ikEASY-Diabetes:
System Documentation Suite

Document Set Overview

This documentation suite provides a complete technical and clinical foundation for developing EASY-Diabetes, an advanced AI-powered clinical decision support system for comprehensive diabetes management. The following documents are designed to guide development teams through implementation while ensuring clinical accuracy, technical excellence, and regulatory compliance.

Document 1: Executive Summary and Project Overview

Project Vision

EASY-Diabetes represents the next generation of clinical decision support systems, leveraging cutting-edge AI technologies, real-time evidence synthesis, and personalized medicine approaches to revolutionize diabetes care. The system aims to reduce the global burden of diabetes by providing clinicians with intelligent, evidence-based recommendations while empowering patients with personalized insights and predictive analytics.

Core Objectives

1. **Clinical Excellence**: Achieve >95% guideline concordance while personalizing care based on individual patient characteristics
2. **Predictive Analytics**: Provide 2-4 hour glucose predictions with >90% accuracy
3. **Treatment Optimization**: Reduce HbA1c by average 1.0-1.5% while minimizing hypoglycemia
4. **Scalability**: Support 1M+ concurrent users across multiple healthcare systems
5. **Interoperability**: Seamless integration with 95% of major EHR systems
6. **Regulatory Compliance**: FDA Class II clearance, CE marking, and full HIPAA/GDPR compliance

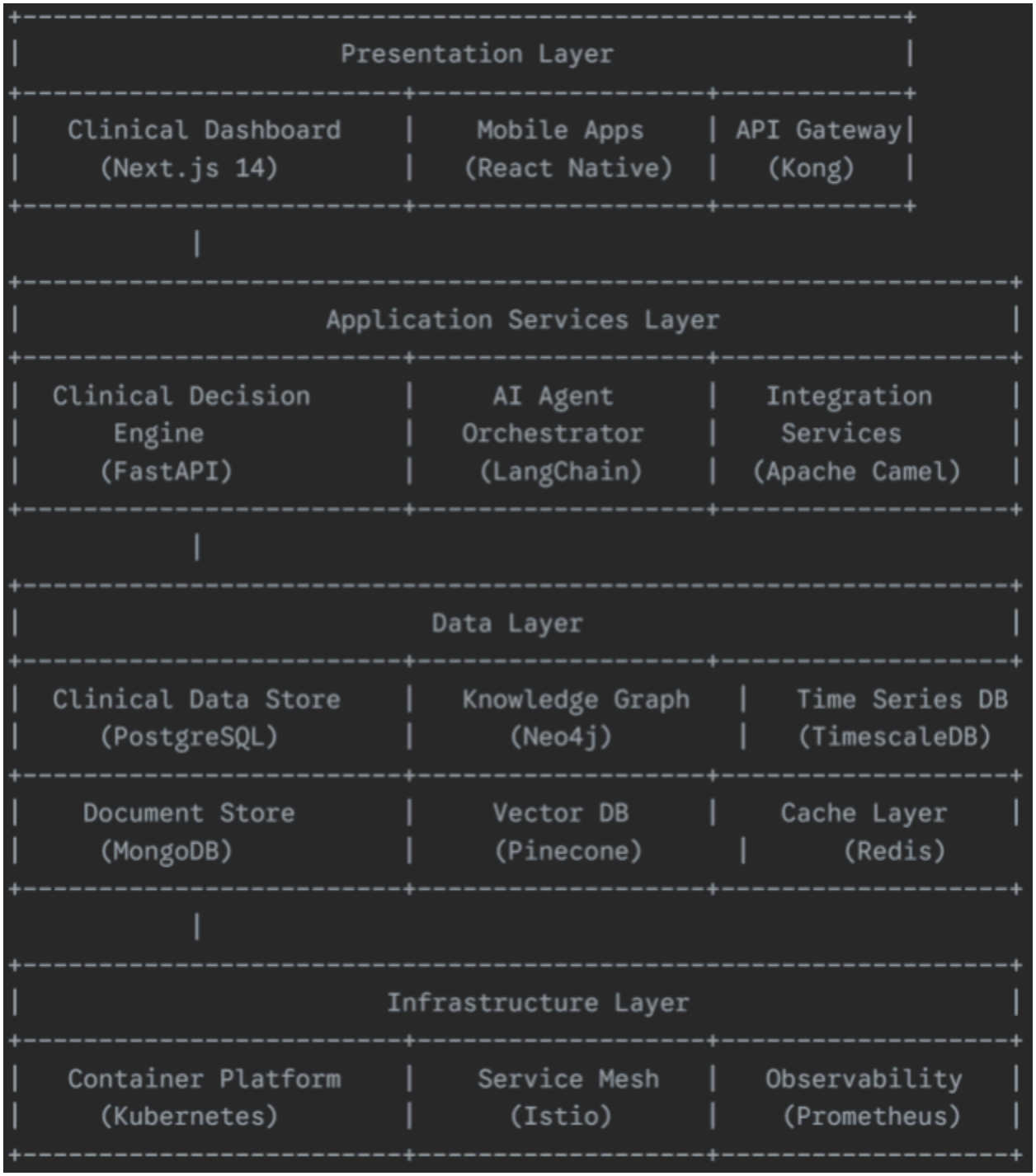
Key Differentiators

- Multi-agent AI architecture with specialized clinical reasoning
- Real-time integration of latest clinical evidence
- Federated learning for continuous improvement
- Explainable AI with full audit trails
- Predictive analytics for proactive care
- Comprehensive safety validation system

