tipovi (Edita, archived): task-update, question, response ostaju u bazi za referencu.

Komande

```
# John loguje task
node ~/system/agents/hivemind/hivemind.js post john task \
  "Opis taska" '{"client":"ime","priority":"high"}'

# John javlja update
node ~/system/agents/hivemind/hivemind.js post john update \
  "Task XY: zavrseno" '{"client":"ime","status":"done"}'

# Builder javlja napredak
node ~/system/agents/hivemind/hivemind.js post builder task-update \
  "Implementation progress" '{"task_id":"123","status":"in_progress"}'

# Validator javlja rezultat
node ~/system/agents/hivemind/hivemind.js post validator validation \
  "Validation passed" '{"task_id":"123","result":"pass"}'

# Čitaj sve
node ~/system/agents/hivemind/hivemind.js read 10

# Čitaj samo od jednog agenta
node ~/system/agents/hivemind/hivemind.js read john 10

# Pretraži
node ~/system/agents/hivemind/hivemind.js query "fitlife"
```

Per-Client DB Pattern

Svaki klijentski projekat ima svoju bazu:

```
~/projects/<klijent>/
├── CLAUDE.md          ← Pravila za taj projekat
├── client.db          ← Klijent-specifični podaci (SQLite)
└── src/               ← Kod
```

Pravilo: Klijentski podaci NIKAD u HiveMind. HiveMind je samo za komunikaciju i koordinaciju između agenata.

Data Field (JSON)

Svaki post može imati `data` JSON polje za strukturirane podatke:

```
{
  "client": "fitlife",
  "priority": "high|medium|low",
  "status": "pending|in_progress|done|blocked",
  "deadline": "2026-02-07",
  "blocked": true,
  "files": ["src/index.html"],
  "ref_task_id": 73
}
```

Primjer Workflow

```
1. John: post john task "Implement landing page for FitLife" {client:fitlife, priority:high, mc_task_id:123}
2. Builder: post builder task-update "FitLife: started" {client:fitlife, status:in_progress, mc_task_id:123}
3. Builder: post builder task-update "FitLife: done" {client:fitlife, status:done, mc_task_id:123}
4. Validator: post validator validation "FitLife: PASS - all criteria met" {client:fitlife, result:pass, mc_task_id:123}
5. John: post john update "FitLife: deployed to production" {client:fitlife, mc_task_id:123}
```

ACK Protocol

Kad primiš novu instrukciju, javi ACK:

```
node hivemind.js post <agent> response "ACK: <kratki opis>"
```

Voice Channel — Google Home TTS Integration

Sta je #voice kanal?

Slack kanal **#voice** omogucava razgovor sa John-om koji se **izgovara na Google Home zvucniku** u kuhinji (Nest Mini) ili dnevnom boravku (Nest Audio).

Kako radi

1. Otvori Slack na telefonu
2. Udji u **#voice** kanal
3. Napisi poruku (ili koristi voice typing na tastaturi)
4. John odgovori tekstom na Slack + izgovori odgovor na zvucniku

Ogranicenja

- John NE CUJE mikrofon — Cast protocol je samo output
- Input je SAMO tekst iz Slacka
- Koristi voice typing na telefonu za hands-free diktat

Glasovi

Jezik	Glas	Detekcija
Bosanski	bs-BA-GoranNeural	ccszd karakteri
Norveski	nb-NO-FinnNeural	aeo karakteri
Engleski	en-US-GuyNeural	engleske rijeci

Uredjaji

Kljuc	Uredjaj	IP
kitchen	Nest Mini (Kjokken)	192.168.68.57
living	Nest Audio	192.168.68.54

Tehnicki detalji

- **Bot:** slack-bot.js (Socket Mode) — detektuje poruke iz #voice kanala
- **AI:** comms-responder.js — VOICE MODE prompt (bez formatiranja, kratko, konverzacijski)
- **TTS:** edge-tts (Microsoft Neural Voices) → MP3 → lokalni HTTP server
- **Cast:** google-home.js → castv2-client → Google Home uredjaj na LAN-u
- **Volume:** 80% default

CLI

```
node ~/system/tools/google-home.js say "tekst" --device kitchen --lang bs
node ~/system/tools/google-home.js say "tekst" --device living --volume 1.0
node ~/system/tools/google-home.js devices
node ~/system/tools/google-home.js discover
```

MCP Setup — Model Context Protocol

MCP Setup — Model Context Protocol

Posljednje ažuriranje: 2026-03-06 **Analiza:** John (2 agenta, ~130K tokena)

Šta je MCP

Model Context Protocol (MCP) omogućava Claude Code-u da direktno poziva servise kao native alate (`mcp__server__tool`), bez potrebe za CLI spawnom ili Bash komandama. Rezultat: brže izvršavanje, manje kontekstnog zagađenja, native integracija.

