6

COURSE 6 · CAPSTONE
INSTRUCTOR CHEAT SHEET

Full-Stack AI-Assisted Development

Eight build cycles · one architecture · one shipped container. Deck ↔ editor, ten times. Architecture diagram is the spine.

8 hr · 10 modules + assessment
57 slides · 2 breaks · 1 deploy
Companion to **week-6-fullstack.html**

0:00	0:30	☐ 1:15	☐ 2:00	2:45	☐ 3:00	☐ 3:45	☐ 4:30	5:15	☐ 5:30	☐ 6:15	☐ 7:00	7:45
M1 Frontier	M2 Setup	M3 Backend	M4 Data	**Break 15′**	M5 Frontend	M6 Pages	M7 AI Chat	**Break 15′**	M8 Auth	M9 Integ.	M10 Deploy	Wrap

1 30 MIN	**The Full-Stack Frontier** `0:00–0:30` FRAMING · NO CODE	*No editor prompts. Show the empty four-layer architecture diagram and tell the room: "Mark this slide. We see it eight more times today."*	ARCHITECTURE 0/4 **Empty runway** L1, L2, L3, L4 all dim. The diagram is anchored.
2 45 MIN	**Environment Setup** `0:30–1:15` AI-GUIDED INSTALL	1. **Install prompt (paste verbatim):** "I need to set up a development environment on [Windows / Mac]. Install Go 1.22+, Node.js 20 LTS, Git, VS Code and Docker Desktop. Step-by-step for my OS, including verify. After install I should run `go version`, `node --version`, `npm --version`, `git --version` and see numbers for all four." 2. **Scaffold** in terminal: `mkdir my-staff-app && cd my-staff-app` → `go mod init my-staff-app` → `mkdir -p cmd/server internal data` → `npm create vite@latest web -- --template react-ts`.	ARCHITECTURE 0/4 **Runway built — no layers lit** Tooling verified, project skeleton on disk. Stack still empty.
3 45 MIN	**Backend from Scratch** `1:15–2:00` BUILD · FIRST SERVER	1. **Prompt 1:** single-file Go server in `cmd/server/main.go`, port 8080, `-port` and `-dev` flags, `/api/v1/health` returning `{"status":"ok"}`. 2. **Prompt 2:** extract `SetupRouter` into `internal/api/router.go` with `writeJSON` / `writeError` helpers. 3. **Prompt 3:** add hardcoded `/api/v1/items` with five military-flavored items. 4. *Verify after each:* `go run ./cmd/server -dev` → hit endpoint in browser. Paste any error back — do not hand-debug.	ARCHITECTURE 1/4 · L2 LIT **Go HTTP Server** Server, router, helpers, two endpoints returning JSON.
4 45 MIN	**Data Layer** `2:00–2:45` BUILD · INTERFACE-FIRST	1. **Prompt 1:** define a `DataStore` Go interface and an `Item` struct in `internal/data/store.go` (list, get, create, update). Plus a `JSONStore` implementation reading `data/items.json`. 2. **Prompt 2:** generate 20 realistic, military-flavored seed items into `data/items.json`. 3. **Prompt 3:** rewire `main.go` to load the store and serve from it; add `/api/v1/items/{id}`. 4. *Verify:* `curl /api/v1/items` and `/api/v1/items/3`.	ARCHITECTURE 2/4 · L3 LIT **Data Layer (behind a contract)** JSONStore satisfies the interface. Same door, swappable engine.
BRK 15 MIN	**Break 1.** Be back at H+3:00. Pre-flight: confirm everyone's backend still runs; have the next prompt loaded so you paste the moment you return.		`2:45–3:00`
5 45 MIN	**Frontend from Scratch** `3:00–3:45` BUILD · FIRST END-TO-END	1. **Prompt 1:** add Tailwind CSS to the existing Vite + React + TS app in `web/`; configure PostCSS. 2. **Prompt 2:** set up a Vite dev proxy in `web/vite.config.ts` that forwards `/api` to `http://localhost:8080`. 3. **Prompt 3:** write a small API client and render the items table at `localhost:5173`. 4. *Two terminals:* backend in one, `npm run dev` in the other. CORS error → fix is the proxy, not middleware.	ARCHITECTURE 3/4 · L1 LIT **React Frontend — full stack, end to end** First time the app is visible to a human. Stop. Have one student share their screen.
