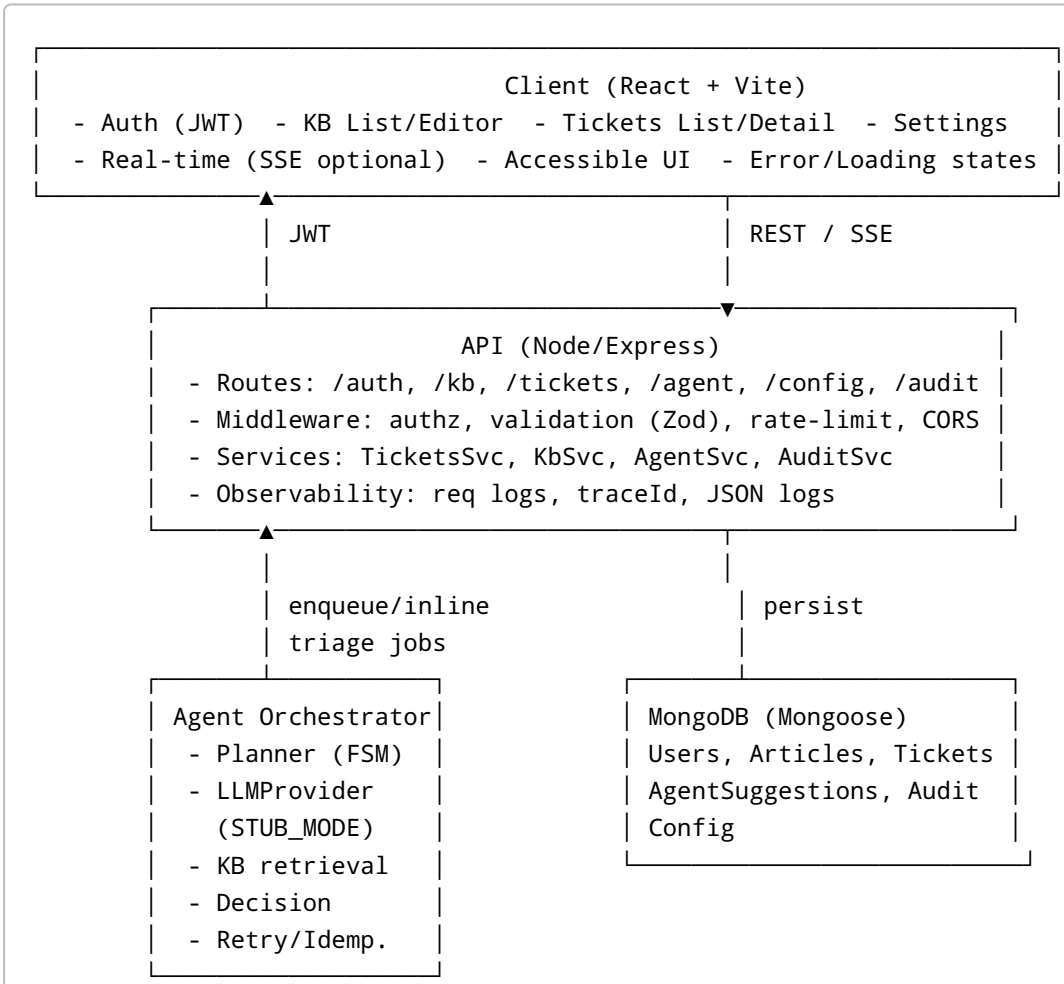
Smart Helpdesk (Agentic Triage)

This doc gives you a **complete blueprint + starter snippets** to deliver the assignment fast, with clean boundaries, testable stubs, and Dockerized DX. It targets **Track A (MERN-only, Agent in Node)**, with notes for extending to **Track B (Python worker)**.

---

1) High-level Architecture



> **Track B note**: Replace in-process Agent Orchestrator with a Redis/BullMQ queue and a **FastAPI worker** that consumes triage jobs and returns structured results to Node.