Ekosistem: 1200+ servera na registry.modelcontextprotocol.io (mart 2026).

Trenutna Konfiguracija (`~/.claude/mcp.json`)

9 Aktivnih MCP Servera

Server	Tip	Komanda	Svrha
email	Custom	`node email-mcp-bridge-v2.js`	IMAP/SMTP za 5 accounta (john, info, alai, alem, dev)
rag	Custom	`node rag-mcp.js`	RAG cache + Ollama routing (10K+ entries, 61% hit rate)
figma	Custom wrapper	`figma-mcp-wrapper.sh`	Figma API + Vaultwarden credential injection

Server	Tip	Komanda	Svrha
youtube-transcript	Community	`npx @fabriqa.ai/youtube-transcript-mcp`	YouTube transcript ekstrakcija
github	Official binary	`github-mcp-server stdio`	GitHub PR/issues/repos/workflows (v0.31.0, Homebrew)
slack	Community	`npx mcp-server-slack`	Slack read/send (workspace: alai-talk, T0AELHU0E13)
postgres-drop	Community	`npx postgres-mcp`	SQL nad Drop dev DB (port 5433)
postgres-plock	Community	`npx postgres-mcp`	SQL nad Plock dev DB (port 5432)
fiken	Custom (John)	`node fiken-mcp.js`	11 Fiken alata — invoices, contacts, balances, dashboard

Uklonjen (2026-03-06): `playwright` — zamijenjen CLI-jem (po Alemovom zahtjevu).

Dostupni Alati po Serveru

mcp__email__*

- `emails_find` — pretraga emailova
- `email_send` — slanje
- `email_respond` — reply/forward
- `emails_modify` — mark/flag/archive
- `folders_list` — lista foldera

mcp__rag__*

- `rag_query` — semantic search (cache + Ollama fallback)
- `rag_learn` — dodavanje Q&A parova
- `rag_stats` — statistike cache-a

mcp__github__*

- `create_or_update_file`, `create_repository`, `create_branch`
- `get_file_contents`, `list_commits`, `list_branches`
- `create_pull_request`, `merge_pull_request`, `get_pull_request`

- `create_issue`, `list_issues`, `update_issue`, `add_issue_comment`
- `search_repositories`, `search_code`, `search_issues`
- `create_review`, `request_reviewers`

mcp__slack__*

- `list_channels`, `post_message`, `reply_to_thread`
- `get_channel_history`, `get_thread_replies`
- `search_messages`, `add_reaction`

mcp__postgres-drop__* / mcp__postgres-plock__*

- `query` — SQL query
- `list_tables` — lista tabela
- `describe_table` — schema tabele

mcp__fiken__*

- `fiken_status` — API health check
- `fiken_dashboard` — CEO finansijski pregled
- `fiken_invoices_list` — lista faktura
- `fiken_invoices_show` — detalji fakture
- `fiken_invoices_sync` — sync → local DB
- `fiken_contacts_list` — lista kontakata
- `fiken_contacts_show` — detalji kontakta
- `fiken_contacts_sync` — sync → local DB
- `fiken_balances` — bankovni salda
- `fiken_transactions_list` — transakcije (--days N)
- `fiken_reconcile` — auto-match transakcija (--dry-run)

Arhitektura — 3-Tier Model

```
Tier 1: MCP Tools (Claude Code native)
  └── mcp__email__*, mcp__rag__*, mcp__github__*, mcp__slack__*, mcp__postgres-*__*,
mcp__fiken__*

Tier 2: CLI Fallback (daemons i agenti bez Claude Code)
  └── mail-native.js, rag-router.js, slack.js, fiken.js
```

```
Tier 3: Orchestration Layer
  └── email-mcp-bridge-v2.js, rag-mcp.js, fiken-mcp.js (wrappuju Tier 2)
```

Pravilo: MCP u Claude Code sesijama. CLI u daemon/cron/background procesima.

