.claude/ Subagent files and admin dashboard

This document contains the generated contents for the `.claude` folder: one `.md` file for each subagent, a shared `status.md` (single-source-of-truth for agents' statuses), a `LOG_FORMAT.md` describing recommended logging fields, and an `admin.html` file that reads the logs and shows the team's last statuses. Also included: Team mantra, motivation, and a high-level team structure & purpose document.

> **Note:** This document stores the latest generated contents. The intention is for you to copy each section into separate files under a real `.claude/` folder in your repo (e.g., `.claude/web-development.md`, `.claude/status.md`, `.claude/admin.html`). The admin.html assumes logs are accessible as JSON files in a `logs/` folder (one latest-status JSON per agent) — sample layout described in `LOG_FORMAT.md`.

File list (to create from this doc)

- `.claude/web-development.md`
- `.claude/java-backend.md`
- `.claude/ui-designer.md`
- `.claude/database-master.md`
- `.claude/test-master.md`
- `.claude/architecture-master.md`
- `.claude/flutter-dart.md`
- `.claude/ci-cd-master.md`
- `.claude/legal-advisor.md`
- `.claude/game-theory.md`
- `.claude/civil-engineer.md`
- `.claude/psychologist-game-dynamics.md`
- `.claude/project-manager.md`
- `.claude/scrum-master.md`
- `.claude/status.md` *(single-source-of-truth, machine-friendly markdown/JSON block inside)*
- `.claude/LOG_FORMAT.md` *(describes logs and the JSON each agent writes)*
- `admin.html` *(dashboard reading logs/ folder; shows last status only)*
- `.claude/MANTRA.md` *(team mantra & motivation)*
- `.claude/TEAM_STRUCTURE.md`

How to use

1. Create a `.claude/` folder in your project root.
2. Copy each `*.md` content block from this file into separate files as listed above.
3. Implement a simple agent runner (or adapt your subagent platform) so that each agent writes a single JSON file `logs/<agent-id>.json` containing *only its last status* when it finishes/updates. The dashboard reads those JSON files and renders the single-line status for each agent.

Agent template (use this as a header inside every agent .md)

```
# Agent: <Name> (id: <short-id>)
Role: <one-line role summary>
Owner: <team or person>
Objective: <single-sentence goal>

## Runtime contract
- Input: { task: string, contextFiles: [paths], config: {} }
- Output: writes `logs/<short-id>.json` with `LATEST_STATUS` JSON (see
LOG_FORMAT.md)

## Logging responsibilities
- Always write the latest state to `logs/<short-id>.json`.
- Append detailed session logs to `logs/archive/<short-id>-<timestamp>.log` if
needed.
- Notify `project-manager` when blocked or when disagreeing with decisions.

## Decision rules
- Use the `status.md` single-source-of-truth to set and read statuses.
- When starting work, set status => `IN_PROGRESS` after writing pre-work entry.
- When blocked, set status => `BLOCKED` and include `blockedBy` field with
pointers.
- When finished, set status => `DONE` with `percentComplete: 100`.
```

LOG_FORMAT.md

This is the standard JSON structure **each agent writes to** `logs/<agent-id>.json`. The admin dashboard reads these files and shows only the latest status for each agent.

```
{
  "agentId": "web-dev",
  "displayName": "Web Development",
  "status": "IDLE|IN_PROGRESS|BLOCKED|REVIEW|DONE|DISAGREE",
  "taskId": "optional-task-guid",
  "taskTitle": "Short task title",
  "percentComplete": 0,
  "importance": "low|medium|high|critical",
  "why": "brief reason for current status (1-2 lines)",
  "when": "2025-10-10T12:34:56Z",
  "blockedBy": ["agent-id or file path or external error"],
  "interactions": [
    {"with":"project-manager","when":"...","note":"short summary"}
  ],
  "disagreement": {
    "with":"agent-id or decision-id",
    "why":"short 1-2 line explanation",
    "severity":"low|medium|high"
  },
  "logsLink": "logs/archive/web-dev-2025-10-10T12-34-56.log"
}
```

Rules: - Each file must contain only a single JSON object (no arrays) and always represent the **last** status. - Timestamps must be ISO-8601 UTC. - `percentComplete` is integer 0–100. - Dashboard trusts these files as the single source for UI state.

