# Prototype Roadmap – AI‑Driven Blockchain‑Backed Database Security System

Target order (per your plan): 1) **Web Dashboard Prototype** 2) **Blockchain Backend** 3) **AI Module**

For **this sprint**, we will deliver a working **Flask + PostgreSQL backend skeleton** that the React dashboard can consume immediately, while stubbing Blockchain + AI endpoints for later.

## 0) Prerequisites

* Python 3.11+
* Node 18+
* Docker + Docker Compose
* PostgreSQL 14+ (local or via Docker)

## 1) Monorepo Layout

ai-blockchain-secure-db/  
├── backend/  
│ ├── app/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── config.py  
│ │ ├── models.py  
│ │ ├── routes/  
│ │ │ ├── auth\_routes.py  
│ │ │ ├── log\_routes.py  
│ │ │ ├── blockchain\_routes.py  
│ │ │ └── ai\_routes.py  
│ │ └── utils/  
│ │ ├── security.py  
│ │ ├── hashing.py  
│ │ └── db\_init.py  
│ ├── migrations/  
│ │ └── 0001\_init.sql  
│ ├── tests/  
│ │ ├── test\_auth.py  
│ │ └── test\_logs.py  
│ ├── requirements.txt  
│ ├── run.py  
│ ├── Dockerfile  
│ └── gunicorn.conf.py  
├── frontend/  
│ ├── src/  
│ │ ├── App.jsx  
│ │ ├── main.jsx  
│ │ ├── pages/  
│ │ │ ├── Login.jsx  
│ │ │ └── Dashboard.jsx  
│ │ ├── components/  
│ │ │ ├── AlertFeed.jsx  
│ │ │ ├── Charts/  
│ │ │ │ ├── AlertFrequencyChart.jsx  
│ │ │ │ └── ConfidenceChart.jsx  
│ │ └── services/api.js  
│ ├── index.html  
│ ├── package.json  
│ └── vite.config.js  
├── docker-compose.yml  
├── .env.example  
├── Makefile  
└── README.md

## 2) Backend – Core Code (Flask + SQLAlchemy + JWT)

### backend/app/\_\_init\_\_.py

from flask import Flask  
from flask\_sqlalchemy import SQLAlchemy  
from flask\_jwt\_extended import JWTManager  
from flask\_cors import CORS  
  
# Globals  
db = SQLAlchemy()  
jwt = JWTManager()  
  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
 CORS(app)  
 app.config.from\_object("app.config.Config")  
  
 db.init\_app(app)  
 jwt.init\_app(app)  
  
 # Blueprints  
 from app.routes.auth\_routes import auth\_bp  
 from app.routes.log\_routes import log\_bp  
 from app.routes.blockchain\_routes import blockchain\_bp  
 from app.routes.ai\_routes import ai\_bp  
  
 app.register\_blueprint(auth\_bp, url\_prefix="/auth")  
 app.register\_blueprint(log\_bp, url\_prefix="/logs")  
 app.register\_blueprint(blockchain\_bp, url\_prefix="/blockchain")  
 app.register\_blueprint(ai\_bp, url\_prefix="/ai")  
  
 return app

### backend/app/config.py

import os  
  
class Config:  
 SQLALCHEMY\_DATABASE\_URI = os.getenv(  
 "DATABASE\_URL",  
 "postgresql+psycopg2://postgres:postgres@db:5432/securitydb"  
 )  
 SQLALCHEMY\_TRACK\_MODIFICATIONS = False  
 SECRET\_KEY = os.getenv("SECRET\_KEY", "supersecret")  
 JWT\_SECRET\_KEY = os.getenv("JWT\_SECRET\_KEY", "jwtsecret")  
 BCRYPT\_ROUNDS = int(os.getenv("BCRYPT\_ROUNDS", 12))

### backend/app/models.py

from app import db  
from datetime import datetime  
import uuid  
  
class User(db.Model):  
 \_\_tablename\_\_ = "users"  
 user\_id = db.Column(db.Integer, primary\_key=True)  
 email = db.Column(db.String(255), unique=True, nullable=False)  
 password\_hash = db.Column(db.Text, nullable=False)  
 role = db.Column(db.String(50), nullable=False)  
 mfa\_enabled = db.Column(db.Boolean, default=True)  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
  
