<https://chatgpt.com/canvas/shared/68094806e5d88191b34f2d49735e7074>

**backend/main.py**

import logging import os from fastapi import FastAPI, HTTPException, UploadFile, File, WebSocket, WebSocketDisconnect, Response, Depends, BackgroundTasks from fastapi.middleware.cors import CORSMiddleware from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm from prometheus\_client import Counter, generate\_latest, CONTENT\_TYPE\_LATEST from sqlalchemy import create\_engine, Column, String, Float, Text from sqlalchemy.ext.declarative import declarative\_base from sqlalchemy.orm import sessionmaker, scoped\_session from ai\_services import classifier, extractor, summarizer from celery import Celery

**--- Logging Configuration ---**

logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(name)s - %(levelname)s - %(message)s" ) logger = logging.getLogger("litautomate")

**--- Environment & Configuration ---**

DATABASE\_URL = os.getenv("DATABASE\_URL", "postgresql+psycopg2://user:pass@postgres:5432/papersdb") CELERY\_BROKER\_URL = os.getenv("CELERY\_BROKER\_URL", "redis://redis:6379/0") CELERY\_RESULT\_BACKEND = os.getenv("CELERY\_RESULT\_BACKEND", "redis://redis:6379/0") SECRET\_KEY = os.getenv("SECRET\_KEY", "CHANGE\_ME") ALGORITHM = "HS256" ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30

**--- Database Setup with SQLAlchemy ---**

engine = create\_engine(DATABASE\_URL, pool\_pre\_ping=True) SessionLocal = scoped\_session(sessionmaker(autocommit=False, autoflush=False, bind=engine)) Base = declarative\_base()

class Paper(Base): **tablename** = 'papers' doc\_id = Column(String, primary\_key=True, index=True) title = Column(String, nullable=False) authors = Column(String, nullable=False) relevance\_score = Column(Float, default=0.0) label = Column(String, default='pending') summary = Column(Text, default='')

Base.metadata.create\_all(bind=engine)

**--- Celery Setup for Background Tasks ---**

celery\_app = Celery( 'litautomate\_tasks', broker=CELERY\_BROKER\_URL, backend=CELERY\_RESULT\_BACKEND )

@celery\_app.task(bind=True, acks\_late=True) def process\_document\_task(self, doc\_id: str, text: str): try: score, label = classifier.predict(text) summary = summarizer.summarize(text) db = SessionLocal() paper = db.query(Paper).get(doc\_id) paper.relevance\_score = score paper.label = label paper.summary = summary db.commit() db.close() logger.info(f"Processed {doc\_id}: {label} ({score})") except Exception as e: logger.error(f"Error processing doc {doc\_id}: {e}") raise self.retry(exc=e, countdown=60)

**--- Authentication ---**

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="/token")

def fake\_verify\_token(token: str = Depends(oauth2\_scheme)): if token != "secrettoken": raise HTTPException(status\_code=401, detail="Invalid or expired token") return token

**--- FastAPI App ---**

app = FastAPI(title="AI LitAutomate API with HIL, Auth & Monitoring")

**--- CORS Configuration ---**

origins = [os.getenv("FRONTEND\_URL", "http://localhost:3000")] app.add\_middleware( CORSMiddleware, allow\_origins=origins, allow\_credentials=True, allow\_methods=["*"], allow\_headers=["*"] )

**--- Prometheus Metrics ---**

classification\_requests = Counter('classification\_requests\_total', 'Total number of document classifications') upload\_requests = Counter('upload\_requests\_total', 'Total number of document upload requests')

**--- WebSocket Connection Manager ---**

class ConnectionManager: def **init**(self): self.active\_connections = [] async def connect(self, ws: WebSocket): await ws.accept(); self.active\_connections.append(ws) def disconnect(self, ws: WebSocket): self.active\_connections.remove(ws) async def broadcast(self, message: dict): for conn in list(self.active\_connections): try: await conn.send\_json(message) except Exception as e: logger.error(f"WebSocket error: {e}") manager = ConnectionManager()