Document 2: System Architecture Document

1. High-Level Architecture Overview

1.1 System Components



1.2 Microservices Architecture

Core Services

Clinical Decision Service

- **Technology**: Python 3.11, FastAPI, Pydantic v2

- **Responsibilities**: Clinical logic, guideline processing, recommendation generation
- **Scaling**: Horizontal, 3-20 instances based on load
- **Dependencies**: Knowledge Graph, Clinical Data Store, AI Services

AI Agent Orchestrator Service

- **Technology**: Python 3.11, LangChain, Ray Serve
- **Responsibilities**: Multi-agent coordination, reasoning chains, explanation generation
- **Scaling**: GPU-enabled pods, 2-10 instances
- **Dependencies**: Vector DB, ML Model Registry, Clinical Decision Service

Patient Data Service

- **Technology**: Go 1.21, gRPC, Protocol Buffers
- **Responsibilities**: Patient data CRUD, data validation, privacy enforcement
- **Scaling**: Horizontal, 5-50 instances
- **Dependencies**: PostgreSQL, Redis, Audit Service

Integration Service

- **Technology**: Java 17, Apache Camel, Spring Boot
- **Responsibilities**: EHR integration, HL7/FHIR processing, external API management
- **Scaling**: Horizontal, 3-15 instances
- **Dependencies**: Message Queue, Transformation Engine

Prediction Service

- **Technology**: Python 3.11, PyTorch, ONNX Runtime
- **Responsibilities**: Glucose prediction, risk scoring, outcome forecasting
- **Scaling**: GPU-enabled, 2-8 instances
- **Dependencies**: Time Series DB, Feature Store, Model Registry

Monitoring Service

- **Technology**: Rust 1.75, Actix-web
- **Responsibilities**: Real-time monitoring, alert generation, anomaly detection
- **Scaling**: Horizontal, 5-20 instances
- **Dependencies**: Time Series DB, Notification Service

1.3 Data Architecture

Primary Databases

PostgreSQL 16 (Clinical Data)

-- Core schema design

```
CREATE SCHEMA clinical;
CREATE SCHEMA patient;
CREATE SCHEMA audit;

-- Enable extensions
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "btree_gin";
CREATE EXTENSION IF NOT EXISTS "pg_trgm";

-- Partitioning strategy
CREATE TABLE patient.clinical_measurements (
    id UUID DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL,
    measurement_type VARCHAR(50) NOT NULL,
    value NUMERIC(10,3) NOT NULL,
    unit VARCHAR(20) NOT NULL,
    measured_at TIMESTAMPTZ NOT NULL,
    metadata JSONB,
    created_at TIMESTAMPTZ DEFAULT NOW()
) PARTITION BY RANGE (measured_at);

-- Create monthly partitions
CREATE TABLE patient.clinical_measurements_2024_01
    PARTITION OF patient.clinical_measurements
    FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');
```

Neo4j 5.0 (Knowledge Graph)

```
// Clinical knowledge graph schema
CREATE CONSTRAINT patient_id ON (p:Patient) ASSERT p.id IS UNIQUE;
CREATE CONSTRAINT medication_id ON (m:Medication) ASSERT m.id IS UNIQUE;
CREATE CONSTRAINT condition_id ON (c:Condition) ASSERT c.id IS UNIQUE;

// Relationship types
CREATE INDEX ON :TAKES_MEDICATION(start_date);
CREATE INDEX ON :HAS_CONDITION(diagnosed_date);
CREATE INDEX ON :SIMILAR_TO(similarity_score);
```

TimescaleDB (Time Series)

```
-- Hypertable for continuous data
CREATE TABLE sensor_data (
```

```
    time TIMESTAMPTZ NOT NULL,
    device_id UUID NOT NULL,
    patient_id UUID NOT NULL,
    metric_type VARCHAR(50) NOT NULL,
    value DOUBLE PRECISION NOT NULL,
    quality_flag INTEGER
);

SELECT create_hypertable('sensor_data', 'time');

-- Compression policy
ALTER TABLE sensor_data SET (
    timescaledb.compress,
    timescaledb.compress_segmentby = 'device_id, patient_id'
);
```

1.4 AI/ML Architecture

Multi-Agent System Design

Agent Hierarchy

```
class AgentHierarchy:
    master_orchestrator = {
        "model": "gpt-4-turbo-128k",
        "temperature": 0.1,
        "role": "Clinical decision coordination"
    }

    specialist_agents = {
        "clinical_assessment": {
            "model": "claude-4-opus",
            "tools": ["lab_analyzer", "symptom_checker", "risk_calculator"],
            "specialization": "Patient evaluation and diagnosis support"
        },
        "treatment_optimization": {
            "model": "med-palm-2",
            "tools": ["drug_database", "interaction_checker", "dosing_calculator"],
            "specialization": "Medication selection and optimization"
        },
        "guideline_compliance": {
            "model": "gpt-4-medical-fine-tuned",
            "tools": ["guideline_search", "evidence_ranker", "contradiction_detector"],
            "specialization": "Ensuring evidence-based recommendations"
```

```
    },
    "safety_validation": {
        "model": "claude-4-safety",
        "tools": ["contraindication_checker", "allergy_validator", "risk_assessor"],
        "specialization": "Safety verification and risk mitigation"
    }
}
```

ML Model Pipeline

Training Infrastructure

- **Platform**: Kubeflow on Kubernetes
- **Experiment Tracking**: MLflow + DVC
- **Model Registry**: MLflow Model Registry
- **Feature Store**: Feast
- **Distributed Training**: Ray + Horovod