class QueryLog(db.Model):  
 \_\_tablename\_\_ = "query\_logs"  
 log\_id = db.Column(db.Integer, primary\_key=True)  
 user\_id = db.Column(db.Integer, db.ForeignKey("users.user\_id"))  
 query\_text = db.Column(db.Text, nullable=False)  
 operation\_type = db.Column(db.String(10))  
 executed\_at = db.Column(db.DateTime, default=datetime.utcnow)  
 client\_ip = db.Column(db.String(45))  
 session\_id = db.Column(db.String(36), default=lambda: str(uuid.uuid4()))  
 blockchain\_hash = db.Column(db.Text)  
  
class Anomaly(db.Model):  
 \_\_tablename\_\_ = "anomalies"  
 anomaly\_id = db.Column(db.Integer, primary\_key=True)  
 log\_id = db.Column(db.Integer, db.ForeignKey("query\_logs.log\_id"))  
 anomaly\_score = db.Column(db.Float, nullable=False)  
 is\_anomalous = db.Column(db.Boolean, nullable=False)  
 model\_version = db.Column(db.String(50))  
 flagged\_at = db.Column(db.DateTime, default=datetime.utcnow)  
  
class Alert(db.Model):  
 \_\_tablename\_\_ = "alerts"  
 alert\_id = db.Column(db.Integer, primary\_key=True)  
 anomaly\_id = db.Column(db.Integer, db.ForeignKey("anomalies.anomaly\_id"))  
 alert\_type = db.Column(db.String(100))  
 confidence = db.Column(db.Float)  
 status = db.Column(db.String(20), default="Open")  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
 resolved\_at = db.Column(db.DateTime)  
  
class BlockchainLog(db.Model):  
 \_\_tablename\_\_ = "blockchain\_logs"  
 blockchain\_id = db.Column(db.Integer, primary\_key=True)  
 log\_id = db.Column(db.Integer, db.ForeignKey("query\_logs.log\_id"))  
 block\_hash = db.Column(db.Text, nullable=False)  
 block\_index = db.Column(db.Integer)  
 transaction\_id = db.Column(db.Text)  
 committed\_at = db.Column(db.DateTime, default=datetime.utcnow)  
  
class ModelConfig(db.Model):  
 \_\_tablename\_\_ = "model\_config"  
 config\_id = db.Column(db.Integer, primary\_key=True)  
 param\_name = db.Column(db.String(100))  
 param\_value = db.Column(db.Text)  
 updated\_at = db.Column(db.DateTime, default=datetime.utcnow)

### backend/app/routes/auth\_routes.py

from flask import Blueprint, request, jsonify  
from werkzeug.security import generate\_password\_hash, check\_password\_hash  
from flask\_jwt\_extended import create\_access\_token  
from app.models import User  
from app import db  
  
auth\_bp = Blueprint("auth", \_\_name\_\_)  
  
@auth\_bp.route("/register", methods=["POST"])  
def register():  
 data = request.get\_json()  
 email = data.get("email")  
 password = data.get("password")  
 role = data.get("role", "viewer")  
  
 if User.query.filter\_by(email=email).first():  
 return jsonify({"error": "Email already registered"}), 400  
  
 user = User(email=email, password\_hash=generate\_password\_hash(password), role=role)  
 db.session.add(user)  
 db.session.commit()  
 return jsonify({"message": "User created"}), 201  
  
@auth\_bp.route("/login", methods=["POST"])  
def login():  
 data = request.get\_json()  
 user = User.query.filter\_by(email=data.get("email")).first()  
 if not user or not check\_password\_hash(user.password\_hash, data.get("password")):  
 return jsonify({"error": "Invalid credentials"}), 401  
 token = create\_access\_token(identity={"email": user.email, "role": user.role})  
 return jsonify({"access\_token": token, "role": user.role}), 200

### backend/app/routes/log\_routes.py

from flask import Blueprint, request, jsonify  
from flask\_jwt\_extended import jwt\_required  
from app.models import QueryLog, Alert  
from app import db  
  
log\_bp = Blueprint("logs", \_\_name\_\_)  
  
@log\_bp.route("/", methods=["GET"])  
@jwt\_required(optional=True)  
def list\_logs():  
 logs = QueryLog.query.order\_by(QueryLog.executed\_at.desc()).limit(100).all()  
 return jsonify([  
 {  
 "id": l.log\_id,  
 "user\_id": l.user\_id,  
 "query": l.query\_text,  
 "operation": l.operation\_type,  
 "time": l.executed\_at.isoformat(),  
 "hash": l.blockchain\_hash,  
 } for l in logs  
 ])  
  
