🚀 CREDITX ECOSYSTEM - PRODUCTION CODE (CONTINUED)

🤖 AGENT BACKEND - PYTHON LANGGRAPH

5. Agent Service - Main FastAPI Server

`apps/agent/requirements.txt`

```
text
# Core Framework
fastapi==0.109.0
uvicorn[standard]==0.27.0
pydantic==2.5.3
pydantic-settings==2.1.0

# LangChain & LangGraph
langgraph==0.2.0
langchain==0.1.0
langchain-openai==0.0.5
langchain-community==0.0.17
langsmith==0.0.77

# Database & Cache
psycopg2-binary==2.9.9
redis==5.0.1
sqlalchemy==2.0.25

# ML & AI
openai==1.10.0
anthropic==0.8.1
tiktoken==0.5.2
sentence-transformers==2.3.1
torch==2.1.2
transformers==4.37.0
```

```
# Utilities
httpx==0.26.0
aiohttp==3.9.1
python-dotenv==1.0.0
python-jose[cryptography]==3.3.0
python-multipart==0.0.6

# Monitoring
prometheus-client==0.19.0
structlog==24.1.0

# Testing
pytest==7.4.4
pytest-asyncio==0.23.3
pytest-cov==4.1.0
```

`apps/agent/src/main.py` (FastAPI Server)

```
python
"""
creditX Ecosystem Agent - Main FastAPI Server
Production-ready LangGraph agent with CopilotKit
integration
"""

import os
import logging
from contextlib import asynccontextmanager
from typing import Optional

from fastapi import FastAPI, HTTPException, Depends, Header
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from pydantic import BaseModel, Field
import structlog

from .agents.creditx_agent import CreditXAgent
from .agents.apps_91_agent import Apps91Agent
from .agents.global_ai_alert_agent import
GlobalAIAlertAgent
from .agents.guardian_ai_agent import GuardianAIAgent
```

```python
from .agents.stolen_phones_agent import StolenPhonesAgent
from .utils.database import init_db, close_db
from .utils.redis_client import init_redis, close_redis
from .middleware.auth import verify_agent_token
from .middleware.tenant import set_tenant_context

# Configure structured logging
structlog.configure(
    processors=[
        structlog.stdlib.filter_by_level,
        structlog.stdlib.add_logger_name,
        structlog.stdlib.add_log_level,
        structlog.stdlib.PositionalArgumentsFormatter(),
        structlog.processors.TimeStamper(fmt="iso"),
        structlog.processors.StackInfoRenderer(),
        structlog.processors.format_exc_info,
        structlog.processors.UnicodeDecoder(),
        structlog.processors.JSONRenderer()
    ],
    wrapper_class=structlog.stdlib.BoundLogger,
    logger_factory=structlog.stdlib.LoggerFactory(),
    cache_logger_on_first_use=True,
)

logger = structlog.get_logger()

# Initialize agents globally
agents = {}

@asynccontextmanager
async def lifespan(app: FastAPI):
    """Application lifespan manager"""
    # Startup
    logger.info("Starting creditX Ecosystem Agent")

    await init_db()
    await init_redis()

    # Initialize all module agents
    agents['creditx'] = CreditXAgent()
    agents['91-apps'] = Apps91Agent()
```

```
    agents['global-ai-alert'] = GlobalAIAlertAgent()
    agents['guardian-ai'] = GuardianAIAgent()
    agents['stolen-phones'] = StolenPhonesAgent()

    logger.info("All agents initialized successfully")

    yield

    # Shutdown
    logger.info("Shutting down creditX Ecosystem Agent")
    await close_redis()
    await close_db()

# Create FastAPI app
app = FastAPI(
    title="creditX Ecosystem Agent",
    description="AI-powered multi-module agent system for
PE portfolio management",
    version="1.0.0",
    lifespan=lifespan,
    docs_url="/docs" if os.getenv("ENVIRONMENT") !=
"production" else None,
    redoc_url="/redoc" if os.getenv("ENVIRONMENT") !=
"production" else None,
)

# CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "https://ecosystem.ai",
        "https://*.ecosystem.ai",
        "http://localhost:3000",
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Request/Response Models
class AgentRequest(BaseModel):
```

```
    """Standard agent request"""
    module: str = Field(..., description="Module name:
creditx, 91-apps, etc.")