---

2) Data Models (Mongoose)

```
// models/User.ts
import { Schema, model } from 'mongoose';
const UserSchema = new Schema({
  name: { type: String, required: true },
```

```
  email: { type: String, required: true, unique: true, index: true },
  password_hash: { type: String, required: true },
  role: { type: String, enum: ['admin','agent','user'], default: 'user' },
}, { timestamps: { createdAt: true, updatedAt: false } });
export default model('User', UserSchema);
```

```
// models/Article.ts
import { Schema, model } from 'mongoose';
const ArticleSchema = new Schema({
  title: { type: String, required: true },
  body: { type: String, required: true },
  tags: { type: [String], default: [] },
  status: { type: String, enum: ['draft','published'], default: 'draft' },
}, { timestamps: { createdAt: false, updatedAt: true } });
export default model('Article', ArticleSchema);
```

```
// models/Ticket.ts
import { Schema, model, Types } from 'mongoose';
const TicketSchema = new Schema({
  title: { type: String, required: true },
  description: { type: String, required: true },
  category: { type: String, enum: ['billing','tech','shipping','other'],
default: 'other' },
  status: { type: String, enum:
['open','triaged','waiting_human','resolved','closed'], default: 'open' },
  createdBy: { type: Types.ObjectId, ref: 'User', required: true },
  assignee: { type: Types.ObjectId, ref: 'User' },
  agentSuggestionId: { type: Types.ObjectId, ref: 'AgentSuggestion' },
  attachments: { type: [String], default: [] }, // URL strings
}, { timestamps: true });
export default model('Ticket', TicketSchema);
```

```
// models/AgentSuggestion.ts
import { Schema, model, Types } from 'mongoose';
const AgentSuggestionSchema = new Schema({
  ticketId: { type: Types.ObjectId, ref: 'Ticket', required: true, index:
true },
  predictedCategory: { type: String, enum:
['billing','tech','shipping','other'], required: true },
  articleIds: { type: [Types.ObjectId], ref: 'Article', default: [] },
  draftReply: { type: String, required: true },
  confidence: { type: Number, min: 0, max: 1, required: true },
  autoClosed: { type: Boolean, default: false },
  modelInfo: {
    provider: String, model: String, promptVersion: String, latencyMs: Number
  },
```

```
}, { timestamps: { createdAt: true, updatedAt: false } });
export default model('AgentSuggestion', AgentSuggestionSchema);
```

```
// models/AuditLog.ts
import { Schema, model, Types } from 'mongoose';
const AuditLogSchema = new Schema({
  ticketId: { type: Types.ObjectId, ref: 'Ticket', index: true },
  traceId: { type: String, required: true, index: true },
  actor: { type: String, enum: ['system','agent','user'], required: true },
  action: { type: String, required: true },
  meta: { type: Schema.Types.Mixed },
  timestamp: { type: Date, default: () => new Date().toISOString() },
});
export default model('AuditLog', AuditLogSchema);
```

```
// models/Config.ts
import { Schema, model } from 'mongoose';
const ConfigSchema = new Schema({
  autoCloseEnabled: { type: Boolean, default: true },
  confidenceThreshold: { type: Number, default: 0.78 },
  slaHours: { type: Number, default: 24 },
});
export default model('Config', ConfigSchema);
```

---

3) API Surface (Express)

Versioned base: `/api/v1`

```
// routes index
app.use('/api/v1/auth', authRouter);
app.use('/api/v1/kb', kbRouter);
app.use('/api/v1/tickets', ticketsRouter);
app.use('/api/v1/agent', agentRouter);
app.use('/api/v1/config', configRouter);
app.use('/api/v1/audit', auditRouter);
```

