Assignment: Smart Helpdesk with Agentic Triage

Goal: Build an end‑to‑end web application where users raise support tickets and an **AI coworker** (agentic workflow) triages them by classifying, fetching relevant knowledge‑base (KB) articles, drafting a reply, and either auto‑resolving or assigning to a human. You may do this as **MERN‑only** (Node orchestrator) or **MERN + Python** (FastAPI worker). You are free to use your favourite vibe coding techniques.

Timebox: 48 hours recommended for take‑home + 60–90 min live review.

What we're evaluating (the "vibe"):

- Thoughtful architecture & boundaries; clean, readable code; small, purposeful commits.
- Problem‑solving clarity; pragmatic trade‑offs; testing mindset; docs & DX.
- Secure defaults; resilience (retries/timeouts/idempotency); observability (logs/trace).
- UI craft: fast, simple, accessible; good empty/error/loading states.
- Agentic reasoning: tool selection, planning, safe prompting/guardrails, fallbacks.
- How you can use AI in your development process.

User Stories

Roles:

- **End User**: creates tickets, views status and agent replies.
- **Support Agent**: reviews triage, edits/drafts final reply, resolves tickets.
- **Admin**: manages KB articles; sets agent thresholds (confidence, auto‑close toggle).

Core flows:

1. **Auth & Roles**: Sign up / sign in with JWT; role‑based access (Admin/Agent/User).
2. **KB Management (Admin)**: CRUD articles (title, body, tags); publish/unpublish.
3. **Ticket Lifecycle (User)**: Create ticket (title, description, category optional, attachments by URL). See timeline of actions.
4. **Agentic Triage (System)**: On new ticket:
   - Classify category (billing / tech / shipping / other).
   - Retrieve top KB articles (keyword search minimum; embedding/vector optional).
   - Draft a suggested reply with citations to KB.

    - Compute a confidence score (0–1). If `auto_close` on **and** score ≥ threshold, auto-reply & close; else assign to a human.
    - Log each step (trace id) to an **Audit Log** visible in UI.
5. **Agent Review (Support Agent)**: Accept/edit draft, send reply, reopen/close ticket.
6. **Notifications**: Emit an in-app notification and/or email stub on status change.

Stretch flows (pick any 1–2):

- Real-time updates via WebSocket/Server-Sent Events.
- SLA checks: mark breach if not responded in X hours; nightly job.
- Feedback loop: thumbs up/down on AI reply; retrainable prompts config.
- Attachments: extract simple text from `.txt`/`.md` URLs and include in triage.

Track Options

Track A — MERN Only (Agent in Node)

- **Frontend:** React + Vite + (Context/Zustand/Redux), React Router. Tailwind optional.
- **Backend:** Node 20+ / Express + Mongoose (MongoDB Atlas/local). Background processing via in-process queue or BullMQ (Redis) if you prefer.
- **Agentic logic:** Implement the workflow in Node. LLM calls **optional**; provide a **deterministic stub** (below) to avoid requiring API keys.

Track B — MERN + Python (Agent Worker)

- **Frontend:** Same as Track A.
- **Backend:** Node/Express is the API gateway & persistence layer; publish triage jobs to a queue (Redis) or HTTP to Python.
- **Agent Worker (Python):** FastAPI + Pydantic + (Celery/RQ optional) for async tasks. Implement LLM calls or deterministic stub. Return structured JSON back to Node.

LLM Usage: If you have a key (e.g., OpenAI), you may use it via environment vars. If not, the **stub** must fully work and be testable. Prompts must be in code or a `.prompt.md` file.

Data Model (suggested)

User

- `_id`, `name`, `email` (unique), `password_hash`, `role` in {`admin`, `agent`, `user`}, `createdAt`.

Article (KB)

- `_id`, `title`, `body`, `tags: string[]`, `status` in {`draft`, `published`}, `updatedAt`.

Ticket

- `_id`, `title`, `description`, `category` in {`billing`, `tech`, `shipping`, `other`}, `status` in {`open`, `triaged`, `waiting_human`, `resolved`, `closed`}, `createdBy`, `assignee`, `agentSuggestionId?`, `createdAt`, `updatedAt`.

AgentSuggestion

- `_id`, `ticketId`, `predictedCategory`, `articleIds: string[]`, `draftReply`, `confidence: number`, `autoClosed: boolean`, `modelInfo` (provider, model, promptVersion, latencyMs), `createdAt`.