Credentials

Server	Metod	Lokacija
email	ENV var u mcp.json	direktno
github	ENV var `GITHUB_PERSONAL_ACCESS_TOKEN`	mcp.json
slack	ENV var `SLACK_BOT_TOKEN`	mcp.json
figma	Vaultwarden injection	figma-mcp-wrapper.sh
fiken	Vaultwarden (live fetch)	fiken.js → vault.js
postgres	Connection URL	mcp.json args

MCP Roadmap — Interni Build Plan

Analiza: 12 kritičnih business toolsa | **Ukupan effort:** ~10.5h | **Ušteda:** 60+ h/god

Faza 1 — Foundation (5h, ODMAH)

MCP Server	Baziran na	Effort	ROI	Zašto first
mc-mcp	mc.js (117K, 200+ ops/week)	2h	□□□□□□	Svaka sesija; task automation backbone
hivemind-mcp	hivemind.js (10K+ entries, 50+ ops/week)	2h	□□□□□□	Memory = kognitivna osnova; 80% brže
ceo-briefing-mcp	ceo-briefing.js (1-2x/sesiji)	1h	□□□□	Quick win; parameterizable intelligence

Faza 2 — Revenue & Ops (5.5h)

MCP Server	Baziran na	Effort	ROI	Impact

sales-pipeline-mcp	sales-pipeline.js + leads.db	2.5h	□□□	Deal updates during client conversations
invoice-mcp	invoice-generator.js (48K) + invoices.db	3h	□□□	Weekly workflow; VAT logic complexity
contacts-mcp	contacts.js + contacts.db (100+ kontakata)	1.5h	□□□	Fast CRM search bez CLI spawna

Faza 3 — Enablers (opcionalno, inkrement.)

MCP Server	Effort	Napomena
contract-mcp	2h	CRUD + renewal automation
proposal-mcp	2.5h	PDF generation za klijente
email-briefing-mcp	1h	ZAKON #17 — unification
signing-mcp	1.5h	Documenso one-command flow
tenders-mcp	2h	On-demand tender search
facts-mcp	1h	ZAKON #16 — decision persistence

Productivity Impact

Tool	CLI ops/week	Ušteda/op	Sedmično
mc.js	200+	450ms	25+ min
hivemind.js	50+	370ms	15+ min
sales-pipeline.js	30+	410ms	12 min
invoice-generator.js	20+	540ms	10 min
UKUPNO	**300+**	—	**~1h/sedmično**

Godišnje: 60+ sati za Alema (30% Claude Code sesija)

Community Alternativa: NEMA

Nijedan community MCP ne pokriva ALAI-specifičnu integraciju (branding, ZAKON enforcement, decision logging, memory model). Sve custom.

Što SE NE?E buildati

MCP Server	Razlog

bookstack-mcp	bookstack-sync.js + CLI dovoljan; rijetko se koristi live
sanity-mcp	Tek kad product CMS dođe u produkciju
vercel-mcp	Nema official MCP; gh CLI dovoljan
calendar-mcp	Baikal CalDAV, nizak prioritet

Community Serveri koji NISU dodani (evaluirani, rejected)

Server	Razlog odbijanja
`@modelcontextprotocol/server-github`	DEPRECATED — zamijenjen official `github-mcp-server`
`@modelcontextprotocol/server-postgres`	DEPRECATED — zamijenjen `postgres-mcp`
`@korotovsky/slack-mcp-server`	Nije na npm; `mcp-server-slack` dovoljan
Kubernetes MCP	Nema Kubernetes infrastrukture trenutno
AWS CloudWatch MCP	Nema AWS deployova trenutno
Notion MCP	Ne koristimo Notion

Fajlovi

- **MCP Config:** `~/.claude/mcp.json`
- **Email MCP:** `~/system/tools/email-mcp-bridge-v2.js`
- **RAG MCP:** `~/system/tools/rag-mcp.js`
- **Fiken MCP:** `~/system/tools/fiken-mcp.js` ← novi
- **Figma wrapper:** `~/system/tools/figma-mcp-wrapper.sh`
- **GitHub binary:** `/opt/homebrew/bin/github-mcp-server` (v0.31.0)

ac-grind.js — $0 Local-Model Code Grinder (per-finding)

What it is

A wrapper around **aider** + the `autocoder.js` differential-tsc-gate logic that fixes **one finding/file per pass** using a **local $0 model** (Ollama `qwen2.5-coder:32b-instruct-q8_0` @ 10.0.0.2:11434, or MLX qwen3-coder-30b @ 11437). Implements the measured recipe from `/tmp/evidence-mlx-autocoder-fix/verification.md`.