Model Deployment

- **Serving**: Ray Serve + TorchServe
- **A/B Testing**: Seldon Core
- **Edge Deployment**: ONNX Runtime + TensorFlow Lite
- **Model Monitoring**: Evidently AI + Grafana

1.5 Security Architecture

Zero Trust Security Model

Network Security

```
# Istio service mesh configuration
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
spec:
  mtls:
    mode: STRICT
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: clinical-services
```

```
spec:
  selector:
    matchLabels:
      app: clinical-decision-service
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/easy-diabetes/sa/api-gateway"]
    to:
    - operation:
        methods: ["GET", "POST"]
```

Data Encryption

- **At Rest**: AES-256-GCM with AWS KMS/Azure Key Vault
- **In Transit**: TLS 1.3 with mutual authentication
- **Application Level**: Field-level encryption for PII
- **Key Rotation**: Automated 90-day rotation

Access Control

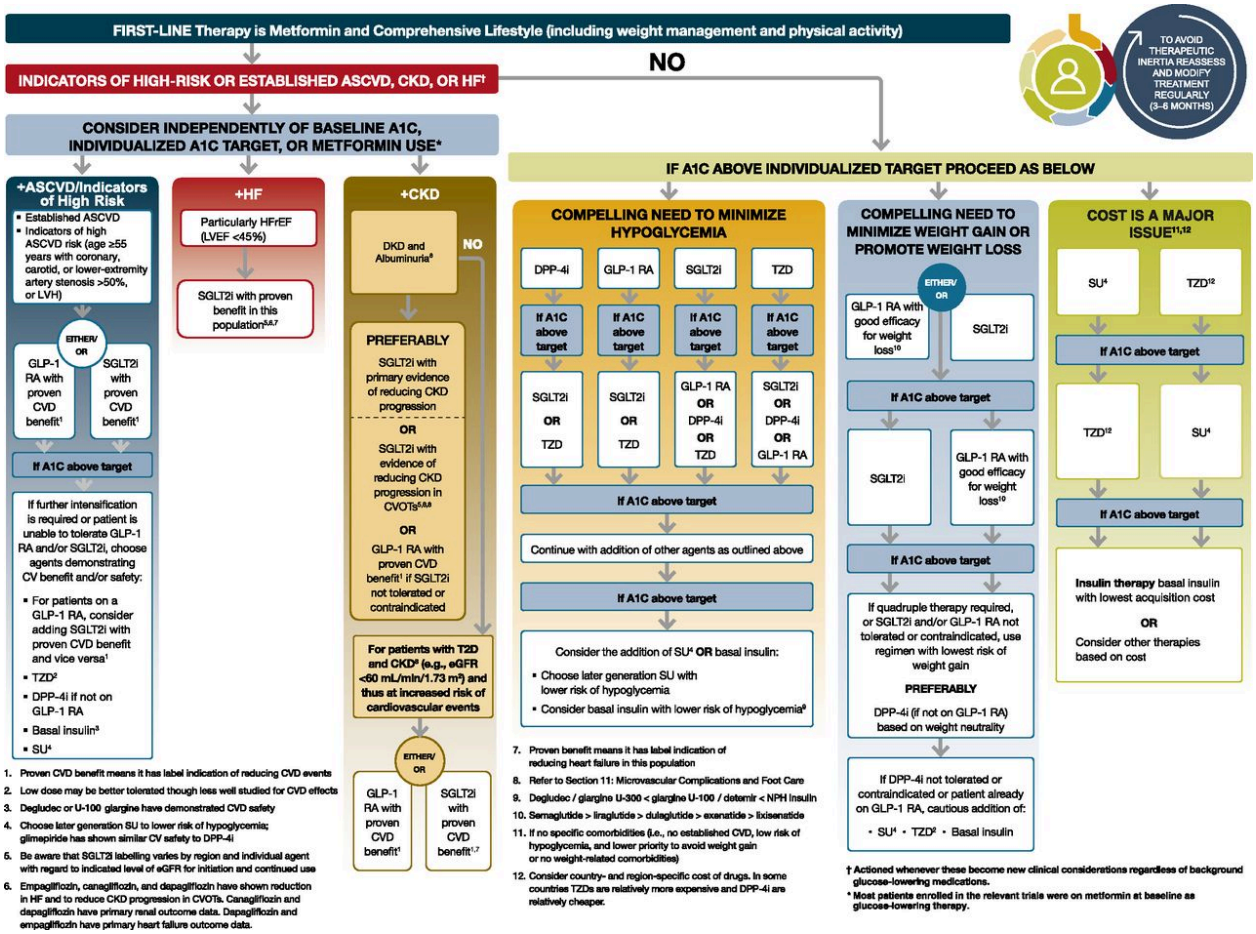
- **Authentication**: OAuth 2.0 + OpenID Connect
- **Authorization**: RBAC with ABAC policies
- **MFA**: TOTP/WebAuthn for clinical users
- **Session Management**: Redis-backed with 15-minute idle timeout

---

Document 3: Medical Knowledge Base and Clinical Guidelines

3.1 Comprehensive Clinical Guidelines Integration

Type 2 Diabetes Management Guidelines



American Diabetes Association (ADA) Standards of Care 2024

- **Glycemic Targets**:
  - General adult target: HbA1c <7.0% (53 mmol/mol)
  - Individualized targets based on:
    - Life expectancy
    - Comorbidities
    - Hypoglycemia risk
    - Disease duration
    - Patient preferences
  - Time in Range (TIR) target: >70% (70-180 mg/dL)
  - Time below range: <4% (<70 mg/dL), <1% (<54 mg/dL)