---

status.md (single-source-of-truth)

This file is a human-readable index and *mirrors* the same data as the logs but in a concise table. It is **not** the canonical machine source: the `logs/<agent-id>.json` files are canonical. Use this file for quick human scan and CI checks.

```
# AGENTS STATUS - last updated: 2025-10-10T00:00:00Z

| agentId | displayName | status | percentComplete | importance | lastUpdated |
|---------|-------------|--------|-----------------|------------|-------------|
| web-dev | Web Development | IDLE | 0 | medium | 2025-10-10T00:00:00Z |
| java-backend | Java Backend | IDLE | 0 | critical | 2025-10-10T00:00:00Z |
... (one line per agent)
```

(Automations may rewrite this when agents change state — but always keep machine-readable copies in `logs/`.)

Agent files

.claude/web-development.md

Use the `Agent template` header (above). Key responsibilities: implement front-end components, own web app structure, accessibility, performance budgets, and integration with backend API schema. Tools: React/Next/Vite, Tailwind (if desired), storybook, playwright for E2E coordination with `test-master`.

Logging specifics

- `importance` default: `high`
- Frequent `percentComplete` updates at milestone boundaries (10%, 25%, 50%, 75%, 100%)

When to set `DISAGREE`

- If backend API contracts would cause breaking UX and backend refuses to adapt.

.claude/java-backend.md

Responsibilities: define APIs, own domain model, performance and observability, own integration with databases and CI/CD pipeline.

Logging specifics: - `importance`: `critical` - `blockedBy` commonly includes DB migrations, infra, or missing API specs from `architecture-master`.

Decision rules: - Use semantic versioning for API changes and notify `project-manager` and `ui-designer` on incompatible changes.

.claude/ui-designer.md

Responsibilities: visual system, component specs, accessibility, copy tone. Provide final Storybook and Figma export snapshots to be used by `web-development` and `flutter-dart` agents.

Logging specifics: - Provide `designSnapshotLink` in `interactions` when handing off.

.claude/database-master.md

Responsibilities: schema design, migrations, query performance, backups, data contracts.

Logging specifics: - `importance`: `critical` for migration windows. - Provide `migrationId` in `interactions` when coordinating.

.claude/test-master.md

Responsibilities: own unit/integration/e2e, test coverage thresholds, flaky test tracking, and test pipelines.

Logging specifics: - Add `testCoverage` field in `interactions` when reporting. - When blocked, include failing test ids in `blockedBy`.

.claude/architecture-master.md

Responsibilities: system-level decisions, tradeoffs, performance budget, security/privacy boundaries.

Logging specifics: - `importance`: `critical` - Provide `decisionId` and a short rationale when changing architecture.

.claude/flutter-dart.md

Responsibilities: mobile & web flutter implementation, share design system with web, implement platform-specific optimizations.

Logging specifics: - Provide `flutterSdk` and `targetPlatforms` in `interactions` when starting.

.claude/ci-cd-master.md

Responsibilities: pipelines, environment gating, canary and rollback strategies, artifact promotion.

Logging specifics: - Provide `pipelineId`, `currentRunId`, and `status` details in `interactions`.

.claude/legal-advisor.md

Responsibilities: licensing, compliance, data protection (GDPR), contract reviews.

Logging specifics: - `importance`: `high` for contractual tasks. - When blocked by unclear product decisions, set `DISAGREE` and provide `legalRiskLevel`.

.claude/game-theory.md

Responsibilities: design incentive structures, token economies, UX incentives, anti-abuse.

Logging specifics: - Provide `gameModel` short description and `simulationResultsLink` if any.

.claude/civil-engineer.md

Responsibilities: provide inputs for physical constraints (e.g., if the product has hardware or space concerns), load calculations or structural recommendations.

Logging specifics: - Include `safetyCritical` boolean when applicable.

.claude/psychologist-game-dynamics.md

Responsibilities: player/user behavior, engagement loops, addiction risk analysis, fairness and ethics of game dynamics.

Logging specifics: - `ethicsRisk` field when relevant; suggest interventions.

.claude/project-manager.md

Responsibilities: own the schedule, risk register, stakeholder comms, and orchestration. All agents must append interactions with project-manager to their `interactions` array when they communicate.