    action: str = Field(..., description="Action to
perform")
    parameters: dict = Field(default_factory=dict,
description="Action parameters")
    context: Optional[dict] = Field(default=None,
description="Additional context")
    tenant_id: int = Field(..., description="Tenant ID")
    user_id: str = Field(..., description="User ID")

class AgentResponse(BaseModel):
    """Standard agent response"""
    success: bool
    result: Optional[dict] = None
    error: Optional[str] = None
    metadata: Optional[dict] = None

# Health check endpoint
@app.get("/health")
async def health_check():
    """Health check endpoint"""
    return {
        "status": "healthy",
        "version": "1.0.0",
        "agents": list(agents.keys()),
    }

# CopilotKit compatibility endpoint
@app.post("/copilotkit")
async def copilotkit_handler(
    request: AgentRequest,
    authorization: Optional[str] = Header(None),
):
    """
    CopilotKit-compatible endpoint for agent interactions
    Handles all module routing and agent orchestration
    """
    try:
        # Verify authentication
```

```
        if not authorization:
            raise HTTPException(status_code=401,
detail="Missing authorization header")

        # Get appropriate agent
        agent = agents.get(request.module)
        if not agent:
            raise HTTPException(
                status_code=404,
                detail=f"Agent for module
'{request.module}' not found"
            )

        # Set tenant context
        await set_tenant_context(request.tenant_id)

        # Execute agent action
        logger.info(
            "Executing agent action",
            module=request.module,
            action=request.action,
            tenant_id=request.tenant_id,
            user_id=request.user_id,
        )

        result = await agent.execute(
            action=request.action,
            parameters=request.parameters,
            context=request.context,
            tenant_id=request.tenant_id,
            user_id=request.user_id,
        )

        return AgentResponse(
            success=True,
            result=result,
            metadata={
                "module": request.module,
                "action": request.action,
                "execution_time_ms":
result.get("execution_time_ms", 0),
```

```
            }
        )

    except HTTPException:
        raise
    except Exception as e:
        logger.error(
            "Agent execution error",
            error=str(e),
            module=request.module,
            action=request.action,
        )
        return AgentResponse(
            success=False,
            error=str(e),
            metadata={"module": request.module, "action":
request.action}
        )

# Module-specific endpoints for direct invocation
@app.post("/agents/creditx")
async def creditx_agent_handler(request: AgentRequest):
    """CreditX Compliance Agent endpoint"""
    request.module = "creditx"
    return await copilotkit_handler(request)

@app.post("/agents/91-apps")
async def apps_91_agent_handler(request: AgentRequest):
    """91 Apps Business Automation Agent endpoint"""
    request.module = "91-apps"
    return await copilotkit_handler(request)

@app.post("/agents/global-ai-alert")
async def global_ai_alert_agent_handler(request:
AgentRequest):
    """Global AI Alert Network Agent endpoint"""
    request.module = "global-ai-alert"
    return await copilotkit_handler(request)

@app.post("/agents/guardian-ai")
async def guardian_ai_agent_handler(request: AgentRequest):
```

```
    """Guardian AI Endpoint Security Agent endpoint"""
    request.module = "guardian-ai"
    return await copilotkit_handler(request)

@app.post("/agents/stolen-phones")
async def stolen_phones_agent_handler(request:
AgentRequest):
    """Stolen/Lost Phones Recovery Agent endpoint"""
    request.module = "stolen-phones"
    return await copilotkit_handler(request)

# Metrics endpoint for Prometheus
@app.get("/metrics")
async def metrics():
    """Prometheus metrics endpoint"""
    from prometheus_client import generate_latest,
CONTENT_TYPE_LATEST
    from fastapi.responses import Response

    return Response(
        content=generate_latest(),
        media_type=CONTENT_TYPE_LATEST,
    )

if __name__ == "__main__":
    import uvicorn

    uvicorn.run(
        "main:app",
        host="0.0.0.0",
        port=8000,
        reload=os.getenv("ENVIRONMENT") != "production",
        log_level="info",
    )
```

6. CreditX Agent Implementation (LangGraph)

apps/agent/src/agents/creditx_agent.py

```
python
"""
CreditX Compliance Automation Agent
Handles KYC, AML, sanctions screening, and regulatory
reporting
"""

import time