Medication Algorithm (2024 Update):

```
class ADAMedicationAlgorithm:
    first_line = {
        "medication": "Metformin",
        "contraindications": ["eGFR <30", "acidosis_risk", "hypoxia"],
        "starting_dose": "500mg daily or BID",
```

```
    "titration": "Increase by 500mg weekly to max 2000mg"
  }

  second_line_options = {
    "ASCVD_or_high_risk": ["GLP-1_RA", "SGLT2i"],
    "HF_or_CKD": ["SGLT2i"],
    "weight_management_priority": ["GLP-1_RA", "dual_GIP_GLP1"],
    "hypoglycemia_concern": ["DPP-4i", "GLP-1_RA", "SGLT2i", "TZD"],
    "cost_priority": ["Sulfonylurea", "TZD"]
  }

  injectable_therapies = {
    "GLP-1_before_insulin": True,
    "basal_insulin_start": "10_units_or_0.1-0.2_units/kg",
    "prandial_addition": "4_units_or_10%_basal_dose"
  }
```

EASD/ADA Consensus Report 2023

- Emphasis on cardiorenal protection
- Early combination therapy for HbA1c >1.5% above target
- Continuous glucose monitoring for all on intensive insulin
- Annual screening updates:
    - Retinopathy: Starting at diagnosis
    - Nephropathy: Annual eGFR and UACR
    - Neuropathy: 10-g monofilament + one additional test
    - ASCVD: Lipids annually, more frequent if abnormal

Type 1 Diabetes Management

ADA/EASD Type 1 Consensus 2024

- **Insulin Regimens**:
    - Multiple daily injections (MDI): ≥4 injections/day
    - Continuous subcutaneous insulin infusion (CSII)
    - Automated insulin delivery (AID) systems preferred

Insulin Dosing Algorithms:

```
class Type1InsulinAlgorithms:
  total_daily_dose = {
    "initial": "0.4-0.5 units/kg/day",
    "honeymoon": "0.2-0.4 units/kg/day",
    "established": "0.5-1.0 units/kg/day",
```

```
    "insulin_resistant": ">1.0 units/kg/day"
}

basal_bolus_split = {
    "basal": "40-50% of TDD",
    "bolus": "50-60% of TDD"
}

carb_ratios = {
    "starting": "1:15 (1 unit per 15g carbs)",
    "calculation": "450-500 / TDD",
    "adjustment": "10-20% changes based on patterns"
}

correction_factors = {
    "starting": "1:50 (1 unit drops BG 50 mg/dL)",
    "calculation": "1700-1800 / TDD",
    "timing": "Give with meal if BG >140 mg/dL"
}
```

Special Populations

Pediatric Diabetes (ISPAD 2024)

- **Glycemic Targets**:
    - HbA1c <7.0% without significant hypoglycemia
    - TIR >70%, TBR <4%
    - Sensor use for all children with T1D

Gestational Diabetes (ACOG/ADA 2024)

- **Screening**:
    - Universal screening at 24-28 weeks
    - Early screening if risk factors
- **Targets**:
    - Fasting <95 mg/dL
    - 1-hour postprandial <140 mg/dL
    - 2-hour postprandial <120 mg/dL

Elderly Diabetes Management (AGS/ADA)

- **Healthy elderly**: Standard targets
- **Complex/intermediate health**: HbA1c <8.0%
- **Very complex/poor health**: HbA1c <8.5%

Diabetes Technology Integration

Continuous Glucose Monitoring (CGM)

```
class CGMIntegration:
    devices = {
        "dexcom_g7": {
            "warm_up": "30 minutes",
            "sensor_life": "10 days",
            "accuracy": "9.1% MARD",
            "features": ["predictive_alerts", "smartphone_direct"]
        },
        "freestyle_libre_3": {
            "warm_up": "60 minutes",
            "sensor_life": "14 days",
            "accuracy": "9.2% MARD",
            "features": ["real_time_continuous", "optional_alarms"]
        },
        "medtronic_guardian_4": {
            "warm_up": "2 hours",
            "sensor_life": "7 days",
            "accuracy": "9.1% MARD",
            "features": ["smart_guard_integration", "calibration_free"]
        }
    }

    metrics = {
        "time_in_range": {"target": 70-180, "goal": ">70%"},
        "time_below_range": {
            "level_1": {"range": 54-69, "goal": "<4%"},
            "level_2": {"range": "<54", "goal": "<1%"}
        },
        "time_above_range": {
            "level_1": {"range": 181-250, "goal": "<25%"},
            "level_2": {"range": ">250", "goal": "<5%"}
        },
        "glycemic_variability": {"cv": "<36%"}
    }
```

Medication Database

Comprehensive Drug Information

Metformin

```
class MetforminProfile:
    mechanism = "Decreases hepatic glucose production, increases insulin sensitivity"

    dosing = {
        "immediate_release": {
            "starting": "500mg daily or BID",
            "maximum": "2550mg daily (850mg TID)",
            "with_meals": True
        },
        "extended_release": {
            "starting": "500-1000mg daily",
            "maximum": "2000mg daily",
            "timing": "with_evening_meal"
        }
    }

    contraindications = [
        "eGFR <30 mL/min/1.73m²",
        "Acute metabolic acidosis",
        "Decompensated heart failure",
        "Severe hepatic impairment"
    ]

    precautions = {
        "eGFR_30_45": "Do not initiate; discontinue if falls below 30",
        "contrast_procedures": "Hold 48 hours post-procedure",
        "surgery": "Hold day of surgery",
        "b12_deficiency": "Monitor annually"
    }

    side_effects = {
        "common": ["GI upset (30%)", "Diarrhea", "Nausea", "Metallic taste"],
        "serious": ["Lactic acidosis (rare)", "B12 deficiency"]
    }
```