AuditLog

- `_id`, `ticketId`, `traceId`, `actor` in {`system`, `agent`, `user`}, `action` (e.g., `TICKET_CREATED`, `AGENT_CLASSIFIED`, `KB_RETRIEVED`, `DRAFT_GENERATED`, `AUTO_CLOSED`, `ASSIGNED_TO_HUMAN`, `REPLY_SENT`), `meta` (JSON), `timestamp`.

Config

- `_id`, `autoCloseEnabled: boolean`, `confidenceThreshold: number` (0–1), `slaHours: number`.

API (minimum)

Auth

- `POST /api/auth/register` → {token}
- `POST /api/auth/login` → {token}

KB

- `GET /api/kb?query=...` (search title/body/tags)
- `POST /api/kb` (admin)

- `PUT /api/kb/:id` (admin)
- `DELETE /api/kb/:id` (admin)

Tickets

- `POST /api/tickets` (user)
- `GET /api/tickets` (filter by status/my tickets)
- `GET /api/tickets/:id`
- `POST /api/tickets/:id/reply` (agent) → change status
- `POST /api/tickets/:id/assign` (admin/agent)

Agent

- `POST /api/agent/triage` (internal) → enqueues triage for a ticket
- `GET /api/agent/suggestion/:ticketId`

Config

- `GET /api/config` / `PUT /api/config` (admin)

Audit

- `GET /api/tickets/:id/audit`

> You may add endpoints; keep them RESTful, versioned if you like (`/api/v1`). Use proper HTTP status codes.

Agentic Workflow (required steps)

1. **Plan**: Build a small planner that decides the steps given a ticket (classification → retrieval → drafting → decision). Hardcode the plan or encode as a simple state machine.
2. **Classify**: Use a prompt or rule-based keywords (deterministic stub allowed). Output schema:
   { "predictedCategory": "billing|tech|shipping|other", "confidence": 0.0 }
3. **Retrieve KB**: At least keyword search (simple regex/BM25/TF-IDF). Return top 3 article IDs with snippet scores.
4. **Draft Reply**: Compose a short answer with numbered references to the selected KB articles. Output schema:
   { "draftReply": "...", "citations": ["<articleId>", "<articleId>"] }

5. **Decision**: If `autoCloseEnabled` and `confidence ≥ threshold` → store suggestion, create agent reply, mark ticket `resolved`, log `AUTO_CLOSED`. Else mark `waiting_human` and assign to a human.
6. **Logging**: Every step must append an **AuditLog** event with a `traceId` (UUID) consistent across the pipeline.

Deterministic LLM Stub (must include):

- Implement `LLMProvider` with an interface `classify(text)`, `draft(text, articles)`.
- Provide a `STUB_MODE=true` env so we can run without keys. The stub should:
  - Classify by simple heuristics (words: "refund/invoice"→billing, "error/bug/stack"→tech, "delivery/shipment"→shipping, else other) and generate a pseudo confidence based on keyword matches.
  - Draft a templated reply inserting KB titles.

Frontend Requirements

- **Pages:** Login/Register; KB List+Editor (admin only); Ticket List; Ticket Detail (conversation thread + agent suggestion + audit timeline); Settings (config).
- **State:** Keep auth token securely; show role-based menus.
- **UX:** Clear CTAs; loading skeletons; error toasts; form validation; responsive layout.
- **Nice to have:** Search & filters; optimistic updates; accessible components (keyboard nav, ARIA labels).

Security & Reliability

- Don't log secrets. Never return stack traces to clients.
- Input validation (e.g., Zod/Joi) on all POST/PUT.
- JWT with expiry & refresh or short-lived access + refresh.
- Rate limit auth & mutation endpoints. CORS configured narrowly.
- Timeouts for agent calls; retry with backoff; idempotency key for triage jobs.

Observability

- Structured logs (JSON) with `traceId` & `ticketId` where relevant.

- Basic request logging middleware (method, path, latency, status).
- Expose `/healthz` and `/readyz`.

DevOps (minimum viable)

- **Docker Compose** with services: `client`, `api`, `mongo`, (`agent`, `redis` if Track B or BullMQ).
- One-command run: `docker compose up`.
- Seed script to insert sample users, KB articles, and tickets.