from typing import Dict, Any, Optional, List
from datetime import datetime

from langgraph.graph import StateGraph, END
from langgraph.checkpoint.postgres import PostgresSaver
from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage,
AIMessage, SystemMessage
from pydantic import BaseModel, Field

import structlog

from ..tools.sanctions_screening import
sanctions_screening_tool
from ..tools.kyc_generation import kyc_generation_tool
from ..tools.document_extraction import
document_extraction_tool
from ..utils.database import get_db_connection

logger = structlog.get_logger()

class CreditXState(BaseModel):
    """State management for CreditX agent"""
    messages: List[Any] = Field(default_factory=list)
    tenant_id: int
    user_id: str
    action: str
    parameters: Dict[str, Any] =
Field(default_factory=dict)
    context: Optional[Dict[str, Any]] = None

    # Workflow state
    current_step: str = "init"
```

```
    transaction_data: Optional[Dict] = None
    screening_result: Optional[Dict] = None
    compliance_score: int = 0
    kyc_report: Optional[Dict] = None
    requires_approval: bool = False

    # Results
    result: Optional[Dict] = None
    error: Optional[str] = None

class CreditXAgent:
    """
    CreditX Compliance Agent using LangGraph

    Workflow:
    1. Receive transaction/entity data
    2. Extract and validate information
    3. Perform sanctions screening
    4. Calculate compliance score
    5. Generate reports if needed
    6. Return results with approval requirement
    """

    def __init__(self):
        self.llm = ChatOpenAI(
            model="gpt-4-turbo-preview",
            temperature=0,
            streaming=True,
        )

        # Build LangGraph workflow
        self.workflow = self._build_workflow()

        # Initialize checkpoint saver for persistence
        self.checkpointer = PostgresSaver.from_conn_string(
            conn_string=self._get_db_url()
        )

        # Compile graph with checkpointing
        self.graph =
self.workflow.compile(checkpointer=self.checkpointer)
```

```
        logger.info("CreditX Agent initialized")

    def _get_db_url(self) -> str:
        """Get database URL from environment"""
        import os
        return os.getenv("DATABASE_URL", "")

    def _build_workflow(self) -> StateGraph:
        """Build LangGraph workflow"""
        workflow = StateGraph(CreditXState)

        # Add nodes
        workflow.add_node("validate_input",
self.validate_input)
        workflow.add_node("extract_data",
self.extract_data)
        workflow.add_node("sanctions_screening",
self.sanctions_screening)
        workflow.add_node("calculate_score",
self.calculate_score)
        workflow.add_node("generate_report",
self.generate_report)
        workflow.add_node("check_approval",
self.check_approval)
        workflow.add_node("finalize", self.finalize)

        # Define edges
        workflow.set_entry_point("validate_input")
        workflow.add_edge("validate_input", "extract_data")
        workflow.add_edge("extract_data",
"sanctions_screening")
        workflow.add_edge("sanctions_screening",
"calculate_score")

        # Conditional edge based on action type
        workflow.add_conditional_edges(
            "calculate_score",
            self.should_generate_report,
            {
                "generate": "generate_report",
```

```
                "skip": "check_approval",
            }
        )

        workflow.add_edge("generate_report",
"check_approval")
        workflow.add_edge("check_approval", "finalize")
        workflow.add_edge("finalize", END)

        return workflow

    async def validate_input(self, state: CreditXState) ->
CreditXState:
        """Validate input parameters"""
        logger.info("Validating input",
action=state.action)

        if state.action not in [
            "screen_transaction",
            "generate_kyc",
            "generate_audit_report",
            "check_sanctions"
        ]:
            state.error = f"Unknown action: {state.action}"
            state.current_step = "error"
            return state

        state.current_step = "validated"
        return state

    async def extract_data(self, state: CreditXState) ->
CreditXState:
        """Extract and structure data using LLM"""
        logger.info("Extracting data", action=state.action)

        # Use LLM to extract structured data
        system_message = SystemMessage(
            content="""You are a compliance data extraction
expert.
            Extract structured transaction or entity data
from the provided information.
```

```
            Ensure all required fields are present and
properly formatted."""