6 45 MIN	**Pages & Navigation** `3:45–4:30` BUILD · MULTI-PAGE UI	1. **Prompt 1:** add `react-router-dom` with a Layout, Dashboard, Items, Settings; dark sidebar (navy `#1a1f36`) with active-link highlighting. 2. **Prompt 2:** item detail page at `/items/:id`; make titles in the items table click through. 3. **Prompt 3:** dashboard with stat cards, recent items, quick actions — using API data, Tailwind defaults.	ARCHITECTURE 3/4 · L1 FLESHED OUT **Same diagram, real-app shape** Sidebar nav, dashboard, detail page. The prototype look is gone.
7 45 MIN	**AI Chat Integration** `4:30–5:15` BUILD · TOOL USE	1. **Pre-flight:** confirm `OPENAI_API_KEY` is set in every shell. 2. **Prompt 1:** chat service in `internal/ai/chat.go` against `gpt-4o`; expose `POST /api/v1/chat`. 3. **Prompt 2:** register a `lookup_items` tool with status / priority / query params; call into `DataStore`. 4. **Prompt 3:** `ChatPage.tsx` with markdown rendering and loading states. 5. *Land it:* ask "what are my high-priority items?" — show the tool call in the server log.	ARCHITECTURE 4/4 · L4 LIT **External Services — the brain comes online** All four layers real. Apex of the day. Pause and let it land.
BRK 15 MIN	**Break 2.** Be back at H+5:30. Pre-flight: confirm chat still works; have the auth-middleware prompt loaded.		`5:15–5:30`
8 45 MIN	**Auth & Middleware** `5:30–6:15` BUILD · GATE-GUARD	1. **Prompt 1:** middleware package — CORS, security headers, auth, chain helper. 2. **Prompt 2:** `/auth/me` + `/auth/switch` endpoints; role picker dropdown in the header (Admin / Staff / User) with a role cookie. 3. **Prompt 3:** role-based filtering across existing handlers (Admin sees all, User sees only their items); conditional sidebar. 4. *Demo:* dev tools → cookies → show role; switch role → data changes.	ARCHITECTURE 4/4 · HARDENED **All four layers, role-aware** Same diagram. Application now treats different humans differently.
9 45 MIN	**External Integrations** `6:15–7:00` BUILD · COMPRESS HERE IF BEHIND	1. **Path A — default (no Azure needed):** add a SQLite-backed `DataStore` implementation using `modernc.org/sqlite` (pure Go); auto-create tables; parameterized queries; `-db` flag toggles JSON vs SQLite. 2. **Path B — Azure AD shop:** Microsoft Graph client (OAuth2 client-credentials, token cache, commercial & GCC High); add `/calendar/today` and `/mail/summary`. 3. *Verify (A):* restart with `-db sqlite`, add an item, restart again, confirm it persists; flip back to `-db json`.	ARCHITECTURE 4/4 · DEEPENED **L3 (Path A) or L4 (Path B) deepened** Engine swapped behind the same interface — the Module 4 lever pays off.
10 45 MIN	**Docker & Deployment** `7:00–7:45` BUILD · SHIP IT	1. **Prompt 1:** generate a multi-stage `Dockerfile` — web-builder (Node), go-builder (Go), runtime (Alpine). Go server serves the React build in production. SPA fallback for non-API routes. Include a `.dockerignore`. 2. **Build:** `docker build -t my-staff-app .` 3. **Run:** `docker run -p 8080:8080 -e OPENAI_API_KEY=… my-staff-app` → open `localhost:8080`. 4. *Optional:* push to ACR; deploy to Azure Container Apps for a public URL.	ARCHITECTURE 4/4 · SHIPPED **All four layers, packaged** One image. Same bytes, any machine. The third victory of the day.
A 15 MIN	**Assessment & Wrap-Up** `7:45–8:00` SELF-ASSESS + HANDOFF	*Walk the rubric on page 2; students self-check, take the knowledge check on the student page, then generate the certificate. Update the frontier map. Close the program.*	ARCHITECTURE 4/4 · DONE **Capstone shipped** Container running, rubric met. Then the program close.