Tool: `~/system/tools/ac-grind.js` · **MC:** #103050 · **Status:** proven (2026-06-06)

The recipe (7 levers)

1. **Scope = 1 finding/file per pass** (broad batches choke the local model).
2. **Auto-attach relevant DEF files as read-only aider context** — biggest accuracy lever (measured 5→1). Resolves the target's `@/…` imports to real files via tsconfig paths.
3. **Qwen sampling**: temp 0.7 / top_p 0.8 / top_k 20 / rep_penalty 1.05.
4. **Repo-map (tree-sitter) + diff edits** (whole-file fallback for small files on aider 0.86.x).
5. **DIFFERENTIAL touched-file-only tsc gate** — repo is ~615 errors RED at baseline, so only NEW errors in the touched file count.
6. **Reflection cap ≥ 6** (aider 0.86.x caps internal loop at 3 → wrapper does ceil(N/3) outer passes, stops on tsc-delta stall).
7. **E2E browser gate** (vite + Playwright + MSAL fixture) as the real DONE signal, not tsc-only.

Usage

```
node ~/system/tools/ac-grind.js --file <abs path> [--desc "what to fix"]
  [--engine ollama|mlx] [--max-reflections 6]
```

Proof (MC #103050, EVVRecordsList.tsx, LumisCare)

- 8 DEF files auto-attached.
- **5 → 0 differential tsc errors** in one aider pass (independently confirmed: `npx tsc -b` clean on fixed file).
- Diff validated vs ground-truth APIs: `PaginationIndicator` (real export, props page/totalPages/rowCount/pageSize/onPageChange), `Badge size="sm"` (valid cva variant). Original `Pagination currentPage=` + `size={16}` were genuine type errors.
- **E2E Playwright PASS** (results.json expected:1 unexpected:0) + real screenshot — component mounts in a real browser without crash.
- $0 (local model), no deploy, proof edit local-only & reverted.

Proven scope / honest limits

- Reliable for **mechanical type-error findings** (prop/API mismatches) where DEF files carry ground-truth types.
- NOT proven for **open-ended feature generation** (model must invent new logic).
- E2E gate currently proves **mount-not-crash**, not full data render — backend must run to prove real rows.

Hardening & real-batch results

Guard A — file-size guard

`DEFAULT_MAX_LINES = 450` (calibrated: proven-good 397 lines < 450 < proven-bad 585 lines). Files over the threshold emit a loud `SKIPPED_TOO_LARGE` box to stderr with exit code 2 — before any model call is made, so no tokens are wasted. CLI flags: `--max-lines N` to override, `--force-large` as escape hatch.

- EVVRecordDetail.tsx (586 lines) → exit 2, SKIPPED_TOO_LARGE (correctly blocked).
- QualificationsTab.tsx (397 lines) → allowed by default, flows to aider (proven-good path confirmed).

Guard B — artifact lint

Diff-scoped JSX-comment and whitespace-expression detection runs after every aider edit. Detects patterns such as `{" "}` whitespace expressions, `{/* narrative comments */}`, and inline narrating comments on JSX tags. On detection, ac-grind auto-strips and re-validates; if the stripped result still fails, it reverts the file and exits with `ARTIFACT_REJECTED` (exit 3). Legitimate `{" "}` (e.g. dash separators) are preserved because detection is diff-scoped, not file-scoped. Never ships dirty output.

- Pattern unit test: 12/12 PASS.

- Integration test (MC #103058 stash artifacts): 5 artifact lines detected, auto-stripped cleanly, legitimate separator preserved.

Real backlog sweep (MC #103063)

Ran across the backoffice module of LumisCare. 15 files edited by ac-grind sweep; 13 kept (clean error reduction), 4 reverted (quote-style-only changes, zero error reduction).

Metric	Value
Files edited by sweep	15
Kept clean (real error reduction)	13
Reverted (quote-style only, no improvement)	4
Baseline tsc errors (origin/dev)	612
After sweep	578
Net reduction	−34 errors (5.5% of backlog)
New errors introduced	0

Shipped as PR #55 to `dev` (branch `ac-grind-sweep-103063`, commit `b17f606`), pending Proveo review. Not merged.

Honest takeaway: ac-grind reliably clears mechanical prop/API type-errors on files at or below 450 lines. Structural errors (e.g. EVVRecordsList.tsx — 5 errors untouched) and over-threshold files still require manual work.

Guard evidence

- `/tmp/evidence-103061/verification.md` — Guard A + B hardening + threshold recalibration to 450
- `/tmp/evidence-103063/verification.md` — real-batch sweep triage table + PR

Evidence

- `/tmp/evidence-103050/verification.md` (+ John independent verification section)
- `/tmp/verify-103050/evidence/` (build-tsc, test-e2e, playwright-results.json, screenshot)
- Recipe origin: `/tmp/evidence-mlx-autocoder-fix/verification.md`
- Related memo: `project_local_model_coding_setup_2026-06-06.md`