Logging specifics: - Provide `nextMilestone` in `interactions` payload.

.claude/scrum-master.md

Responsibilities: collect agent feedback, run retros, resolve cross-agent disagreements, keep the status file coherent.

Logging specifics: - `retrosSummaryLink` can be provided in `interactions`.

---

admin.html (dashboard)

Below is a self-contained `admin.html` that reads `logs/` JSON files and renders a small dashboard showing each agent's last status. It expects the project to serve `logs/` as static JSON (one file per agent). The code is intentionally simple so you can extend it.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Agents — Admin Dashboard</title>
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <style>
    body{font-family:Inter,system-ui,Arial;margin:20px}
    .grid{display:grid;grid-template-columns:repeat(auto-fit,minmax(280px,
1fr));gap:12px}
```

```
    .card{border-radius:10px;padding:12px;border:1px solid #e0e0e0;box-shadow:0
2px 6px rgba(0,0,0,0.03)}
    .status-IDLE{background:#fff}
    .status-IN_PROGRESS{background:#fffbe6}
    .status-BLOCKED{background:#ffe6e6}
    .status-DONE{background:#e6ffed}
    .status-DISAGREE{background:#fff0f5}
  </style>
</head>
<body>
  <h1>Team — Last Status Dashboard</h1>
  <p>Shows each agent's *last* status (single-file per agent in /logs).</p>
  <div id="grid" class="grid"></div>
  <script>
    async function load() {
      try {
        const indexResp = await fetch('logs/index.json');
        const index = await indexResp.json();
        const grid = document.getElementById('grid');
        for(const entry of index.agents){
          try{
            const r = await fetch('logs/' + entry.agentId + '.json');
            const s = await r.json();
            const card = document.createElement('div');
            card.className = 'card status-' + (s.status || 'IDLE');
            card.innerHTML = `
              <h3>${s.displayName} <small>(${s.agentId})</small></h3>
              <p><strong>Status:</strong> ${s.status} — <strong>$
{s.percentComplete}%</strong></p>
              <p><strong>Task:</strong> ${s.taskTitle || '—'}</p>
              <p><strong>Importance:</strong> ${s.importance || 'medium'}</p>
              <p><em>${s.why || ''}</em></p>
              <p style="font-size:0.8em;color:#666">Last: ${s.when}</p>
            `;
            grid.appendChild(card);
          }catch(e){
            console.error('failed to load agent', entry.agentId, e);
          }
        }
      }catch(e){
        console.error('failed to load index', e);
        document.getElementById('grid').innerHTML = '<p>Unable to load logs/
index.json — serve a JSON index with an `agents` array.</p>';
      }
    }
    load();
  </script>
```

```
</body>
</html>
```

Notes: - `logs/index.json` is a small manifest file your infra updates when new agents appear. Example:

```
{ "agents": [ { "agentId": "web-dev" }, { "agentId":"java-backend" } ] }
```

- The dashboard intentionally reads a single JSON file per agent and shows only the latest state. Archive logs are kept in `logs/archive/` and not shown by default.

.claude/MANTRA.md

```
We build boldly, precisely, and ethically.

Our mission: Turn ideas into working software fast — without sacrificing craft.

We believe in radical ownership, tiny fast feedback loops, and compassionate
disagreement.

Daily motivation: ‘Ship clarity, not chaos.’

This team’s north star: deliver meaningful, measurable customer value every
sprint.
```

TEAM_STRUCTURE.md

- **Purpose:** Convert high-level product ideas into production-grade software via specialized subagents that handle implementation, review, and risk.
- **Dynamics:** Agents propose, implement, report. `project-manager` coordinates; `scrum-master` collects disagreements; `architecture-master` and `legal-advisor` veto or approve critical changes.
- **Ambition:** Build a resilient, auditable, and observable automation-first engineering organization where each subagent is accountable.

Quick-start checklist

1. Create `.claude/` directory and copy files.
2. Add a `logs/` folder with `index.json` and one sample `web-dev.json` matching LOG_FORMAT.

3. Serve the repo with a static server (or adjust `admin.html` to your backend) and open `admin.html`.
4. Hook agents to write `logs/<agent-id>.json` on start/change/finish.

End of generated content.