SGLT2 Inhibitors

```
class SGLT2InhibitorProfiles:
    medications = {
        "empagliflozin": {
            "doses": ["10mg", "25mg"],
            "cv_outcome_trial": "EMPA-REG",
            "cv_death_reduction": "38%",
            "renal_benefits": "EMPA-KIDNEY positive"
```

```
        },
        "dapagliflozin": {
            "doses": ["5mg", "10mg"],
            "cv_outcome_trial": "DECLARE-TIMI",
            "hf_reduction": "27%",
            "renal_trial": "DAPA-CKD positive"
        },
        "canagliflozin": {
            "doses": ["100mg", "300mg"],
            "cv_outcome_trial": "CANVAS",
            "renal_trial": "CREDENCE positive",
            "warnings": ["amputation_risk", "fracture_risk"]
        },
        "ertugliflozin": {
            "doses": ["5mg", "15mg"],
            "cv_outcome_trial": "VERTIS-CV",
            "cv_neutral": True
        }
    }

    class_effects = {
        "benefits": [
            "HbA1c reduction: 0.5-1.0%",
            "Weight loss: 2-3 kg",
            "BP reduction: 3-5 mmHg",
            "Cardiovascular protection",
            "Renal protection"
        ],
        "risks": [
            "Genital mycotic infections: 5-10%",
            "Euglycemic DKA: rare but serious",
            "Volume depletion",
            "Acute kidney injury (initial)",
            "Fournier's gangrene (rare)"
        ]
    }
```

GLP-1 Receptor Agonists

```
class GLP1AgonistProfiles:
    medications = {
        "semaglutide": {
            "formulations": {
                "ozempic": {"doses": ["0.25mg", "0.5mg", "1mg", "2mg"], "frequency": "weekly"},
```

```
            "rybelsus": {"doses": ["3mg", "7mg", "14mg"], "frequency": "daily_oral"},
            "wegovy": {"doses": ["up to 2.4mg"], "indication": "weight_management"}
        },
        "cv_trial": "SUSTAIN-6, PIONEER-6",
        "weight_loss": "5-15%"
    },
    "dulaglutide": {
        "doses": ["0.75mg", "1.5mg", "3mg", "4.5mg"],
        "frequency": "weekly",
        "cv_trial": "REWIND",
        "device": "single_use_pen"
    },
    "liraglutide": {
        "doses": ["0.6mg", "1.2mg", "1.8mg"],
        "frequency": "daily",
        "cv_trial": "LEADER",
        "cv_death_reduction": "22%"
    },
    "tirzepatide": {
        "doses": ["2.5mg", "5mg", "7.5mg", "10mg", "12.5mg", "15mg"],
        "frequency": "weekly",
        "mechanism": "dual GIP/GLP-1",
        "hba1c_reduction": "up to 2.5%",
        "weight_loss": "up to 22.5%"
    }
}
```

Clinical Decision Rules

Insulin Initiation and Titration

```
class InsulinAlgorithms:
    basal_initiation = {
        "starting_dose": "10 units or 0.1-0.2 units/kg",
        "timing": "bedtime or same time daily",
        "titration": {
            "frequency": "every 3 days",
            "fasting_target": "80-130 mg/dL",
            "adjustment_rules": [
                {"if_fbg": ">180", "increase": 4},
                {"if_fbg": "140-180", "increase": 2},
                {"if_fbg": "80-130", "no_change": True},
                {"if_fbg": "<80", "decrease": 2}
```

```
        ]
    }
}

prandial_addition = {
    "when": "basal optimized but A1c above target",
    "starting": "4 units or 10% of basal dose",
    "meal": "largest meal first",
    "titration": "increase by 1-2 units every 3 days until target"
}

basal_bolus_optimization = {
    "basal_testing": "skip meal and check BG q2h",
    "dawn_phenomenon": "3am BG check",
    "somogyi_effect": "2-3am hypoglycemia followed by rebound"
}
```

Cardiovascular Risk Management

ASCVD Risk Calculation and Management

```
class CVRiskManagement:
    risk_calculator = {
        "variables": [
            "age", "sex", "race", "total_cholesterol",
            "hdl", "systolic_bp", "bp_treatment",
            "diabetes", "smoking"
        ],
        "categories": {
            "low": "<5%",
            "borderline": "5-7.4%",
            "intermediate": "7.5-19.9%",
            "high": "≥20%"
        }
    }

    statin_therapy = {
        "diabetes_40_75_years": "moderate_intensity",
        "diabetes_with_ascvd": "high_intensity",
        "diabetes_10yr_risk_>20%": "high_intensity",
        "high_intensity": ["atorvastatin 40-80mg", "rosuvastatin 20-40mg"],
        "moderate_intensity": ["atorvastatin 10-20mg", "rosuvastatin 5-10mg",
                               "simvastatin 20-40mg", "pravastatin 40-80mg"]
```

```
    }

    bp_targets = {
        "general_diabetes": "<140/90",
        "high_cv_risk": "<130/80 if tolerated",
        "albuminuria": "<130/80"
    }

    antiplatelet = {
        "secondary_prevention": "aspirin 75-162mg",
        "primary_prevention": "consider if 10yr risk >10%"
    }
```