Key route snippets:

```
// routes/auth.ts
router.post('/register', validate(RegisterSchema), async (req,res)=>{/*
create user, hash(bcrypt), jwt */})
router.post('/login', validate(LoginSchema), async (req,res)=>{/* verify +
jwt */})
```

```
// routes/kb.ts (admin only for mutations)
router.get('/', authAny, async (req,res)=>{/* search title/body/tags with
regex + pagination */})
router.post('/', authRole('admin'), validate(ArticleUpsertSchema), async
(req,res)=>{/* create */})
router.put('/:id', authRole('admin'), validate(ArticleUpsertSchema), async
(req,res)=>{/* update */})
router.delete('/:id', authRole('admin'), async (req,res)=>{/* delete */})
```

```
// routes/tickets.ts
router.post('/', authRole('user','agent','admin'),
validate(TicketCreateSchema), async (req,res)=>{
  const ticket = await TicketsSvc.create({...req.body, createdBy:
req.user.id});
  await AgentSvc.enqueueTriage(ticket._id, req.headers['idempotency-key']);
  res.status(201).json(ticket);
})
router.get('/', authAny, async (req,res)=>{/* filters: status, mine,
pagination */})
router.get('/:id', authAny, async
(req,res)=>{/* includes suggestion + thread */})
router.post('/:id/reply', authRole('agent','admin'), validate(ReplySchema),
async (req,res)=>{/* add reply, set status */})
router.post('/:id/assign', authRole('agent','admin'),
validate(AssignSchema), async (req,res)=>{/* change assignee */})
```

```
// routes/agent.ts (internal)
router.post('/triage', authInternalOrRole('admin'), async (req,res)=>{/*
manually kick triage */})
router.get('/suggestion/:ticketId', authAny, async (req,res)=>{/* return
AgentSuggestion */})
```

```
// routes/config.ts
router.get('/', authAny, async (req,res)=>{ res.json(await
ConfigSvc.get()); })
router.put('/', authRole('admin'), validate(ConfigSchema), async
(req,res)=>{ res.json(await ConfigSvc.update(req.body)); })
```

```
// routes/audit.ts
router.get('/tickets/:id', authAny, async (req,res)=>{ res.json(await
AuditSvc.listByTicket(req.params.id)); })
```

---

4) Security, Validation & Middleware

- **JWT**: short-lived access + refresh pair (or single short-lived for brevity). Store in memory (client) with refresh endpoint. Attach `sub`, `role` claims.
- **Validation**: Zod schemas per endpoint; 400 on failure.
- **Rate limiting**: login/register + mutations with `express-rate-limit`.
- **CORS**: allow only client origin.
- **No stack traces** to clients; centralized error handler maps to problem+json.

```
// middleware/validate.ts (Zod)
export const validate = (schema) => (req,res,next)=>{
  const parsed = schema.safeParse({ body:req.body, query:req.query,
params:req.params });
  if(!parsed.success) return res.status(400).json({ error:
'validation_failed', issues: parsed.error.flatten() });
  Object.assign(req, parsed.data); next();
}
```

---

5) Agentic Workflow (Planner + Stub LLM)