**--- Health & Metrics Endpoints ---**

@app.get("/healthz") def health\_check(): return {"status": "ok"}

@app.get("/metrics", dependencies=[Depends(fake\_verify\_token)]) def metrics(): return Response(generate\_latest(), media\_type=CONTENT\_TYPE\_LATEST)

**--- Token Endpoint ---**

@app.post("/token") def login(form\_data: OAuth2PasswordRequestForm = Depends()): if form\_data.username == "admin" and form\_data.password == "password": return {"access\_token": "secrettoken", "token\_type": "bearer"} raise HTTPException(status\_code=400, detail="Incorrect username or password")

**--- List Papers ---**

@app.get("/papers", dependencies=[Depends(fake\_verify\_token)]) def list\_papers(): db = SessionLocal() papers = db.query(Paper).all() db.close() return papers

**--- Classify Endpoint ---**

@app.post("/classify/{doc\_id}", dependencies=[Depends(fake\_verify\_token)]) def classify\_document(doc\_id: str, background\_tasks: BackgroundTasks): classification\_requests.inc() db = SessionLocal() try: paper = db.query(Paper).get(doc\_id) if not paper: raise HTTPException(404, "Paper not found") text = extractor.fetch\_text(doc\_id) background\_tasks.add\_task(process\_document\_task.delay, doc\_id, text) return {"msg": "Classification queued", "doc\_id": doc\_id} except HTTPException: raise except Exception as e: logger.error(f"Error in classify endpoint: {e}") raise HTTPException(500, "Internal server error") finally: db.close()

**--- Upload Document Endpoint ---**

@app.post("/upload", dependencies=[Depends(fake\_verify\_token)]) async def upload\_document(file: UploadFile = File(...)): upload\_requests.inc() try: contents = await file.read() text = extractor.extract\_text\_from\_bytes(contents) doc\_id = file.filename title, authors = extractor.extract\_metadata(doc\_id, text) db = SessionLocal() paper = Paper(doc\_id=doc\_id, title=title, authors=authors) db.merge(paper) db.commit() db.close() logger.info(f"Uploaded {doc\_id}, queued for processing") process\_document\_task.delay(doc\_id, text) return {"msg": "Upload successful, processing queued", "doc\_id": doc\_id} except Exception as e: logger.error(f"Error uploading document: {e}") raise HTTPException(400, "Failed to upload and process document")

**--- WebSocket Endpoint ---**

@app.websocket("/ws") async def websocket\_endpoint(ws: WebSocket): await manager.connect(ws) try: while True: await ws.receive\_text() except WebSocketDisconnect: manager.disconnect(ws)

**--- Dockerfile for backend ---**

FROM python:3.10-slim WORKDIR /app COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt COPY . . CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

**--- docker-compose.yml ---**

version: '3.8' services: redis: image: redis:6-alpine healthcheck: test: ["CMD", "redis-cli", "ping"] interval: 30s timeout: 10s retries: 5 start\_period: 30s ports: - '6379:6379'

postgres: image: postgres:13-alpine environment: POSTGRES\_USER: user POSTGRES\_PASSWORD: pass POSTGRES\_DB: papersdb healthcheck: test: ["CMD-SHELL", "pg\_isready -U user -d papersdb"] interval: 30s timeout: 10s retries: 5 start\_period: 30s volumes: - pgdata:/var/lib/postgresql/data ports: - '5432:5432'

backend: build: context: ./backend dockerfile: Dockerfile environment: DATABASE\_URL: postgresql+psycopg2://user:pass@postgres:5432/papersdb CELERY\_BROKER\_URL: redis://redis:6379/0 CELERY\_RESULT\_BACKEND: redis://redis:6379/0 SECRET\_KEY: CHANGE\_ME FRONTEND\_URL: [http://localhost:3000](http://localhost:3000/) ports: - '8000:8000' depends\_on: redis: condition: service\_healthy postgres: condition: service\_healthy healthcheck: test: ["CMD-SHELL", "curl -f http://localhost:8000/healthz || exit 1"] interval: 30s timeout: 10s retries: 5 start\_period: 30s

celery\_worker: build: context: ./backend dockerfile: Dockerfile command: celery -A main.celery\_app worker --loglevel=info environment: DATABASE\_URL: postgresql+psycopg2://user:pass@postgres:5432/papersdb CELERY\_BROKER\_URL: redis://redis:6379/0 CELERY\_RESULT\_BACKEND: redis://redis:6379/0 depends\_on: redis: condition: service\_healthy postgres: condition: service\_healthy healthcheck: test: ["CMD-SHELL", "celery -A main.celery\_app inspect ping"] interval: 30s timeout: 10s retries: 5 start\_period: 30s

volumes: pgdata:

