Phase 1:

You are my senior full stack engineer and product designer.

New Project: OpenClaw Web UI Control Center (Local Mission Control)
Goal: Build a polished, local hosted “mission control” dashboard that helps me manage all my OpenClaw projects, tasks, sub agents, automations, and output folders in one place.

Important context
- This is NOT replacing the existing OpenClaw Web UI. This is an optional second dashboard focused on operating multiple projects at once.
- I run many projects in parallel and I need strict separation by project.
- This must be hosted locally (LAN only by default). It can run on Mac or Linux, and optionally on a VPS later, but build for local first.
- It must integrate with my OpenClaw configuration and filesystem outputs. It should link to output folders and open files quickly.

Primary user
- Me (operator) and optionally a small internal team
- I need quick visibility, fast triage, and clear next actions across projects.

High level outcomes
1) One dashboard to view all projects and their status at a glance
2) Create and manage projects, tasks, and sub agents with clear workflows
3) Track active tasks, queued tasks, and completed tasks with logs and artifacts
4) Built in web chat that can talk to an OpenClaw agent in the context of a selected project
5) A cron and scheduler area for recurring jobs (view, create, edit, run now, enable/disable)
6) Strong file and output management: link to folders, browse artifacts, search, and open in OS file explorer
7) Clean UI, fast, reliable, with an operational feel (mission control)

Non goals
- Do not attempt to rebuild OpenClaw’s entire UI or add features that already exist there.
- Do not add cloud dependencies. Local first. Offline friendly.
- No heavy enterprise auth. Keep it simple and secure for local use.

Tech requirements (choose a modern but simple stack)
- Frontend: React + TypeScript + Tailwind
- Backend: Node.js (Express or Fastify) + TypeScript
- DB: SQLite (via Prisma or Drizzle). Local file stored under a project data directory.
- Real time: WebSocket for task status updates and chat streaming.
- Packaging: Docker Compose optional, but must also run via npm scripts locally.

- OS integration: Provide “Open Folder” and “Reveal in Finder” style actions (Mac) and equivalent for Linux. If direct opening is not possible from browser, implement a backend endpoint that triggers OS open via shell command with strong path validation.

Integration requirements
- OpenClaw config file lives at: ~/.openclaw/openclaw.json
- The Control Center must be able to read that config (read only by default).
- It should detect agents defined in config and show them as available “sub agents”.
- It should support a per project mapping to:
  - which agent(s) are used
  - working directory root
  - output folder root
  - environment variables or secrets references (do not store secrets in plain text)
- It must show running tasks and completed tasks, and point to their output artifacts on disk.

Core features (must build)
A) Projects
- Create new project wizard
  - Name, description, tags
  - Project root directory path
  - Output directory path
  - Default agent or agent group (from openclaw.json)
  - Optional Git repo link (just metadata)
  - Operations cadence (daily, weekly) metadata
- Project detail page
  - Overview: status, last run, open tasks, next scheduled run
  - Tasks board: Backlog, Planned, In Progress, Blocked, Done
  - Agents: assigned agents, agent notes, quick actions
  - Files: output folders, artifacts, recent files
  - Activity timeline: task events, scheduler runs, chat summaries

B) Tasks
- Task types
  1. Plan task (research, PRD, roadmap)
  2. Build task (feature build, bug fix, implementation)
  3. Ops task (monitoring, outreach, reporting, recurring maintenance)
- Task fields
  - Title, description, type, priority, status, due date
  - Project, assignee agent, dependencies
  - Links: docs, URLs
  - Output folder pointer and expected deliverables checklist
- Task execution
  - “Run task” button that triggers an OpenClaw run via a backend command integration (see integration section below)

  - Store logs, start and end timestamps, exit status
  - Capture produced artifacts list (scan output folder before and after run, then diff)

C) Agents and sub agents
- Agents index page
  - Show all agents discovered from ~/.openclaw/openclaw.json
  - Show which projects use which agents
  - Notes and capabilities fields (editable in Control Center DB, not written back to openclaw.json unless explicitly enabled later)
- Per project “agent loadout”
  - Define which agent is primary and which are sub agents
  - Quick “chat as agent” from the project context

D) Scheduler (cron job setup area)
- Scheduler page
  - List jobs: name, project, schedule (cron expression), enabled, last run, next run
  - Create job wizard
    - Select project
    - Select job type: run task, run agent prompt, run maintenance script
    - Cron expression builder plus raw input
    - “Run now” button
  - Execution log history for each job
- Implementation detail
  - Use a Node scheduler (like node-cron) that is always on while backend is running
  - Save schedules to SQLite, reload on startup
  - Add safe guards to prevent overlapping runs unless explicitly allowed
  - Provide timezone handling and show timezone clearly in UI