Finite State Machine (deterministic): `PLAN → CLASSIFY → RETRIEVE_KB → DRAFT → DECIDE → WRITE_RESULTS`

```
// agent/llmProvider.ts
export interface LLMProvider {
  classify(input: string): Promise<{ predictedCategory:
'billing'|'tech'|'shipping'|'other'; confidence: number }>
  draft(input: string, articles: { _id:string, title:string }[]): Promise<{
draftReply:string, citations:string[] }>
}

export class StubLLM implements LLMProvider {
  promptVersion = 'stub.v1';
  private score(text:string, words:string[]) { return words.reduce((s,w)=> s
+ (text.toLowerCase().includes(w)?1:0), 0); }
  async classify(text:string){
    const b=this.score(text,['refund','invoice','payment','charge']);
    const t=this.score(text,['error','bug','stack','500','crash','login']);
    const s=this.score(text,['delivery','shipment','package','tracking']);
    let cat:'billing'|'tech'|'shipping'|'other'='other'; let
raw=Math.max(b,t,s);
    if(b===raw && b>0) cat='billing'; else if(t===raw && t>0) cat='tech';
else if(s===raw && s>0) cat='shipping';
    const confidence = Math.min(1, 0.4 + 0.2*raw); // simple pseudo-
confidence
    return { predictedCategory: cat, confidence };
  }
```

```
  async draft(text:string, articles:{_id:string,title:string}[]){
    const citations = articles.slice(0,3).map(a=>a._id);
    const titles = articles.slice(0,3).map((a,i)=>`${i+1}. ${a.title}
`).join('\n');
    const draftReply = `Thanks for reaching out! Based on your message, here
is what may help:\n\n${titles}\n\nPlease review the above articles. If this
doesn’t resolve your issue, reply to reopen.`;
    return { draftReply, citations };
  }
}
```

```
// agent/retrieval.ts (keyword search + simple scoring)
import Article from '../models/Article';
export async function retrieveTopArticles(query: string, limit=3){
  const rx = new RegExp(query.split(/\s+/).map(s=>escapeRegex(s)).join('|'),
'i');
  const docs = await Article.find({ $or:[ {title: rx}, {body: rx}, {tags: {
$in: query.toLowerCase().split(/\s+/) }} ], status:'published' });
  // naive rank: title hit + body hit + tag hits
  const scored = docs.map(d=>({ doc:d, score: (rx.test(d.title)?2:0) +
(rx.test(d.body)?1:0) + (d.tags?.length||0) }));
  return scored.sort((a,b)=>b.score-a.score).slice(0,limit).map(s=>s.doc);
}
function escapeRegex(s:string){ return s.replace(/[.*+?^${}()|[\]\\]/g, '\\
$&'); }
```

```
// agent/orchestrator.ts
import { v4 as uuid } from 'uuid';
import Ticket from '../models/Ticket';
import AgentSuggestion from '../models/AgentSuggestion';
import AuditLog from '../models/AuditLog';
import Config from '../models/Config';
import { retrieveTopArticles } from './retrieval';
import { StubLLM } from './llmProvider';

const llm = new StubLLM();

export async function triageTicket(ticketId: string, idempotencyKey?: string)
{
  const traceId = uuid();
  const started = Date.now();
  const ticket = await Ticket.findById(ticketId);
  if(!ticket) throw new Error('ticket_not_found');
  await log('system','PLAN',{ state:'start' });

  const { predictedCategory, confidence } = await llm.classify(`$
{ticket.title}\n${ticket.description}`);
  await log('system','AGENT_CLASSIFIED',{ predictedCategory, confidence });
```

```
  const articles = await retrieveTopArticles(`${ticket.title} $
{ticket.description}`);
  await log('system','KB_RETRIEVED',{ articleIds: articles.map(a=>a._id) });

  const { draftReply, citations } = await llm.draft(ticket.description,
articles);
  await log('system','DRAFT_GENERATED',{ citations });

  const cfg = await Config.findOne() || { autoCloseEnabled: true,
confidenceThreshold: 0.78 } as any;
  let autoClosed = false; let status: typeof ticket.status = 'waiting_human';
  if(cfg.autoCloseEnabled && confidence >= cfg.confidenceThreshold){
    autoClosed = true; status = 'resolved';
  }

  const suggestion = await AgentSuggestion.create({
    ticketId: ticket._id,
    predictedCategory,
    articleIds: articles.map(a=>a._id),
    draftReply,
    confidence,
    autoClosed,
    modelInfo: { provider: process.env.STUB_MODE==='true'?'stub':'openai',
model: 'stub', promptVersion: llm.promptVersion, latencyMs: Date.now()-
started }
  });

  ticket.category = predictedCategory;
  ticket.status = autoClosed ? 'resolved' : 'waiting_human';
  ticket.agentSuggestionId = suggestion._id;
  await ticket.save();

  await log('system', autoClosed? 'AUTO_CLOSED' : 'ASSIGNED_TO_HUMAN', {
threshold: cfg.confidenceThreshold });

  return { suggestionId: suggestion._id, autoClosed, traceId };

  async function log(actor:'system'|'agent'|'user', action:string, meta:any){
    await AuditLog.create({ ticketId, traceId, actor, action, meta,
timestamp: new Date().toISOString() });
  }
}
```

Retries/Timeouts/Idempotency - Wrap `triageTicket` calls with a **p-retry** and a **p-timeout** wrapper. - Use `idempotencyKey` header to avoid double-processing (store a short-lived key in Mongo or Redis). For brevity, show pattern:

```
// services/AgentSvc.ts
const inFlight = new Map<string, Promise<any>>();
export async function enqueueTriage(ticketId:string, idemKey?:string){
```

```
  const key = idemKey || ticketId;
  if(inFlight.has(key)) return inFlight.get(key);
  const p = triageTicket(ticketId).finally(()=> inFlight.delete(key));
  inFlight.set(key, p); return p;
}
```

---

6) Observability

- **Request logger**: method, path, status, latency, userId, traceId (if present).
- **AuditLog** for agent steps (immutable).
- **/healthz** (basic) and **/readyz** (DB ping).

```
// middleware/requestLogger.ts
app.use(async (req,res,next)=>{
  const t=Date.now();
  res.on('finish',()=>{
    console.log(JSON.stringify({ lvl:'info', ts:new Date().toISOString(),
method:req.method, path:req.path, status:res.statusCode, ms:Date.now()-t,
user:req.user?.id }));
  });
  next();
});
```

---

7) Frontend (React + Vite)

Pages - **Auth**: Login/Register. - **KB**: List, Search, Create/Edit (Admin only). - **Tickets**: List (filters), Detail (conversation, agent suggestion, audit timeline), Create. - **Settings**: Config (autoCloseEnabled, confidenceThreshold, slaHours).