        )

        human_message = HumanMessage(
            content=f"Extract compliance data from:
{state.parameters}"
        )

        response = await self.llm.ainvoke([system_message,
human_message])

        # Parse response (simplified - would use structured
output in production)
        state.transaction_data = state.parameters
        state.current_step = "extracted"

        return state

    async def sanctions_screening(self, state:
CreditXState) -> CreditXState:
        """Perform sanctions screening"""
        logger.info("Performing sanctions screening",
tenant_id=state.tenant_id)

        try:
            # Call sanctions screening tool
            screening_result = await
sanctions_screening_tool(

counterparty=state.transaction_data.get("counterparty"),

amount=state.transaction_data.get("amount"),

currency=state.transaction_data.get("currency"),
            )

            state.screening_result = screening_result
            state.current_step = "screened"

        except Exception as e:
```

```
            logger.error("Sanctions screening failed",
error=str(e))
            state.error = f"Screening failed: {str(e)}"
            state.current_step = "error"

        return state

    async def calculate_score(self, state: CreditXState) ->
CreditXState:
        """Calculate compliance score"""
        logger.info("Calculating compliance score")

        score = 100

        if state.screening_result:
            status = state.screening_result.get("status",
"")

            if status == "FLAGGED":
                score -= 50
            elif status == "BLOCKED":
                score = 0

            matches = state.screening_result.get("matches",
[])
            score -= len(matches) * 10

        state.compliance_score = max(0, min(100, score))
        state.current_step = "scored"

        return state

    def should_generate_report(self, state: CreditXState)
-> str:
        """Determine if report generation is needed"""
        if state.action in ["generate_kyc",
"generate_audit_report"]:
            return "generate"
        return "skip"

    async def generate_report(self, state: CreditXState) ->
```

```
CreditXState:
        """Generate compliance report"""
        logger.info("Generating report",
action=state.action)

        try:
            if state.action == "generate_kyc":
                report = await kyc_generation_tool(

entity_id=state.parameters.get("entity_id"),

report_type=state.parameters.get("report_type",
"standard"),
                )
                state.kyc_report = report

            state.current_step = "report_generated"

        except Exception as e:
            logger.error("Report generation failed",
error=str(e))
            state.error = f"Report generation failed:
{str(e)}"

        return state

    async def check_approval(self, state: CreditXState) ->
CreditXState:
        """Check if human approval is required"""
        logger.info("Checking approval requirements")

        # Approval required if:
        # - Sanctions flagged or blocked
        # - Compliance score < 70
        # - High value transaction (> $1M)

        state.requires_approval = (
            state.screening_result and
state.screening_result.get("status") in ["FLAGGED",
"BLOCKED"]
        ) or (
```

```
            state.compliance_score < 70
        ) or (
            state.transaction_data
            and state.transaction_data.get("amount", 0) >
1000000
        )

        state.current_step = "approval_checked"
        return state

    async def finalize(self, state: CreditXState) ->
CreditXState:
        """Finalize and prepare response"""
        logger.info("Finalizing response")

        state.result = {
            "action": state.action,
            "compliance_score": state.compliance_score,
            "sanctions_status":
state.screening_result.get("status") if
state.screening_result else None,
            "requires_approval": state.requires_approval,
            "kyc_report": state.kyc_report,
            "timestamp": datetime.utcnow().isoformat(),
        }

        state.current_step = "completed"
        return state

    async def execute(
        self,
        action: str,
        parameters: Dict[str, Any],
        context: Optional[Dict[str, Any]],
        tenant_id: int,
        user_id: str,
    ) -> Dict[str, Any]:
        """Execute agent workflow"""
        start_time = time.time()

        try:
```

```
            # Create initial state
            initial_state = CreditXState(
                tenant_id=tenant_id,
                user_id=user_id,
                action=action,
                parameters=parameters,
                context=context,
            )

            # Execute graph
            final_state = await self.graph.ainvoke(
                initial_state.dict(),
                config={
                    "configurable": {
                        "thread_id": f"{tenant_id}
_{user_id}_{int(time.time())}",
                    }
                }
            )

            execution_time = (time.time() - start_time) *
1000

            if final_state.get("error"):
                return {
                    "success": False,
                    "error": final_state["error"],
                    "execution_time_ms": execution_time,
                }

            return {
                "success": True,
                **final_state["result"],
                "execution_time_ms": execution_time,
            }

        except Exception as e:
            logger.error("Agent execution failed",
error=str(e))
            return {
                "success": False,
```

```
                "error": str(e),
                "execution_time_ms": (time.time() -
start_time) * 1000,
            }
```

7. Database Package - Migrations

packages/database/package.json

```
json
{
  "name": "@creditx/database",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "migrate": "prisma migrate deploy",
    "migrate:dev": "prisma migrate dev",
    "migrate:create": "prisma migrate dev --create-only",
    "seed": "tsx seed.ts",
    "studio": "prisma studio",
    "generate": "prisma generate"
  },
  "dependencies": {
    "@prisma/client": "^5.8.0"
  },
  "devDependencies": {
    "prisma": "^5.8.0",
    "tsx": "^4.7.0"
  }
}
```

packages/database/schema.prisma

```
text
// creditX Ecosystem - Database Schema
// Multi-tenant PostgreSQL with Row-Level Security

generator client {
  provider = "prisma-client-js"
```

```
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

//
============================================================
=================
// CORE MULTI-TENANCY
//
============================================================
=================

model Tenant {
  id           Int       @id @default(autoincrement())
  externalId   String    @unique @default(uuid())
  name         String
  domain       String    @unique
  schemaName   String    @unique @map("schema_name")
  status       String    @default("active")
  modulesEnabled String[] @map("modules_enabled")
  settings     Json      @default("{}")
  createdAt    DateTime  @default(now()) @map("created_at")
  updatedAt    DateTime  @updatedAt @map("updated_at")

  users        User[]
  transactions Transaction[]
  leads        Lead[]
  endpoints    Endpoint[]
  devices      Device[]

  @@map("tenants")
}

model User {
  id              String    @id @default(uuid())
  tenantId        Int       @map("tenant_id")
  email           String    @unique
  name            String?
