Technology Research

Tech evaluations, framework comparisons, stack decisions.

Overview

Technology Research Overview

Tech evaluations, framework comparisons, and stack decisions.

Owner: John **Last Verified:** 2026-02-17

Contents

To be populated from technology research documents

GOTCHA Framework Deep Dive

GOTCHA Framework — Complete Reference

> Last Verified: 2026-02-17 | Owner: John

GOTCHA is the 6-layer architecture for agentic systems. This is the FOUNDATION of how John operates.

Overview

GOTCHA is a 6-layer framework that separates concerns between what needs to happen (Goals), who coordinates it (Orchestration), what executes it (Tools), what informs decisions (Context), what guides behavior (Hard prompts), and what shapes responses (Args).

GOT (Engine):

- **Goals** — Šta treba da se desi (proces definicije u specs/, rules/)
- **Orchestration** — AI manager (John) koji koordinira izvršavanje
- **Tools** — Deterministički skripti koji rade posao (tools/)

CHA (Context):

- **Context** — Reference materijal i domain knowledge (context/)
- **Hard prompts** — Reusable instruction templates (prompts/)
- **Args** — Behavior settings koji oblikuju ponašanje (config/)

Principle: 90%^5 = 59%

AI greši kumulativno. Ako je svaki korak 90% tačan:

- 1 korak: 90%

- 2 koraka: 81%
- 3 koraka: 73%
- 4 koraka: 66%
- **5 koraka: 59%**

Zato:

- **Pouzdanost** → deterministički kod (tools)
- **Fleksibilnost** → LLM (John)
- **Proces** → goals/specs
- **Znanje** → context/memory

Layer 1: GOALS (specs/ + rules/)

Purpose: Define WHAT needs to happen, not HOW.

Components:

- `~/system/specs/` — Implementation plans, feature specs
- `~/system/rules/` — Process standards, lessons learned
- Mission Control tasks — Active work queue

John's role: Read specs before implementing. Follow rules during execution. Check MC for priorities.

Layer 2: ORCHESTRATION (John)

Purpose: AI manager that coordinates execution.

Responsibilities:

- Read goals/specs to understand requirements
- Choose appropriate tools from manifest
- Apply context from memory/files
- Delegate to sub-agents when appropriate
- Handle errors and adapt

Key principle: John sits between "what" (goals) and "how" (tools). Reads instructions, applies context, delegates well, handles failures.

Layer 3: TOOLS (tools/)

Purpose: Deterministic execution. Do ONE thing reliably.

Components:

- `~/system/tools/` — All system tools (150+ scripts)
- `~/system/tools/manifest.md` — Tool registry (CHECK FIRST before writing new tools)

Rules:

1. ALWAYS check manifest before creating new tool
2. One tool = one responsibility
3. Zero deps when possible
4. Exit codes: 0=success, 1=error, 2=retry
5. JSON output for programmatic use

Example tools:

- `mc.js` — Mission Control task management
- `figma-extract.js` — Figma REST API wrapper
- `sales-pipeline.js` — CRM operations
- `invoice-generator.js` — Invoice creation + reminders

Layer 4: CONTEXT (context/ + memory/)

Purpose: Domain knowledge and reference material.

Components:

- `~/system/context/` — 506 reference files
- `~/system/agents/hivemind/` — SQLite knowledge base (8561 entries)
- `MEMORY.md` — Key learnings and laws
- `SESSION-STATE.md` — Current session context

Usage:

- Read context files before making decisions
- Query HiveMind for past lessons: `node ~/system/agents/hivemind/hivemind.js query "keyword"`
- Post new learnings: `node ~/system/agents/hivemind/hivemind.js post john knowledge "lesson"`

Layer 5: HARD PROMPTS (prompts/)

Purpose: Reusable instruction templates.

Components:

- `~/system/prompts/` — 3 templates
- Anti-hallucination rules
- Problem-solving frameworks

Use cases:

- Builder agent instructions
- Validator agent instructions
- Research agent templates

Layer 6: ARGS (config/)

Purpose: Behavior settings and configurations.

Components:

- `~/system/config/` — 76 config files
- Environment variables
- API keys and credentials
- Feature flags

Boot Sequence

Every session MUST start with:

```
bash ~/system/boot.sh
```

Boot verificira svih 6 GOTCHA layera:

1. GOALS — specs/ + rules/ exist
2. ORCHESTRATION — agents defined
3. TOOLS — manifest.md accessible
4. CONTEXT — context/ + HiveMind operational

5. HARD PROMPTS — prompts/ available
6. ARGS — config/ accessible

Output: Status report + recent changelog + task summary + email briefing + checklist reminders

---

Enforcement: Hooks

GOTCHA is enforced via `~/.claude/hooks/` (Python scripts):

Hook	Layer	What It Blocks
gotcha-enforcer.py	GOALS	Write/Edit without GOTCHA checklist
security-guard.py	TOOLS	Access to forbidden paths
hallucination-detector.py	CONTEXT	Phantom dependencies, fabricated APIs
plan-enforcer.py	ORCHESTRATION	Implementation without approved plan
agent-protocol-enforcer.py	ORCHESTRATION	Agent spawns without CORE PROTOCOL

GOTCHA vs AIOS

History: System was originally called AIOS (AI Operating System). On 2026-02-06, all AIOS references were removed from active files. Only GOTCHA remains.