State: small `Zustand` store for auth + user, React Query for data fetching (nice retries, cache, loading states).

```
// src/store/auth.ts
import create from 'zustand';
export const useAuth = create<{ token:string|null;
role:'admin'|'agent'|'user'|null; set:(t:string,r:any)=>void; clear:()=>void}
>(set=>({
  token:null, role:null, set:(t,r)=>set({token:t, role:r}), clear:
()=>set({token:null,role:null})
}));
```

```
// src/pages/TicketDetail.tsx (sketch)
export default function TicketDetail(){
  // fetch ticket, suggestion, audit timeline; render draft + actions
```

```
  // show statuses with badges; loading skeletons; error toasts
}
```

Accessibility: keyboard focus, ARIA labels on forms and buttons. **Loading**: skeleton placeholders. **Errors**: toast notifications.

SSE (stretch): subscribe to `/api/v1/tickets/:id/stream` for live audit updates.

---

8) DevOps: Docker & Seeds

docker-compose.yml

```yaml
version: '3.9'
services:
  mongo:
    image: mongo:7
    ports: [ '27017:27017' ]
    volumes: [ mongo-data:/data/db ]
  api:
    build: ./api
    environment:
      - MONGO_URI=mongodb://mongo:27017/helpdesk
      - PORT=8080
      - JWT_SECRET=change-me
      - AUTO_CLOSE_ENABLED=true
      - CONFIDENCE_THRESHOLD=0.78
      - STUB_MODE=true
    ports: [ '8080:8080' ]
    depends_on: [ mongo ]
  client:
    build: ./client
    environment:
      - VITE_API_BASE=http://localhost:8080/api/v1
    ports: [ '5173:5173' ]
    depends_on: [ api ]
volumes:
  mongo-data:
```

Seed script (`api/scripts/seed.ts`)

```
// connect, upsert admin/agent/user, insert KB + Tickets (from JSON fixtures)
```

Run:

```
docker compose up --build
# API at :8080, Client at :5173
```

---

9) Testing Plan

Backend (Jest): 5 tests minimum 1. **Auth**: register+login returns JWT; rejects bad creds. 2. **KB Search**: returns expected seed article for query "payment". 3. **Ticket Create**: creates ticket & enqueues triage (spy on AgentSvc). 4. **Agent Decision**: with threshold 0.5 and billing keywords, status becomes `resolved`. 5. **Audit Logging**: triage writes ordered events with same `traceId`.

Frontend (Vitest + RTL): 3 tests minimum 1. **Login form**: validates required fields and shows API error. 2. **Ticket create form**: validates title/description and shows success. 3. **Ticket detail**: renders agent draft + audit timeline skeleton → loaded state.

Fixtures: `api/fixtures/*.json` for users, KB, tickets, and stub LLM outputs.

---

10) Security & Reliability Checklist

- [x] Zod input validation; sanitize regex.
- [x] JWT expiry; refresh endpoint; role-based guards.
- [x] Rate limit on `/auth` and mutating endpoints.
- [x] Timeouts/retries around triage; idempotency key support.
- [x] CORS restricted to client origin.
- [x] No secrets in logs.

---

11) README Outline (copy-paste friendly)

```
# Smart Helpdesk (Agentic Triage)

## Architecture
- Diagram + rationale (MERN-only; agent orchestrated in Node, stubbed LLM for
determinism).

## Setup
```bash
cp api/.env.example api/.env
cp client/.env.example client/.env
docker compose up --build
```

- API: `http://localhost:8080/api/v1`
- Client: `http://localhost:5173`
```

Seed

```
docker exec -it <api> node dist/scripts/seed.js
```

Agent: Plan & Guardrails

- FSM: plan→classify→retrieve→draft→decide→write
- StubLLM (STUB_MODE=true) uses heuristics; capped output; schemas validated.
- Prompts versioned (`promptVersion`).

Testing

- `npm run test` in `api` and `client`.