```

```
  role            String    @default("user")
  authProvider    String    @map("auth_provider")
  authProviderId  String?   @map("auth_provider_id")
  permissions     Json       @default("{}")
  lastLoginAt     DateTime? @map("last_login_at")
  createdAt       DateTime  @default(now())
@map("created_at")

  tenant          Tenant    @relation(fields: [tenantId],
references: [id])
  auditLogs       AuditLog[]

  @@map("users")
}

//
================================================================
================
// CREDITX COMPLIANCE MODULE
//
================================================================
================

model Transaction {
  id                String    @id @default(uuid())
  tenantId          Int       @map("tenant_id")
  transactionDate   DateTime  @map("transaction_date")
  amount            Decimal   @db.Decimal(15, 2)
  currency          String    @db.VarChar(3)
  counterparty      String
  description       String?
  sanctionsStatus   String    @map("sanctions_status")
  complianceScore   Int       @map("compliance_score")
  kycDocumentUrl    String?   @map("kyc_document_url")
  metadata          Json      @default("{}")
  createdAt         DateTime  @default(now())
@map("created_at")
  updatedAt         DateTime  @updatedAt @map("updated_at")

  tenant            Tenant    @relation(fields: [tenantId],
references: [id])
```

```
  approvals         ApprovalWorkflow[]

  @@index([tenantId, sanctionsStatus])
  @@index([tenantId, transactionDate])
  @@map("transactions")
}

model AuditLog {
  id           String    @id @default(uuid())
  tenantId     Int       @map("tenant_id")
  action       String
  userId       String    @map("user_id")
  resourceType String    @map("resource_type")
  resourceId   String    @map("resource_id")
  changes      Json      @default("{}")
  ipAddress    String    @map("ip_address")
  timestamp    DateTime  @default(now())

  user         User      @relation(fields: [userId],
references: [id])

  @@index([tenantId, timestamp])
  @@index([resourceType, resourceId])
  @@map("audit_logs")
}

//
============================================================
=================
// 91 APPS AUTOMATION MODULE
//
============================================================
=================

model Lead {
  id              String    @id @default(uuid())
  tenantId        Int       @map("tenant_id")
  externalId      String?   @map("external_id")
  name            String
  email           String
  company         String
```

```
  status          String    @default("new")
  score           Int       @default(0)
  lastActivityAt  DateTime? @map("last_activity_at")
  assignedTo      String?   @map("assigned_to")
  metadata        Json      @default("{}")
  createdAt       DateTime  @default(now())
@map("created_at")
  updatedAt       DateTime  @updatedAt @map("updated_at")

  tenant          Tenant    @relation(fields: [tenantId],
references: [id])
  activities      LeadActivity[]

  @@index([tenantId, status])
  @@index([tenantId, score])
  @@map("leads")
}

model LeadActivity {
  id          String    @id @default(uuid())
  leadId      String    @map("lead_id")
  activityType String   @map("activity_type")
  description String
  metadata    Json      @default("{}")
  createdAt   DateTime  @default(now()) @map("created_at")

  lead        Lead      @relation(fields: [leadId],
references: [id])

  @@index([leadId, createdAt])
  @@map("lead_activities")
}

model AutomationWorkflow {
  id              String    @id @default(uuid())
  tenantId        Int       @map("tenant_id")
  workflowType    String    @map("workflow_type")
  triggerEvent    String    @map("trigger_event")
  conditions      Json      @default("{}")
  actions         Json      @default("{}")
  status          String    @default("active")
```

```
  executionCount  Int       @default(0)
@map("execution_count")
  lastExecutedAt  DateTime? @map("last_executed_at")
  createdAt       DateTime  @default(now())
@map("created_at")

  executions      WorkflowExecution[]

  @@index([tenantId, status])
  @@map("automation_workflows")
}

model WorkflowExecution {
  id           String    @id @default(uuid())
  workflowId   String    @map("workflow_id")
  tenantId     Int       @map("tenant_id")