@log\_bp.route("/add", methods=["POST"])  
@jwt\_required(optional=True)  
def add\_log():  
 data = request.get\_json()  
 log = QueryLog(  
 user\_id=data.get("user\_id"),  
 query\_text=data.get("query\_text"),  
 operation\_type=data.get("operation\_type"),  
 client\_ip=request.remote\_addr,  
 )  
 db.session.add(log)  
 db.session.commit()  
 return jsonify({"id": log.log\_id}), 201  
  
@log\_bp.route("/alerts", methods=["GET"])  
@jwt\_required(optional=True)  
def list\_alerts():  
 alerts = Alert.query.order\_by(Alert.created\_at.desc()).limit(100).all()  
 return jsonify([  
 {  
 "id": a.alert\_id,  
 "type": a.alert\_type,  
 "confidence": a.confidence,  
 "status": a.status,  
 "created\_at": a.created\_at.isoformat(),  
 } for a in alerts  
 ])

### backend/app/routes/blockchain\_routes.py (stub for Phase 2)

from flask import Blueprint, request, jsonify  
from app.models import QueryLog, BlockchainLog  
from app import db  
  
blockchain\_bp = Blueprint("blockchain", \_\_name\_\_)  
  
@blockchain\_bp.route("/commit", methods=["POST"])  
def commit\_hash():  
 data = request.get\_json()  
 # TODO: compute SHA-256, call Fabric SDK, return tx details  
 ql = QueryLog.query.get(data.get("log\_id"))  
 if not ql:  
 return jsonify({"error": "log not found"}), 404  
  
 # Placeholder commit  
 block\_hash = "pending"  
 tx\_id = "pending"  
  
 b = BlockchainLog(log\_id=ql.log\_id, block\_hash=block\_hash, transaction\_id=tx\_id)  
 db.session.add(b)  
 db.session.commit()  
  
 return jsonify({"message": "queued", "blockchain\_log\_id": b.blockchain\_id}), 202  
  
@blockchain\_bp.route("/verify/<int:log\_id>", methods=["GET"])  
def verify(log\_id):  
 # TODO: verify against Fabric network  
 entry = BlockchainLog.query.filter\_by(log\_id=log\_id).first()  
 if not entry:  
 return jsonify({"verified": False, "reason": "no record"}), 404  
 return jsonify({"verified": entry.block\_hash != "pending", "tx": entry.transaction\_id})

### backend/app/routes/ai\_routes.py (stub for Phase 3)

from flask import Blueprint, request, jsonify  
  
ai\_bp = Blueprint("ai", \_\_name\_\_)  
  
@ai\_bp.route("/score", methods=["POST"])  
def score():  
 # TODO: call anomaly detector service; return score + is\_anomalous  
 payload = request.get\_json()  
 return jsonify({"score": 0.02, "is\_anomalous": False, "model\_version": "v0"})

### backend/app/utils/security.py

import base64  
from Crypto.Cipher import AES  
from Crypto.Random import get\_random\_bytes  
  
# NOTE: For demo purposes; use a KMS in production  
  
def pad(s: bytes) -> bytes:  
 pad\_len = 16 - len(s) % 16  
 return s + bytes([pad\_len]) \* pad\_len  
  
def unpad(s: bytes) -> bytes:  
 pad\_len = s[-1]  
 return s[:-pad\_len]  
  
def aes\_encrypt(plaintext: str, key: bytes) -> str:  
 iv = get\_random\_bytes(16)  
 cipher = AES.new(key, AES.MODE\_CBC, iv)  
 ct = cipher.encrypt(pad(plaintext.encode()))  
 return base64.b64encode(iv + ct).decode()  
  
def aes\_decrypt(token: str, key: bytes) -> str:  
 raw = base64.b64decode(token)  
 iv, ct = raw[:16], raw[16:]  
 cipher = AES.new(key, AES.MODE\_CBC, iv)  
 pt = unpad(cipher.decrypt(ct))  
 return pt.decode()

### backend/app/utils/hashing.py

import hashlib  
  
def sha256\_hex(payload: str) -> str:  
 return hashlib.sha256(payload.encode("utf-8")).hexdigest()

### backend/app/utils/db\_init.py

from app import db  
  
def init\_db():  
 db.create\_all()

### backend/run.py

from app import create\_app  
from app.utils.db\_init import init\_db  
  
app = create\_app()  
  
with app.app\_context():  
 init\_db()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="0.0.0.0", port=8000)

### backend/requirements.txt

flask==3.0.3  
flask-cors==4.0.1  
flask-jwt-extended==4.6.0  
flask-sqlalchemy==3.1.1  
psycopg2-binary==2.9.9  
werkzeug==3.0.3  
pycryptodome==3.20.0  
gunicorn==21.2.0