Rubric, principles, recovery

The rubric you grade against in the last 15 minutes, the principles you repeat all day, and the recovery cues for when something breaks live.

Use during: assessment block
Use during: any stuck moment
Companion to **week-6-fullstack.html**

ASSESSMENT RUBRIC

Hit the Minimum column to certify. Target column is the bar to set for the real-world build that follows class.

CRITERION	MINIMUM (CERTIFIED)	TARGET
API endpoints	3 returning JSON	**5+ with full CRUD**
Frontend pages	2 with navigation	**4+ with sidebar, routes, detail**
Data layer	Reads from JSON	**Interface-based, swappable**
AI integration	Chat sends / receives	**Tool use against live data**
Authentication	Role picker present	**Role-based filtering enforced**
Deployment	Container runs locally	**Deployed to Azure with a public URL**

SIX PRINCIPLES TO REPEAT ALL DAY

Plant in the opening; cite by number when something breaks.

- **1. The conversation is the IDE.** The AI chat is the primary tool; the editor is for verification.
- **2. Scaffold → flesh out → integrate.** Skeleton first. Always. Make it work before you make it pretty.
- **3. The 3-minute rule.** Stuck for 3 minutes? Paste the error back to the AI. Demonstrate this aloud the first time it happens.
- **4. Incremental deployment.** Verify after each prompt. Container is the end — not a finale.
- **5. Interface first. Implementation later.** Define *what* before *how*. (Module 4 is where this earns its keep; Module 9 is where it gets paid.)
- **6. Copy the error, not your frustration.** Give the AI the message, not the mood.

RECOVERY & PACING CUES

When the room slips, this is the order of operations.

- **Answer-key reference.** The finished build lives at `heywood-inventory/` ; file map in `heywood-inventory/docs/STUDENT_GUIDE.md` (../../heywood-inventory/docs/STUDENT_GUIDE.md). Send a student there *only after* the 3-Minute Rule (Principle 3) fires *twice* on the same prompt — never as a starting point. Pull up **slide 49** on the projector when you call it out.
- **Pacing valve.** If you fall behind, compress **Module 9** (External Integrations — default to Path A only). Everyone still ships a container.
- **Module 2 gotchas.** Go not on PATH → restart terminal. `npm create vite` fails → `npm install -g npm@latest` . Docker needs admin → defer to Module 10.
- **Module 3 gotchas.** Port 8080 in use → switch to 8081. Module-name mismatch → align `go.mod` . Missing `go.sum` → `go mod tidy` .
- **Module 5 gotchas.** CORS error → the fix is the Vite proxy, *not* CORS middleware. PostCSS / Tailwind config is the other 30-second fix.
- **Module 7 gotcha.** No `OPENAI_API_KEY` → `export OPENAI_API_KEY=…` . Tool-format errors: paste the OpenAI error back — it fixes its own definition.
- **Module 10 gotcha.** Bloated image → `.dockerignore` . SPA 404 on direct nav → SPA fallback missing. Azure: `az login` first.
- **Architecture cadence.** Return to the diagram *nine times* today — once empty (M1), once after setup (M2 · 0/4), then after every build module (M3 1/4 → M4 2/4 → M5 3/4 → M6 still 3/4 → M7 4/4 → M8 hardened → M9 deepened → M10 shipped). It is the spine of the day.
- **Tone shifts.** M3 — first JSON in browser, twenty seconds of silence is fine. M5 — have a student share screen. M7 — this is the apex; let it land. M10 — pause ten seconds at the diagram and name what they did.