Microvascular Complication Screening

Comprehensive Screening Protocols

```
class ComplicationScreening:
    retinopathy = {
        "initial": "at diagnosis for T2D, 5 years after onset for T1D",
        "frequency": "annual if no retinopathy",
        "method": "dilated fundoscopy or retinal photography",
        "referral": "moderate NPDR or worse"
    }

    nephropathy = {
        "tests": ["annual eGFR", "annual UACR"],
        "ckd_staging": {
            "G1": {"egfr": "≥90", "description": "normal"},
            "G2": {"egfr": "60-89", "description": "mild decrease"},
            "G3a": {"egfr": "45-59", "description": "moderate decrease"},
            "G3b": {"egfr": "30-44", "description": "moderate decrease"},
            "G4": {"egfr": "15-29", "description": "severe decrease"},
            "G5": {"egfr": "<15", "description": "kidney failure"}
        },
        "albuminuria_categories": {
            "A1": "<30 mg/g",
            "A2": "30-300 mg/g",
            "A3": ">300 mg/g"
        }
    }

    neuropathy = {
```

```
    "screening": "annually starting at diagnosis for T2D",
    "tests": [
      "10-g monofilament",
      "vibration perception",
      "ankle reflexes",
      "temperature sensation"
    ],
    "autonomic": [
      "orthostatic hypotension",
      "gastroparesis symptoms",
      "erectile dysfunction"
    ]
  }
```

Document 4: Technical Implementation Guide

4.1 Development Environment Setup

Prerequisites and Tools

Development Machine Requirements

```
# Minimum hardware requirements
- CPU: 8 cores (16 threads)
- RAM: 32GB (64GB recommended)
- Storage: 500GB SSD
- GPU: NVIDIA RTX 3060 or better (for local ML development)

# Required software
- Docker Desktop 4.26+
- Kubernetes (minikube or kind for local)
- Python 3.11+
- Node.js 20 LTS
- Go 1.21+
- Rust 1.75+
- PostgreSQL 16
- Redis 7+
```

Local Development Setup

1. Clone and Initialize Repository

```
# Clone repository
git clone https://github.com/easy-diabetes/easy-diabetes.git
cd easy-diabetes

# Initialize git-lfs for large model files
git lfs install
git lfs pull

# Setup pre-commit hooks
pip install pre-commit
pre-commit install

# Copy environment templates
cp .env.example .env.local
cp docker-compose.override.yml.example docker-compose.override.yml
```

2. Container Environment Setup

```
# docker-compose.yml
version: '3.9'

services:
  postgres:
    image: timescale/timescaledb-ha:pg16
    environment:
      POSTGRES_USER: easy_diabetes
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: easy_diabetes
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d
    ports:
      - "5432:5432"

  neo4j:
    image: neo4j:5.15-enterprise
    environment:
      NEO4J_AUTH: neo4j/${NEO4J_PASSWORD}
      NEO4J_ACCEPT_LICENSE_AGREEMENT: yes
      NEO4J_dbms_memory_heap_max__size: 2G
```

```
    volumes:
      - neo4j_data:/data
    ports:
      - "7474:7474"
      - "7687:7687"

  redis:
    image: redis:7-alpine
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data
    ports:
      - "6379:6379"

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9092:9092"

  clinical-api:
    build:
      context: ./services/clinical-api
      dockerfile: Dockerfile.dev
    environment:
      DATABASE_URL:
postgresql://easy_diabetes:${DB_PASSWORD}@postgres:5432/easy_diabetes
      REDIS_URL: redis://redis:6379
      NEO4J_URI: bolt://neo4j:7687
    volumes:
      - ./services/clinical-api:/app
    ports:
      - "8000:8000"
    command: uvicorn main:app --reload --host 0.0.0.0

volumes:
  postgres_data:
  neo4j_data:
```

```
  redis_data:
```

3. Python Environment Setup

```
# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install development dependencies
pip install -r requirements-dev.txt

# Install project in editable mode
pip install -e .

# Setup Jupyter kernel for notebooks
python -m ipykernel install --user --name easy-diabetes

# Install ML dependencies
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
pip install transformers datasets accelerate
```

4. Frontend Development Setup

```
# Navigate to frontend directory
cd frontend

# Install dependencies
npm install

# Setup environment variables
cp .env.example .env.local

# Start development server
npm run dev

# In another terminal, start Storybook for component development
npm run storybook
```

IDE Configuration

VS Code Settings

```
{
  "python.linting.enabled": true,
  "python.linting.pylintEnabled": true,
  "python.linting.flake8Enabled": true,
  "python.formatting.provider": "black",
  "python.formatting.blackArgs": ["--line-length", "100"],
  "python.sortImports.provider": "isort",
  "python.testing.pytestEnabled": true,
  "editor.formatOnSave": true,
  "editor.codeActionsOnSave": {
    "source.organizeImports": true
  },
  "[typescript]": {
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  },
  "[typescriptreact]": {
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  }
}
```

4.2 Core Service Implementation

Clinical Decision Service

FastAPI Application Structure

```
# services/clinical-api/main.py
from fastapi import FastAPI, Depends, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
import uvicorn

from app.core.config import settings
from app.core.database import init_db
from app.api.v1.api import api_router
from app.middleware.auth import AuthMiddleware
from app.middleware.logging import LoggingMiddleware
from app.core.events import create_start_app_handler, create_stop_app_handler

@asynccontextmanager
async def lifespan(app: FastAPI):
    # Startup
    await create_start_app_handler(app)()
```

```
    await init_db()
    yield
    # Shutdown
    await create_stop_app_handler(app)()

app = FastAPI(
    title="EASY-Diabetes Clinical API",
    description="Clinical Decision Support System for Diabetes Management",
    version="2.0.0",
    lifespan=lifespan
)

# Middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.add_middleware(AuthMiddleware)
app.add_middleware(LoggingMiddleware)

# Include routers
app.include_router(api_router, prefix=settings.API_V1_STR)

if __name__ == "__main__":
    uvicorn.run(
        "main:app",
        host="0.0.0.0",
        port=8000,
        reload=settings.DEBUG,
        log_config="logging.yaml"
    )
```