  inputData    Json      @map("input_data")
  outputData   Json?     @map("output_data")
  status       String    @default("pending")
  errorMessage String?   @map("error_message")
  durationMs   Int?      @map("duration_ms")
  executedAt   DateTime  @default(now()) @map("executed_at")

  workflow     AutomationWorkflow @relation(fields:
[workflowId], references: [id])

  @@index([workflowId, executedAt])
  @@map("workflow_executions")
}

//
============================================================
================
// GLOBAL AI ALERT MODULE
//
============================================================
================

model ThreatIntelligence {
  id             String    @id @default(uuid())
  tenantId       Int       @map("tenant_id")
```

```
  sourceIp      String    @map("source_ip")
  destIp        String    @map("dest_ip")
  dnsQuery      String?   @map("dns_query")
  packetMetadata Json     @default("{}")
@map("packet_metadata")
  threatType    String    @map("threat_type")
  threatScore   Int       @map("threat_score")
  severity      String
  detectedAt    DateTime  @default(now())
@map("detected_at")
  resolvedAt    DateTime? @map("resolved_at")
  resolution    String?

  @@index([tenantId, detectedAt])
  @@index([threatScore, severity])
  @@map("threat_intelligence")
}

model NetworkDevice {
  id              String    @id @default(uuid())
  tenantId        Int       @map("tenant_id")
  deviceType      String    @map("device_type")
  macAddress      String    @map("mac_address")
  ipAddress       String    @map("ip_address")
  hostname        String?
  baselineProfile Json      @default("{}")
@map("baseline_profile")
  lastSeenAt      DateTime? @map("last_seen_at")
  createdAt       DateTime  @default(now())
@map("created_at")

  @@index([tenantId, deviceType])
  @@map("network_devices")
}

//
============================================================
================
// GUARDIAN AI ENDPOINT SECURITY MODULE
//
============================================================
```

```
=================

model Endpoint {
  id                  String    @id @default(uuid())
  tenantId            Int       @map("tenant_id")
  deviceId            String    @unique @map("device_id")
  deviceType          String    @map("device_type")
  osVersion           String    @map("os_version")
  agentVersion        String    @map("agent_version")
  lastCheckinAt       DateTime? @map("last_checkin_at")
  status              String    @default("online")
  baselineEstablished Boolean   @default(false)
@map("baseline_established")
  baselineData        Json      @default("{}")
@map("baseline_data")
  createdAt           DateTime  @default(now())
@map("created_at")

  tenant              Tenant    @relation(fields:
[tenantId], references: [id])
  events              EndpointEvent[]
  incidents           Incident[]

  @@index([tenantId, status])
  @@map("endpoints")
}

model EndpointEvent {
  id           String    @id @default(uuid())
  endpointId   String    @map("endpoint_id")
  tenantId     Int       @map("tenant_id")
  eventType    String    @map("event_type")
  eventData    Json      @map("event_data")
  anomalyScore Int       @map("anomaly_score")
  flagged      Boolean   @default(false)
  timestamp    DateTime  @default(now())

  endpoint     Endpoint  @relation(fields: [endpointId],
references: [id])

  @@index([endpointId, timestamp])
```

```
  @@index([tenantId, flagged])
  @@map("endpoint_events")
}

model Incident {
  id              String    @id @default(uuid())
  endpointId      String    @map("endpoint_id")
  tenantId        Int       @map("tenant_id")
  incidentType    String    @map("incident_type")
  severity        String
  description     String
  status          String    @default("open")
  isolatedAt      DateTime? @map("isolated_at")
  resolvedAt      DateTime? @map("resolved_at")
  resolutionNotes String?   @map("resolution_notes")
  createdAt       DateTime  @default(now())
@map("created_at")

  endpoint        Endpoint  @relation(fields: [endpointId],
references: [id])

  @@index([tenantId, status])
  @@map("incidents")
}

//
============================================================
================
// STOLEN/LOST PHONES MODULE
//
============================================================