**k8s/deployment.yaml**

apiVersion: apps/v1 kind: Deployment metadata: name: litautomate-backend spec: replicas: 3 selector: matchLabels: app: litautomate template: metadata: labels: app: litautomate spec: containers: - name: backend image: myrepo/litautomate:latest ports: - containerPort: 8000 env: - name: DATABASE\_URL value: "postgresql+psycopg2://user:pass@postgres:5432/papersdb" - name: CELERY\_BROKER\_URL value: "redis://redis:#{6379}/0" - name: CELERY\_RESULT\_BACKEND value: "redis://redis:6379/0" readinessProbe: httpGet: path: /healthz port: 8000 initialDelaySeconds: 5 periodSeconds: 10 failureThreshold: 3 livenessProbe: httpGet: path: /healthz port: 8000 initialDelaySeconds: 15 periodSeconds: 20 failureThreshold: 5 restartPolicy: Always

**k8s/service.yaml**

apiVersion: v1 kind: Service metadata: name: litautomate-service spec: selector: app: litautomate ports:

* protocol: TCP port: 80 targetPort: 8000 type: LoadBalancer

**k8s/ingress.yaml**

apiVersion: networking.k8s.io/v1 kind: Ingress metadata: name: litautomate-ingress annotations: kubernetes.io/ingress.class: "nginx" nginx.ingress.kubernetes.io/rewrite-target: /$1 nginx.ingress.kubernetes.io/proxy-read-timeout: "3600" nginx.ingress.kubernetes.io/proxy-send-timeout: "3600" # OAuth2 Proxy authentication nginx.ingress.kubernetes.io/auth-url: "https://$host/oauth2/auth" nginx.ingress.kubernetes.io/auth-signin: "https://$host/oauth2/start?rd=$request\_uri" nginx.ingress.kubernetes.io/auth-response-headers: "X-Auth-Request-User, X-Auth-Request-Email" spec: tls:

* hosts:
  + litautomate.example.com secretName: litautomate-tls rules:
* host: litautomate.example.com http: paths:
  + path: /(.\*) pathType: Prefix backend: service: name: litautomate-service port: number: 80
  + path: /ws pathType: Prefix backend: service: name: litautomate-service port: number: 80

**k8s/oauth2-proxy.yaml**

**apiVersion: apps/v1 kind: Deployment metadata: name: oauth2-proxy spec: replicas: 2 selector: matchLabels: app: oauth2-proxy template: metadata: labels: app: oauth2-proxy spec: containers: - name: oauth2-proxy image: quay.io/oauth2-proxy/oauth2-proxy:latest args: - --provider=oidc - --oidc-issuer-url=**[**https://accounts.example.com**](https://accounts.example.com/) **- --client-id=${OAUTH2\_PROXY\_CLIENT\_ID} - --client-secret=${OAUTH2\_PROXY\_CLIENT\_SECRET} - --cookie-secret=${OAUTH2\_PROXY\_COOKIE\_SECRET} - --upstream=**[**http://litautomate-service.default.svc.cluster.local:80**](http://litautomate-service.default.svc.cluster.local/) **- --http-address=0.0.0.0:4180 env: - name: OAUTH2\_PROXY\_CLIENT\_ID valueFrom: secretKeyRef: name: oauth2-proxy-secret key: client-id - name: OAUTH2\_PROXY\_CLIENT\_SECRET valueFrom: secretKeyRef: name: oauth2-proxy-secret key: client-secret - name: OAUTH2\_PROXY\_COOKIE\_SECRET valueFrom: secretKeyRef: name: oauth2-proxy-secret key: cookie-secret**

apiVersion: v1 kind: Service metadata: name: oauth2-proxy spec: selector: app: oauth2-proxy ports:

* name: http port: 4180 targetPort: 4180