### backend/gunicorn.conf.py

bind = "0.0.0.0:8000"  
workers = 2  
threads = 2  
timeout = 60

## 3) Database Migration (from Appendix 1)

### backend/migrations/0001\_init.sql

CREATE TABLE IF NOT EXISTS users (  
 user\_id SERIAL PRIMARY KEY,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 password\_hash TEXT NOT NULL,  
 role VARCHAR(50) CHECK (role IN ('admin','auditor','viewer')),  
 mfa\_enabled BOOLEAN DEFAULT TRUE,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE IF NOT EXISTS query\_logs (  
 log\_id SERIAL PRIMARY KEY,  
 user\_id INTEGER REFERENCES users(user\_id),  
 query\_text TEXT NOT NULL,  
 operation\_type VARCHAR(10) CHECK (operation\_type IN ('SELECT','INSERT','UPDATE','DELETE')),  
 executed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 client\_ip VARCHAR(45),  
 session\_id UUID,  
 blockchain\_hash TEXT  
);  
  
CREATE TABLE IF NOT EXISTS anomalies (  
 anomaly\_id SERIAL PRIMARY KEY,  
 log\_id INTEGER REFERENCES query\_logs(log\_id),  
 anomaly\_score FLOAT NOT NULL,  
 is\_anomalous BOOLEAN NOT NULL,  
 model\_version VARCHAR(50),  
 flagged\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE IF NOT EXISTS alerts (  
 alert\_id SERIAL PRIMARY KEY,  
 anomaly\_id INTEGER REFERENCES anomalies(anomaly\_id),  
 alert\_type VARCHAR(100),  
 confidence FLOAT,  
 status VARCHAR(20) DEFAULT 'Open',  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 resolved\_at TIMESTAMP  
);  
  
CREATE TABLE IF NOT EXISTS blockchain\_logs (  
 blockchain\_id SERIAL PRIMARY KEY,  
 log\_id INTEGER REFERENCES query\_logs(log\_id),  
 block\_hash TEXT NOT NULL,  
 block\_index INTEGER,  
 transaction\_id TEXT,  
 committed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE IF NOT EXISTS model\_config (  
 config\_id SERIAL PRIMARY KEY,  
 param\_name VARCHAR(100),  
 param\_value TEXT,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

## 4) Docker & Compose

### Dockerfile (backend)

FROM python:3.11-slim  
WORKDIR /app  
COPY requirements.txt ./  
RUN pip install --no-cache-dir -r requirements.txt  
COPY . .  
EXPOSE 8000  
CMD ["gunicorn", "run:app", "-c", "gunicorn.conf.py"]

### docker-compose.yml

version: "3.9"  
services:  
 db:  
 image: postgres:14  
 environment:  
 POSTGRES\_USER: postgres  
 POSTGRES\_PASSWORD: postgres  
 POSTGRES\_DB: securitydb  
 ports:  
 - "5432:5432"  
 volumes:  
 - pgdata:/var/lib/postgresql/data  
  
 api:  
 build: ./backend  
 environment:  
 DATABASE\_URL: postgresql+psycopg2://postgres:postgres@db:5432/securitydb  
 JWT\_SECRET\_KEY: devsecret  
 SECRET\_KEY: devsecret  
 depends\_on:  
 - db  
 ports:  
 - "8000:8000"  
 command: gunicorn run:app -c gunicorn.conf.py  
  
volumes:  
 pgdata:

### .env.example

DATABASE\_URL=postgresql+psycopg2://postgres:postgres@localhost:5432/securitydb  
JWT\_SECRET\_KEY=change\_me  
SECRET\_KEY=change\_me  
BCRYPT\_ROUNDS=12

### Makefile

.PHONY: up down logs fmt test seed  
  
up:  
 docker compose up -d --build  
  
down:  
 docker compose down -v  
  
logs:  
 docker compose logs -f  
  
test:  
 pytest -q  
  
seed:  
 psql postgres://postgres:postgres@localhost:5432/securitydb \  
 -f backend/migrations/0001\_init.sql

## 5) Frontend – Web Dashboard Prototype (files + stubs)

### frontend/src/services/api.js

import axios from "axios";  
const api = axios.create({ baseURL: import.meta.env.VITE\_API\_URL || "http://localhost:8000" });  
  
api.interceptors.request.use((config) => {  
 const token = localStorage.getItem("token");  
 if (token) config.headers.Authorization = `Bearer ${token}`;  
 return config;  
});  
  
export default api;