Testing

- **Backend:** At least 5 tests (Jest/Vitest) covering: auth, KB search, ticket create, agent triage decision, audit logging.
- **Frontend:** At least 3 tests (Vitest/RTL) for rendering + form validation.
- **Fixtures:** Provide JSON fixtures for stubbed LLM outputs and seed data.
- **Postman/Thunder tests (optional):** Include a collection.

Acceptance Criteria (we will verify)

1. Can register/login and create a ticket as a normal user.
2. Creating a ticket triggers triage; an **AgentSuggestion** is persisted.
3. If confidence ≥ threshold and auto-close is on, ticket is moved to `resolved` with agent reply appended; user sees the reply.
4. If below threshold, ticket becomes `waiting_human`; an agent can open the ticket, review the draft, edit, and send.
5. Audit timeline shows ordered steps with timestamps and `traceId`.
6. KB search returns relevant articles for simple queries.
7. App runs with `STUB_MODE=true` and **no external keys**.
8. `docker compose up` brings the stack up; clear README with envs and seed steps.

Deliverables

- **Source code** in a public/private repo.
- **README** with:
  - Architecture diagram + brief rationale.
  - Setup (env vars, Docker, seed) & run instructions.
  - How agent works (plan, prompts, tools) & guardrails.
  - Testing instructions & coverage summary.
- **Short Loom/video (≤5 min)** walkthrough: KB add, ticket create, triage, resolution.
- A public url deployed on your favorite cloud like v0.

Scoring Rubric (100 points)

- Core functionality & correctness – **30**
- Code quality (structure, naming, modularity) – **15**
- Data modeling & API design – **10**
- UI/UX (clarity, states, accessibility basics) – **10**
- Testing (breadth + meaningful assertions) – **10**
- Agentic workflow design (planning, tools, guardrails, logs) – **15**
- Security & reliability (validation, rate limits, retries) – **5**
- DevOps & DX (Docker, scripts, README) – **5**

Disqualifiers: committed secrets; cannot run locally; failing build; no README.

Anti‑Cheat & Review Plan

- Require a short Loom demo and commit history across the timebox.
- 60–90 min live review: ask to add a small feature (e.g., change confidence threshold logic to use per‑category threshold) and fix a seeded bug.
- Randomize 2–3 KB articles/tickets at review time to check adaptability.

Starter Seed (example)

Env (example)

```
PORT=8080
MONGO_URI=mongodb://mongo:27017/helpdesk
JWT_SECRET=change-me
AUTO_CLOSE_ENABLED=true
```

```
CONFIDENCE_THRESHOLD=0.78
STUB_MODE=true
OPENAI_API_KEY= # optional
```

KB Seed (abbrev)

```
[
{"title":"How to update payment
method","body":"...","tags":["billing","payments"],"status":"published"},
{"title":"Troubleshooting 500 errors","body":"...","tags":["tech","errors"],"status":"published"},
{"title":"Tracking your shipment","body":"...","tags":["shipping","delivery"],"status":"published"}
]
```

Ticket Seed (abbrev)

```
[
{"title":"Refund for double charge","description":"I was charged twice for order
#1234","category":"other"},
{"title":"App shows 500 on login","description":"Stack trace mentions auth
module","category":"other"},
{"title":"Where is my package?","description":"Shipment delayed 5 days","category":"other"}
]
```

Hints & Gotchas

- Keep **prompts** and **stub rules** versioned; include `promptVersion` in `modelInfo`.
- Avoid leaky abstractions: separate `agent` from `kb` and `tickets` services.
- Immutability for audit events; don't rewrite history.
- Be explicit about timezones, ISO timestamps.

Optional Extras (pick any)

- Simple RAG: store TF-IDF vectors or call a local embedding model; fall back to keyword search.
- Feature flags for `autoCloseEnabled` and per-category thresholds.
- Export audit logs as NDJSON.
- Minimal role-based UI tests with Playwright.

Submission Checklist

What great submissions usually show

- **Clarity over cleverness**; simple patterns that scale.
- **Guardrails** around LLMs: input/output schemas, max tokens, temperature, system prompts, refusal handling.
- **Developer empathy**: fast local setup, meaningful errors, seed data, fixtures.

Good luck — have fun building your agentic helpdesk! 🚀