Why: GOTCHA is the framework. AIOS was branding. System needs framework, not branding.

Backup: Original working system preserved at `~/clawd/` as reference.

Tool-First Protocol

GOTCHA Layer 3 (Tools) has a search order:

1. **Naši alati** (`~/system/tools/manifest.md`) — CHECK FIRST
2. **Naši skillovi** (`~/.claude/skills/`) — Use existing skills
3. **Naša baza** (HiveMind) — Query for past solutions
4. **Internet** (WebSearch/WebFetch) — Research if no internal solution
5. **Ažuriraj bazu** — Post new learnings to HiveMind

Rule: NEVER write new code before checking manifest. NEVER research externally before checking HiveMind.

For full system handbook, see `~/system/CLAUDE.md`

Figma Design Pipeline

Figma Design Pipeline — Complete Workflow

> Last Verified: 2026-02-17 | Owner: John

This is the PROVEN pipeline for going from design brief to deployed code using Figma as the central hub.

The 7-Step Pipeline

```
BRIEF → STITCH → FIGMA → EXTRACT → ASSEMBLE → VALIDATE → DEPLOY
  ↓        ↓        ↓        ↓          ↓          ↓         ↓
 Spec    Generate  Import   Tokens     React      Compare    Ship
         (FREE)   (manual)  (auto)     (auto)     (visual)
```

Step 1: BRIEF

Parse requirements into structured spec.

Input: Client requirements, competitor analysis, user stories **Output:** Design brief with: target users, key features, visual style, brand guidelines **Tool:** Manual or agent-generated

Step 2: STITCH

Generate 3 design variants using Google Stitch (FREE).

Tool: `stitch-generate.js` **Command:**

```
node ~/system/tools/stitch-generate.js --brief "Drop" --screen "dashboard" --industry "fintech" --primary "#0B6E35" --secondary "#D4A017" --model pro --options 3
```

Output: 3 HTML/CSS variants **Cost:** $0 (Gemini 2.5 Pro is free)

Step 3: FIGMA

Import HTML into Figma for refinement.

Method: Use `html.to.design` plugin (80-90% accuracy) or Copy to Figma **Manual step:** Refine layout, adjust spacing, apply design system tokens **Output:** Figma file with frames ready for export

Step 4: EXTRACT

Pull design tokens and component data via Figma REST API.

Tool: `figma-extract.js` **Commands:**

```
# Extract design tokens
node ~/system/tools/figma-extract.js extract-tokens FILE_KEY

# Extract components
node ~/system/tools/figma-extract.js extract-components FILE_KEY

# Export frame as image
node ~/system/tools/figma-extract.js export-image FILE_KEY NODE_ID --format png --scale 2 --
output /tmp/frame.png
```

Output: JSON tokens, component metadata, reference images

Step 5: ASSEMBLE

Convert Figma frames to React + Tailwind code.

Tool: `figma-to-react.js` **Command:**

```
node ~/system/tools/figma-to-react.js FILE_KEY NODE_ID --output Login.tsx
```

Features:

- Auto Layout → Flexbox (VERTICAL/HORIZONTAL, gap, padding)
- Semantic HTML (h1-h6, p, span)
- Color conversion (RGBA → hex, Tailwind arbitrary values)
- Typography mapping (font family/weight/size/lineHeight)
- Effects (shadows → shadow-sm/md/lg/xl/2xl)

- Border radius presets
- Zero deps, rate limiting with exponential backoff

Output: Valid JSX/TSX with proper nesting

Step 6: VALIDATE

Visual comparison: built page vs Figma design.

Tool: `figma-validate.js` **Command:**

```
node ~/system/tools/figma-validate.js compare FILE_KEY NODE_ID http://localhost:3000/login --output /tmp/validate/
```

Features:

- Pixel-by-pixel comparison using sharp
- Configurable threshold (default 10%)
- Diff report with highlighted regions
- Exit codes: 0=PASS, 1=FAIL, 2=ERROR
- Viewport customization

Output: Markdown report + diff image with red overlay for differences

CRITICAL: Enforces ZAKON #0.1 — "pogledati ≠ vidjeti". Lists DIFFERENCES, not similarities.