Clinical Decision Engine Implementation

```
# services/clinical-api/app/core/clinical_engine.py
from typing import List, Dict, Any, Optional
import asyncio
from datetime import datetime, timedelta

from app.models.patient import Patient
from app.models.clinical import ClinicalData, Medication, Condition
```

```
from app.services.guidelines import GuidelineService
from app.services.risk_assessment import RiskAssessmentService
from app.services.medication_optimizer import MedicationOptimizer
from app.ai.agents import ClinicalAgentOrchestrator

class ClinicalDecisionEngine:
    def __init__(self):
        self.guideline_service = GuidelineService()
        self.risk_service = RiskAssessmentService()
        self.medication_optimizer = MedicationOptimizer()
        self.agent_orchestrator = ClinicalAgentOrchestrator()

    async def generate_recommendations(
        self,
        patient: Patient,
        clinical_data: ClinicalData,
        request_context: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Generate comprehensive clinical recommendations"""

        # Parallel execution of assessment tasks
        assessment_tasks = [
            self._assess_current_state(patient, clinical_data),
            self._calculate_risk_scores(patient, clinical_data),
            self._identify_treatment_gaps(patient, clinical_data),
            self._check_medication_issues(patient, clinical_data)
        ]

        results = await asyncio.gather(*assessment_tasks)
        current_state, risk_scores, treatment_gaps, medication_issues = results

        # Generate recommendations using AI agents
        agent_recommendations = await self.agent_orchestrator.generate_recommendations(
            patient=patient,
            clinical_data=clinical_data,
            assessment_results={
                "current_state": current_state,
                "risk_scores": risk_scores,
                "treatment_gaps": treatment_gaps,
                "medication_issues": medication_issues
            }
        )

        # Validate recommendations against guidelines
```

```
        validated_recommendations = await self._validate_recommendations(
            agent_recommendations,
            patient,
            clinical_data
        )

        # Generate explanation and evidence
        explanation = await self._generate_explanation(
            validated_recommendations,
            patient,
            clinical_data
        )

        return {
            "recommendations": validated_recommendations,
            "risk_assessment": risk_scores,
            "explanation": explanation,
            "evidence": self._compile_evidence(validated_recommendations),
            "confidence_scores": self._calculate_confidence(validated_recommendations),
            "generated_at": datetime.utcnow().isoformat()
        }

    async def _assess_current_state(
        self,
        patient: Patient,
        clinical_data: ClinicalData
    ) -> Dict[str, Any]:
        """Assess patient's current clinical state"""

        latest_labs = clinical_data.get_recent_labs(days=90)
        current_medications = clinical_data.get_active_medications()
        recent_events = clinical_data.get_recent_events(days=30)

        return {
            "glycemic_control": self._assess_glycemic_control(latest_labs),
            "medication_adherence": await self._calculate_adherence(patient, current_medications),
            "complication_status": self._assess_complications(clinical_data),
            "lifestyle_factors": await self._assess_lifestyle(patient),
            "recent_events": self._summarize_events(recent_events)
        }
```

AI Agent Implementation

Multi-Agent Orchestrator

```
# services/clinical-api/app/ai/agents/orchestrator.py
from typing import Dict, Any, List
import asyncio
from langchain.agents import AgentExecutor
from langchain.chains import LLMChain
from langchain.memory import ConversationSummaryBufferMemory

from app.ai.agents.clinical_assessment import ClinicalAssessmentAgent
from app.ai.agents.guideline_compliance import GuidelineComplianceAgent
from app.ai.agents.medication_optimization import MedicationOptimizationAgent
from app.ai.agents.safety_validation import SafetyValidationAgent
from app.ai.agents.explanation_generation import ExplanationGenerationAgent

class ClinicalAgentOrchestrator:
    def __init__(self):
        self.agents = {
            "assessment": ClinicalAssessmentAgent(),
            "guidelines": GuidelineComplianceAgent(),
            "medication": MedicationOptimizationAgent(),
            "safety": SafetyValidationAgent(),
            "explanation": ExplanationGenerationAgent()
        }

        self.memory = ConversationSummaryBufferMemory(
            memory_key="chat_history",
            return_messages=True,
            max_token_limit=2000
        )

    async def generate_recommendations(
        self,
        patient: Dict[str, Any],
        clinical_data: Dict[str, Any],
        assessment_results: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Orchestrate multiple agents to generate recommendations"""

        # Phase 1: Clinical Assessment
        assessment_output = await self.agents["assessment"].analyze(
            patient=patient,
            clinical_data=clinical_data,
            context=assessment_results
```

```
)

# Phase 2: Guideline Application (parallel execution)
guideline_tasks = []
for condition in patient.get("conditions", []):
    task = self.agents["guidelines"].apply_guidelines(
        condition=condition,
        patient_context=patient,
        assessment=assessment_output
    )
    guideline_tasks.append(task)

guideline_results = await asyncio.gather(*guideline_tasks)

# Phase 3: Medication Optimization
medication_recommendations = await self.agents["medication"].optimize(
    current_medications=clinical_data.get("medications", []),
    patient_context=patient,
    guidelines=guideline_results,
    assessment=assessment_output
)

# Phase 4: Safety Validation
safety_validated = await self.agents["safety"].validate(
    recommendations=medication_recommendations,
    patient=patient,
    clinical_data=clinical_data
)

# Phase 5: Generate Explanations
explanations = await self.agents["explanation"].generate(
    recommendations=safety_validated,
    reasoning_chain=self._build_reasoning_chain(
        assessment_output,
        guideline_results,
        medication_recommendations,
        safety_validated
    )
)

return {
    "recommendations": safety_validated["safe_recommendations"],
    "explanations": explanations,
    "confidence": self._calculate_overall_confidence(safety_validated),
```

```
        "agent_outputs": {
            "assessment": assessment_output,
            "guidelines": guideline_results,
            "medication": medication_recommendations,
            "safety": safety_validated
        }
    }
```