Acceptance Checklist

1. Register/login; create ticket.
2. Triage runs; `AgentSuggestion` persisted.
3. High confidence → auto-resolve with draft reply visible.
4. Low confidence → waiting_human; agent edits & sends.
5. Audit timeline shows steps with `traceId`.
6. KB search returns expected articles.
7. Runs with STUB_MODE=true.
8. `docker compose up` spins the stack.

```
---

## 12) Nice-to-haves & Stretch

- **RAG-lite**: TF-IDF vectors in Mongo (precompute) → fallback to
keyword.
- **Per-category thresholds**: extend `Config` to an object
`{ thresholds: { billing, tech, shipping, other } }`.
- **SSE**: stream audit events to ticket detail page.
- **NDJSON Export**: `/api/v1/audit/export.ndjson?ticketId=...`.

---

## 13) File/Folder Structure

```text
repo/
  api/
    src/
      models/ (User, Article, Ticket, AgentSuggestion, AuditLog, Config)
      routes/ (auth, kb, tickets, agent, config, audit)
      agent/ (llmProvider.ts, orchestrator.ts, retrieval.ts)
      middleware/ (auth.ts, validate.ts, errors.ts, requestLogger.ts)
```
```

```
      services/ (AgentSvc.ts, TicketsSvc.ts, KbSvc.ts, AuditSvc.ts,
ConfigSvc.ts)
      utils/ (jwt.ts, rateLimit.ts, http.ts)
      app.ts, server.ts
    fixtures/ (users.json, kb.json, tickets.json)
    scripts/ (seed.ts)
    tests/ (auth.spec.ts, kb.spec.ts, ticket.spec.ts, agent.spec.ts,
audit.spec.ts)
    Dockerfile
  client/
    src/
      pages/ (Login, Register, Kb, TicketList, TicketDetail, Settings)
      components/ (Form, Table, Timeline, DraftCard, StatusBadge)
      store/ (auth.ts)
      api/ (client.ts)
      App.tsx, main.tsx
    index.html
    Dockerfile
  docker-compose.yml
  README.md
```

---

14) Minimal Code to Start Server (Express)

```
// api/src/app.ts
import express from 'express';
import cors from 'cors';
import helmet from 'helmet';
import morgan from 'morgan';
import mongoose from 'mongoose';
import routes from './routes';

export async function createApp(){
  await mongoose.connect(process.env.MONGO_URI!);
  const app = express();
  app.use(helmet());
  app.use(cors({ origin: process.env.CLIENT_ORIGIN || 'http://localhost:
5173' }));
  app.use(express.json({ limit:'1mb' }));
  app.use(morgan('tiny'));
  app.get('/healthz',(_,res)=>res.send('ok'));
  app.use('/api/v1', routes);
  app.use((err,req,res,next)=>{ console.error(err); res.status(500).json({
error:'internal_error' }); });
  return app;
}

// api/src/server.ts
import { createApp } from './app';
```

```
const PORT = Number(process.env.PORT||8080);
createApp().then(app=> app.listen(PORT, ()=>console.log(`api on :${PORT}`)));
```

---

15) Frontend: Quick Scaffolding (Vite)

```
npm create vite@latest client -- --template react-ts
cd client && npm i axios zustand @tanstack/react-query react-router-dom
```

```
// client/src/api/client.ts
import axios from 'axios';
export const api = axios.create({ baseURL: import.meta.env.VITE_API_BASE });
export function setAuth(token?:string){ if(token)
api.defaults.headers.common['Authorization'] = `Bearer ${token}`; else
delete api.defaults.headers.common['Authorization']; }
```

```
// client/src/App.tsx (skeleton with Router)
```

---

16) How to Demo (≤5 min Loom)

1) Add a KB article.
2) Create a ticket (“Refund for double charge”).
3) Show audit timeline + agent draft.
4) Toggle `autoCloseEnabled` / change `confidenceThreshold` and repeat.
5) Show waiting_human flow: agent edits and sends final reply.

---

17) Review-time Change (prep)

- Add **per-category thresholds** to Config and change decision logic:

```
const thresh = cfg.thresholds?.[predictedCategory] ??
cfg.confidenceThreshold;
if(cfg.autoCloseEnabled && confidence >= thresh){ ... }
```

---

18) Anti-cheat hygiene

- No committed secrets.
- Deterministic stubs with fixtures.
- Small commits; meaningful messages; timestamps over the timebox.

You can now scaffold from this doc. If you want, I can generate the initial repo files (Dockerfiles, package.json, seed fixtures) next.