E) Web chat (per project)
- Chat UI inside each project
  - Select agent
  - Chat messages stream in real time
  - Include context badges: project, selected task (optional), selected files (optional)
  - Provide “Send output to task notes” and “Create task from chat”
- Minimal chat memory
  - Store chat sessions per project in SQLite
  - Provide searchable chat history
  - Allow exporting a chat session to a markdown file in the project output directory

F) Files and outputs
- Files tab per project
  - Show output directory tree
  - Search filenames and basic text search for small files
  - “Open folder” action

  - “Copy path” action
  - “Attach file to task” and “Link file to task”
- Artifact indexing
  - Background indexing process (local) that updates recent files list and simple metadata
  - Must not read secrets or scan outside configured directories

G) Global Mission Control dashboard
- Home screen widgets
  - Active tasks across all projects
  - Next scheduled runs
  - Recent task completions
  - Projects with blockers
  - Quick create: project, task, scheduler job
- A command palette (Ctrl K)
  - Jump to project
  - Create task
  - Run task
  - Open output folder

Integration details (how to run OpenClaw)
- Implement an adapter layer with 2 modes

  Mode 1: Stub mode (for early development)
  - Running a task simulates progress and writes a fake log

  Mode 2: Real OpenClaw integration
  - Provide a backend setting where I specify the OpenClaw CLI command and working directory
  - The system runs a child process, streams stdout and stderr to the UI, saves logs to disk and DB
  - Store per project run configuration: cwd, env var references, output folder
- Safety
  - Validate all paths against allowlisted project roots
  - Do not allow arbitrary shell commands from the UI
  - Only allow running predefined templates and known OpenClaw commands
  - Provide an “admin settings” page to manage allowlists

Security and privacy
- Local only by default
  - Bind backend to 127.0.0.1
  - Add an optional basic auth toggle for LAN
- Secrets
  - Do not store API keys in SQLite in plain text
  - Provide a way to reference environment variables
  - Provide a settings page that shows which env vars are required per project

UX and design requirements
- Polished mission control style UI
- Fast navigation, minimal clutter
- Clear project separation
- Consistent components, good spacing, modern look
- Dark mode optional but preferred

Deliverables required from you (agent)
1) A product spec
  - User stories
  - Screens list
  - Navigation map
  - Data model
  - Permissions model (even if simple)
  - Edge cases

2) A technical design doc
  - Architecture diagram (described in text)
  - API routes list
  - DB schema (tables and key fields)
  - WebSocket events
  - Scheduler design
  - File indexing approach
  - OpenClaw integration adapter approach
  - Security considerations and path validation strategy

3) Implementation plan
  - Milestones (at least 5)
  - Each milestone includes specific tickets
  - Identify what can be stubbed first
  - Identify what needs real integration later

4) A working MVP implementation
  - Repo structure
  - Backend server with SQLite
  - Frontend dashboard with routing
  - Projects CRUD
  - Tasks board CRUD
  - Scheduler CRUD with node-cron
  - Chat UI with stub streaming
  - File browser for output directories
  - WebSocket live updates for task runs
  - Basic settings page for OpenClaw config path and allowlists

Acceptance criteria
- I can create a project in under 60 seconds
- I can create tasks and move them across statuses with drag and drop
- I can click “Run task” and see streaming logs and status updates
- I can create a scheduled job and it runs at the next schedule
- I can open the output folder for a project from the UI
- I can chat with an agent in a project context and save the chat to a task note
- The app runs locally with a single command and persists data across restarts

Constraints and quality bar
- Prefer simple, reliable choices over flashy ones
- Strong input validation throughout
- Handle errors with helpful messages
- No em dashes anywhere in UI text, logs, or docs
- Provide meaningful sample data for demo

Now do the following, in order
Step 1: Ask any critical questions only if truly blocking. Otherwise assume sensible defaults.
Step 2: Produce the product spec and technical design doc.
Step 3: Produce the implementation plan with milestones and tickets.
Step 4: Generate the initial codebase with clear instructions to run locally.
Step 5: List immediate next improvements after MVP.

Output format
- Use clear headings
- Use code blocks for schema, routes, and commands
- Provide copy paste run commands
- Keep docs and code consistent

Start now.

Phase 2: Wiring back end (100 hours to build this)

You are my senior full stack engineer and systems architect.

Project: OpenClaw Mission Control
Current Status: The app is already live and running at http://localhost:3001/

Important:
- This is NOT a greenfield build.
- Do NOT rebuild the project from scratch.

- We are refactoring, wiring, and upgrading the existing live app.
- Many UI sections currently show “Coming Soon”.
- Buttons exist but are not wired to backend logic.
- We must turn this into a fully functional Mission Control.