### frontend/src/pages/Login.jsx

import { useState } from "react";  
import api from "../services/api";  
  
export default function Login() {  
 const [email, setEmail] = useState("");  
 const [password, setPassword] = useState("");  
  
 const onSubmit = async (e) => {  
 e.preventDefault();  
 const { data } = await api.post("/auth/login", { email, password });  
 localStorage.setItem("token", data.access\_token);  
 window.location.href = "/";  
 };  
  
 return (  
 <form onSubmit={onSubmit} className="p-6 max-w-sm mx-auto">  
 <input className="border p-2 w-full mb-2" placeholder="Email" value={email} onChange={(e)=>setEmail(e.target.value)} />  
 <input className="border p-2 w-full mb-4" placeholder="Password" type="password" value={password} onChange={(e)=>setPassword(e.target.value)} />  
 <button className="border px-4 py-2" type="submit">Login</button>  
 </form>  
 );  
}

### frontend/src/pages/Dashboard.jsx

import { useEffect, useState } from "react";  
import api from "../services/api";  
import AlertFrequencyChart from "../components/Charts/AlertFrequencyChart";  
import ConfidenceChart from "../components/Charts/ConfidenceChart";  
  
export default function Dashboard(){  
 const [logs, setLogs] = useState([]);  
 const [alerts, setAlerts] = useState([]);  
  
 useEffect(() => {  
 api.get("/logs/").then(({data}) => setLogs(data));  
 api.get("/logs/alerts").then(({data}) => setAlerts(data));  
 }, []);  
  
 return (  
 <div className="p-6 grid gap-4">  
 <h1 className="text-2xl font-semibold">Security Dashboard</h1>  
 <AlertFrequencyChart data={alerts} />  
 <ConfidenceChart data={alerts} />  
 <div className="border rounded p-4">  
 <h2 className="font-semibold mb-2">Recent Query Logs</h2>  
 <ul className="space-y-1">  
 {logs.map(l => (  
 <li key={l.id} className="text-sm">[{l.operation}] {l.query} <em className="opacity-60">{l.time}</em></li>  
 ))}  
 </ul>  
 </div>  
 </div>  
 );  
}

### frontend/src/components/Charts/AlertFrequencyChart.jsx

import { useMemo } from "react";  
  
export default function AlertFrequencyChart({ data }){  
 // Replace with recharts later; simple placeholder  
 const count = useMemo(() => data.length, [data]);  
 return <div className="border rounded p-4">Alert Count (placeholder): {count}</div>;  
}

### frontend/src/components/Charts/ConfidenceChart.jsx

export default function ConfidenceChart({ data }){  
 const avg = data.length ? (data.reduce((s,a)=>s+(a.confidence||0),0)/data.length).toFixed(2) : 0;  
 return <div className="border rounded p-4">Avg Confidence (placeholder): {avg}</div>;  
}

### frontend/src/App.jsx

import { BrowserRouter, Routes, Route } from "react-router-dom";  
import Login from "./pages/Login";  
import Dashboard from "./pages/Dashboard";  
  
export default function App(){  
 return (  
 <BrowserRouter>  
 <Routes>  
 <Route path="/login" element={<Login/>} />  
 <Route path="/" element={<Dashboard/>} />  
 </Routes>  
 </BrowserRouter>  
 );  
}

## 6) Tests (minimal to start)

### backend/tests/test\_auth.py

from app import create\_app, db  
  
def test\_app\_starts():  
 app = create\_app()  
 with app.app\_context():  
 assert app is not None

### backend/tests/test\_logs.py

from app import create\_app  
  
def test\_logs\_endpoint\_smoke():  
 app = create\_app()  
 client = app.test\_client()  
 resp = client.get('/logs/')  
 assert resp.status\_code == 200

## 7) Run Instructions

**Local (Docker Compose):**

cp .env.example .env  
make up  
make logs

Open API: http://localhost:8000

**Seed DB:**

make seed

**Frontend (after npm create vite@latest):**

cd frontend  
npm install  
npm run dev

## 8) Next Sprints

* **Blockchain Backend (Phase 2):**
  + Implement hashing.sha256\_hex in /blockchain/commit and integrate Fabric SDK; persist block\_hash, transaction\_id, block\_index.
  + Add /blockchain/verify/:log\_id real verification.
* **AI Module (Phase 3):**
  + Create a small Python service or module to score queries with Isolation Forest; call from /ai/score.
  + Store anomalies + generate alerts with confidence.
* **Dashboard polish:**
  + Replace placeholder charts with Recharts; add role-based UI (admin/auditor/viewer); add login guard.