Step 7: DEPLOY

Docker → Fly.io / Vercel

Vercel:

```
vercel --prod
```

Fly.io:

```
flyctl deploy
```

Figma REST API Quick Reference

Authentication

```
curl -H "X-Figma-Token: YOUR_TOKEN" https://api.figma.com/v1/files/FILE_KEY
```

Note: Personal Access Tokens max 90 days (2025 change)

Key Endpoints

Endpoint	Method	What It Does
`/v1/files/{key}`	GET	Full file data (nodes, styles, components)
`/v1/files/{key}/nodes?ids=X,Y`	GET	Specific nodes only
`/v1/images/{key}?ids=X&format=png&scale=2`	GET	Export as image
`/v1/files/{key}/variables/local`	GET	All design variables
`/v1/files/{key}/components`	GET	All components
`/v1/files/{key}/styles`	GET	All styles

Export Formats

Format	Scale	Notes
PNG	0.01x–4x	Max 32 megapixels, DPI = 72 × scale
JPG	0.01x–4x	Max 32 megapixels
SVG	1x only	Options: outline text, include IDs, simplify strokes
PDF	1x only	Vector output

URLs expire after 30 days.

Rate Limits (Leaky Bucket)

Tier	Endpoints	Professional
1	Files, exports	12/min
2	Variables, components	30/min
3	Writes	30/min

Best practices: Batch IDs (comma-separated), cache aggressively, use webhooks not polling.

Design Tokens — Three-Tier Architecture

MANDATORY structure:

```
PRIMITIVE (raw values, named by appearance)
  blue-50: #E3F2FD
  blue-500: #2196F3
  spacing-4: 4px
  spacing-16: 16px

SEMANTIC (purpose-based, references primitive)
  color-primary: → blue-500
  color-background: → blue-50
  spacing-component-padding: → spacing-16
  text-body: → font-size-16

COMPONENT (scoped, references semantic)
  button-background-primary: → color-primary
  button-padding: → spacing-component-padding
  card-border-radius: → radius-md
```

Variable Types

| Type | Use For | Example | |------|---------|---------|| | COLOR | All colors | `#0B6E35`, `rgba(11,110,53,1)` | | NUMBER | Spacing, sizing, opacity | `16`, `1.5`, `8` | | STRING | Font families, labels | `'Inter'`, `'DM Sans'` | | BOOLEAN | Feature flags | `true`, `false` | | ALIAS | References to other variables | `→ blue-500` |

Figma ? Code Mapping

Auto Layout ? CSS Flexbox

Figma	CSS
Horizontal	`flex-direction: row`
Vertical	`flex-direction: column`

Figma	CSS
Spacing between	`gap: Xpx`
Space between mode	`justify-content: space-between`
Hug contents	`width/height: auto`
Fill container	`flex: 1`
Fixed	Absolute pixel value
Padding	`padding: T R B L`
Align top	`align-items: flex-start`
Align center	`align-items: center`

Component Variants ? React Props

```
Figma: Button / variant=primary, size=large, disabled=false
React: <Button variant="primary" size="lg" disabled={false} />
```

Tools Status

Production Ready

Tool	Coverage	Status
`figma-extract.js`	80% token extraction	□ PROD (653 lines, zero deps)
`figma-to-react.js`	95% Figma→React+Tailwind	□ PROD (580 lines, zero deps)
`figma-validate.js`	100% visual validation	□ PROD (500+ lines, sharp+playwright)
`design-to-code.js`	85% HTML→TSX conversion	□ PROD (600+ Tailwind mappings)
`stitch-generate.js`	70% Stitch automation	⚠ Works but brittle selectors

For complete Figma knowledge base, see `~/system/context/figma-knowledge-base.md`

Mission Control — Task Management System

Mission Control — Complete Reference

> Last Verified: 2026-02-17 | Owner: John

Mission Control (MC) is the active task management system. It replaced Taskwarrior and enforces GOTCHA compliance via hooks.

Overview

Status: PRIMARY task system (replaced Taskwarrior) **Backend:** SQLite (`~/system/databases/mission-control.db`) **Frontend:** Web dashboard at http://localhost:3030 **Enforcement:** gotcha-enforcer.py reads `/tmp/mc-active-task` + DB **Daemons:** com.john.mc-dashboard + com.john.mc-session-worker