Primary Goal
Modernize the UI to feel futuristic and then wire every function end to end so that:
- Projects
- Tasks
- Agents
- Task runs
- Scheduler / cron jobs
- Logs
- Files
- Web chat
- Streaming updates

…are all real and integrated with OpenClaw.

This must operate locally and integrate with:
~/.openclaw/openclaw.json

==
PHASE 1: CODEBASE AUDIT
==

Before changing anything:

1. Inspect the running app architecture:
   - Frontend framework
   - Backend framework
   - DB layer
   - WebSocket implementation
   - Routing structure
   - Current schema

2. Identify:
   - Which UI components are placeholders
   - Which buttons lack onClick handlers
   - Which handlers exist but lack backend endpoints
   - Which endpoints exist but do not persist
   - Which backend services are stubs
   - Where “Coming Soon” is hardcoded

3. Produce:
   - A wiring gap report
   - A list mapping UI actions → missing backend logic
   - A list mapping backend endpoints → missing DB writes
   - A list mapping OpenClaw integration points → not implemented

Do NOT skip this audit.

==
PHASE 2: WIRING PLAN
==

Create a full wiring map:

For every major feature, define:

UI Action
→ API Route
→ DB Mutation
→ OpenClaw Adapter Action
→ WebSocket Event
→ UI State Update

Cover:

1) Projects CRUD
2) Tasks CRUD + drag and drop status changes
3) Run Task button
4) Agent discovery from ~/.openclaw/openclaw.json
5) Assign agents to project
6) Scheduler create / enable / disable / run now
7) Task run log streaming
8) Web chat streaming
9) File browser + open folder action

Nothing may remain unconnected.

==
PHASE 3: OPENCLAW ADAPTER (REAL)
==

Implement a backend OpenClaw adapter layer.

Requirements:

- Read ~/.openclaw/openclaw.json
- Discover available agents
- Allow per project configuration:
  cwd
  output directory
  selected agent
  environment references

- Execute OpenClaw via child_process
- Stream stdout/stderr via WebSocket
- Persist logs to disk
- Persist run records to DB
- Support cancellation
- Strict path validation
- No arbitrary command execution

All execution must:
- Be allowlisted
- Validate project root boundaries
- Bind backend to 127.0.0.1

==
PHASE 4: SCHEDULER SYSTEM
==

Implement real cron functionality.

- node-cron or equivalent
- Jobs stored in SQLite
- Reload on server restart
- Prevent overlapping runs per project by default
- Record:
  job started
  job completed
  job failed
- WebSocket updates for scheduler runs

UI must allow:
- Create job
- Edit cron expression
- Enable/disable
- Run now
- View history

- See next run time

==
PHASE 5: WEB CHAT (REAL)
==

Implement per-project chat.

- Select agent
- Stream responses live
- Store chat sessions
- Allow:
    Save to task notes
    Create task from chat
    Export to markdown in project output folder

If OpenClaw chat endpoint exists, use it.
If not, create a ChatAdapter interface that uses OpenClaw CLI mode.

No “Coming Soon”.
Chat must function.

==
PHASE 6: FILES SYSTEM
==

Implement per-project file browser.

- Tree view
- Filename search
- Preview small text files
- Copy path
- Open folder via backend command
- Link file to task

Must validate all paths.

==
PHASE 7: FUTURISTIC UI MODERNIZATION
==

Upgrade visual design without breaking wiring.

Requirements:

- Dark mode default
- Subtle glow accents
- Glass panels
- Clean typography scale
- Animated state transitions
- Real time indicators
- Command palette (Ctrl K) wired
- No clutter
- No em dashes anywhere

Do not introduce performance issues.

UI must feel like:
“AI Operations Command Center”

==
DATABASE REQUIREMENTS
==

Ensure schema supports:

Projects
Tasks
TaskRuns
Agents
ProjectAgents
SchedulerJobs
SchedulerRuns
ChatSessions
ChatMessages
Settings
Artifacts

Provide final schema.

==
ACCEPTANCE CRITERIA
==

Every button works.
Every section is functional.
No placeholder text remains.
Task runs stream logs live.

Scheduler executes and logs.
Agents load from openclaw.json.
Chat streams and saves.
Files browser works.
App persists across restart.
Dashboard shows real data.

==
OUTPUT FORMAT
==

1) Audit report
2) Wiring map
3) Updated architecture
4) Final DB schema
5) Milestone plan (at least 6)
6) Implementation tasks
7) Smoke test checklist
8) UI modernization spec

Do not rebuild from scratch.
Upgrade and wire the existing live system.

Start with the audit. Respond back with an estimate of cost and token usage.