================

model Device {
  id                String    @id @default(uuid())
  tenantId          Int       @map("tenant_id")
  deviceId          String    @unique @map("device_id")
  ownerUserId       String    @map("owner_user_id")
  deviceType        String    @map("device_type")
  osVersion         String    @map("os_version")
  status            String    @default("active")
```

```
  lastLocation      Json?     @map("last_location")
  lastLocationAt    DateTime? @map("last_location_at")
  stolenAt          DateTime? @map("stolen_at")
  recoveredAt       DateTime? @map("recovered_at")
  insuranceClaimId  String?   @map("insurance_claim_id")
  createdAt         DateTime  @default(now())
@map("created_at")

  tenant            Tenant    @relation(fields: [tenantId],
references: [id])
  locationHistory   LocationHistory[]
  recoveryWorkflows RecoveryWorkflow[]

  @@index([tenantId, status])
  @@map("devices")
}

model LocationHistory {
  id             String    @id @default(uuid())
  deviceId       String    @map("device_id")
  tenantId       Int       @map("tenant_id")
  location       Json
  accuracyMeters Int       @map("accuracy_meters")
  locationMethod String    @map("location_method")
  timestamp      DateTime  @default(now())

  device         Device    @relation(fields: [deviceId],
references: [id])

  @@index([deviceId, timestamp])
  @@map("location_history")
}

model RecoveryWorkflow {
  id                  String    @id @default(uuid())
  deviceId            String    @map("device_id")
  tenantId            Int       @map("tenant_id")
  workflowStatus      String    @map("workflow_status")
  playbookActions     Json      @map("playbook_actions")
  authoritiesNotified  Boolean   @default(false)
@map("authorities_notified")
```

```
  insuranceClaimFiled  Boolean   @default(false)
@map("insurance_claim_filed")
  chainOfCustody       Json[]    @map("chain_of_custody")
  createdAt            DateTime  @default(now())
@map("created_at")

  device               Device    @relation(fields:
[deviceId], references: [id])

  @@map("recovery_workflows")
}

//
================================================================
================
// SHARED SERVICES
//
================================================================
================

model ApprovalWorkflow {
  id           String    @id @default(uuid())
  tenantId     Int       @map("tenant_id")
  resourceType String    @map("resource_type")
  resourceId   String    @map("resource_id")
  status       String    @default("pending")
  requestedBy  String    @map("requested_by")
  approvedBy   String?   @map("approved_by")
  metadata     Json      @default("{}")
  requestedAt  DateTime  @default(now())
@map("requested_at")
  respondedAt  DateTime? @map("responded_at")

  transaction  Transaction? @relation(fields: [resourceId],
references: [id])

  @@index([tenantId, status])
  @@map("approval_workflows")
}

model IntegrationConnection {
```

```
  id              String    @id @default(uuid())
  tenantId        Int       @map("tenant_id")
  integrationType String    @map("integration_type")
  credentials     Json      @default("{}") // Encrypted
  settings        Json      @default("{}")
  lastSyncAt      DateTime? @map("last_sync_at")
  status          String    @default("active")
  createdAt       DateTime  @default(now())
@map("created_at")

  syncLogs        IntegrationSyncLog[]

  @@index([tenantId, integrationType])
  @@map("integration_connections")
}

model IntegrationSyncLog {
  id                String    @id @default(uuid())
  connectionId      String    @map("connection_id")
  syncType          String    @map("sync_type")
  recordsProcessed  Int       @map("records_processed")
  errors            Int       @default(0)
  durationMs        Int       @map("duration_ms")
  startedAt         DateTime  @map("started_at")
  completedAt       DateTime? @map("completed_at")

  connection        IntegrationConnection @relation(fields:
[connectionId], references: [id])

  @@index([connectionId, startedAt])
  @@map("integration_sync_logs")
}
```

Continue next files with:

- ✅ Docker Configuration
- ✅ CI/CD Pipeline (GitHub Actions)
- ✅ Terraform Infrastructure

- ✅ Additional API Endpoints
- ✅ Integration Connectors