---

CLI Commands

Basic Operations

```
# List all open tasks
node ~/system/tools/mc.js list

# List my tasks only
node ~/system/tools/mc.js list --owner john

# Add new task
```

```
node ~/system/tools/mc.js add "Title" --desc "Description" --priority H --owner john

# Start task (unlocks Write/Edit)
node ~/system/tools/mc.js start <id>

# Complete task with outcome summary
node ~/system/tools/mc.js done <id> "outcome"

# Pause task (blocks Write/Edit)
node ~/system/tools/mc.js pause <id>

# Resume paused task
node ~/system/tools/mc.js resume <id>

# Block task with reason
node ~/system/tools/mc.js block <id> "reason"

# Assign to owner
node ~/system/tools/mc.js assign <id> <owner>

# Show full details
node ~/system/tools/mc.js show <id>

# Show audit trail
node ~/system/tools/mc.js history <id>

# Who's working on what
node ~/system/tools/mc.js active

# Summary counts
node ~/system/tools/mc.js stats
```

Backward Compatibility

```
# Old task.sh wrapper (proxies to mc.js)
~/system/tools/task.sh list
~/system/tools/task.sh add "Title"
~/system/tools/task.sh start <id>
~/system/tools/task.sh done <id>
```

Dashboard (CEO UI)

URL: http://localhost:3030 **Features:**

- CRUD operations
- Pause/resume tasks
- Assign ownership
- Priority updates
- Mobile-friendly
- Auto-refresh
- LAN accessible

LaunchAgent: com.john.mc-dashboard (auto-start on boot)

Enforcement — GOTCHA Integration

How It Works

1. **Start task:** `mc.js start <id>` creates `/tmp/mc-active-task` with task ID
2. **Hook reads:** `gotcha-enforcer.py` checks `/tmp/mc-active-task` on every Write/Edit/Bash
3. **Validates:** Hook queries `mission-control.db` to verify task exists and is active
4. **Blocks:** If no active task → BLOCKED. Must `mc.js start` first.
5. **Done:** `mc.js done` removes `/tmp/mc-active-task`, blocks further edits until next start

GOTCHA Checklist Requirement

Rule: BEFORE any Write/Edit/Bash, you must have `/tmp/gotcha-task-{id}.md` with 6 sections:

```
# GOTCHA Checklist — MC Task #{id}

## G — Goal
Šta tačno radim? Koji spec/kriterij?

## O — Options
Koje opcije imam? Koju biram i zašto?

## T — Tools
Koji alati? Jesam li provjerio manifest?

## C — Context
```

```
Šta sam pročitao/verificirao prije?

## H — Hazards
Šta može poći po krivu? Backup plan?

## A — Acceptance
Kako ću znati da je gotovo?
```

Enforcement: Hook blocks Write/Edit until checklist exists.

Rules

1. **Svaki task od Alema → ODMAH u MC**
   - `mc.js add "Title" --desc "X" --priority H --owner john`
2. **Prije rada →** `mc.js start <id>`
   - Kreira `/tmp/mc-active-task`
   - Unlocks Write/Edit/Bash
3. **Kad završiš →** `mc.js done <id> "outcome"`
   - Removes active task flag
   - Blocks further edits
   - Records outcome in DB
4. **Kraj sesije →** `mc.js list`
   - Check remaining open tasks
   - Create follow-up tasks if needed

Session Execution Tools

These are ephemeral — die when session ends:

```
// Create task (visible in Claude Code UI only)
TaskCreate({
  subject: "Task title",
  description: "Full description",
  activeForm: "focus" | "backlog"
})

// Update task status
TaskUpdate({
  taskId: "task_xxx",
```

```
  status: "completed" | "cancelled"
})
```

Use cases:

- Quick sub-tasks during implementation
- Temporary tracking within session
- NOT for persistence — use MC for that

Database Schema

File: `~/system/databases/mission-control.db`

Tables:

- `tasks` — Core task data (id, title, description, status, priority, owner, created_at, updated_at)
- `task_history` — Audit trail (task_id, action, actor, timestamp, metadata)
- `task_assignments` — Assignment tracking

Status values: `open`, `active`, `paused`, `blocked`, `done`, `cancelled`

Priority values: `L` (low), `M` (medium), `H` (high)

Owner values: `john`, `alem`, `edita`, `—` (unassigned)

Integration with Other Systems

Event Bus

MC emits events on task changes:

- `task.created` — New task added
- `task.status_changed` — Status updated
- `task.assigned` — Owner changed
- `task.completed` — Task marked done

Subscribers: pipeline-watcher, autowork daemon, session-worker

HiveMind

Completed tasks logged to HiveMind:

```
node ~/system/agents/hivemind/hivemind.js post john success "MC #<id>: <outcome>"
```

Migration from Taskwarrior

Status: Taskwarrior still installed but NOT primary **Source of truth:** MC DB **Old commands:** `task list` → use `mc.js list` **Data:** Not migrated. MC started fresh.

For CLI implementation, see `~/system/tools/mc.js` (2000+ lines, SQLite + better-sqlite3)

Petter Graff — Software Architect

Knowledge base: articles, whitepapers, case studies, and architectural principles from Petter Graff (CTO Pratexo, edge computing & distributed systems expert)