Clinical Assessment Agent

```
# services/clinical-api/app/ai/agents/clinical_assessment.py
from langchain.agents import Tool, AgentExecutor
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain.llms import ChatOpenAI

class ClinicalAssessmentAgent:
    def __init__(self):
        self.llm = ChatOpenAI(
            model="gpt-4-turbo",
            temperature=0.1,
            max_tokens=2000
        )

        self.tools = [
            Tool(
                name="LabAnalyzer",
                func=self._analyze_labs,
                description="Analyze laboratory results and identify abnormalities"
            ),
            Tool(
                name="CGMAnalyzer",
                func=self._analyze_cgm,
                description="Analyze continuous glucose monitoring data patterns"
            ),
            Tool(
                name="MedicationReviewer",
                func=self._review_medications,
                description="Review current medications for efficacy and issues"
            ),
            Tool(
                name="ComplicationScreener",
                func=self._screen_complications,
                description="Screen for diabetes complications"
```

```
            )
        ]

        self.assessment_prompt = PromptTemplate(
            input_variables=["patient_data", "clinical_data", "tool_outputs"],
            template="""
            You are an expert endocrinologist assessing a patient with diabetes.

            Patient Information:
            {patient_data}

            Clinical Data:
            {clinical_data}

            Analysis Results:
            {tool_outputs}

            Provide a comprehensive clinical assessment including:
            1. Current glycemic control status
            2. Risk factors and complications
            3. Treatment effectiveness
            4. Key clinical concerns
            5. Recommended focus areas for optimization

            Format your response as a structured clinical assessment.
            """
        )

    async def analyze(
        self,
        patient: Dict[str, Any],
        clinical_data: Dict[str, Any],
        context: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Perform comprehensive clinical assessment"""

        # Execute tools to gather information
        tool_results = {}
        for tool in self.tools:
            result = await tool.func(clinical_data)
            tool_results[tool.name] = result

        # Generate assessment using LLM
        assessment_chain = LLMChain(
```

```
            llm=self.llm,
            prompt=self.assessment_prompt
        )

        assessment = await assessment_chain.arun(
            patient_data=patient,
            clinical_data=clinical_data,
            tool_outputs=tool_results
        )

        # Structure the output
        return {
            "clinical_summary": assessment,
            "tool_results": tool_results,
            "key_findings": self._extract_key_findings(assessment),
            "risk_factors": self._identify_risk_factors(tool_results),
            "optimization_opportunities": self._identify_opportunities(assessment)
        }
```

Database Schema Implementation

PostgreSQL Schema

```
-- Core patient schema
CREATE SCHEMA IF NOT EXISTS patient;
CREATE SCHEMA IF NOT EXISTS clinical;
CREATE SCHEMA IF NOT EXISTS audit;

-- Enable necessary extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE EXTENSION IF NOT EXISTS "btree_gin";

-- Patient table with encryption for sensitive data
CREATE TABLE patient.patients (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    external_id VARCHAR(100) UNIQUE NOT NULL,
    -- Encrypted personal information
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    date_of_birth DATE NOT NULL,
    gender VARCHAR(10) CHECK (gender IN ('male', 'female', 'other')),
    -- Contact information (encrypted)
```

```
    email TEXT,
    phone TEXT,
    -- Clinical identifiers
    mrn VARCHAR(50),
    -- Metadata
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    created_by UUID REFERENCES auth.users(id),
    is_active BOOLEAN DEFAULT true,
    -- Indexes
    CONSTRAINT patients_external_id_key UNIQUE (external_id)
);

-- Enable row-level security
ALTER TABLE patient.patients ENABLE ROW LEVEL SECURITY;

-- Clinical measurements with partitioning
CREATE TABLE clinical.measurements (
    id UUID DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patient.patients(id),
    measurement_type VARCHAR(50) NOT NULL,
    value NUMERIC(10,3) NOT NULL,
    unit VARCHAR(20) NOT NULL,
    measured_at TIMESTAMPTZ NOT NULL,
    source VARCHAR(50), -- 'manual', 'device', 'lab', 'ehr'
    device_id UUID,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id, measured_at)
) PARTITION BY RANGE (measured_at);

-- Create monthly partitions
DO $$
DECLARE
    start_date DATE := '2024-01-01';
    end_date DATE;
    partition_name TEXT;
BEGIN
    FOR i IN 0..23 LOOP
        end_date := start_date + INTERVAL '1 month';
        partition_name := 'measurements_' || TO_CHAR(start_date, 'YYYY_MM');

        EXECUTE format(
            'CREATE TABLE clinical.%I PARTITION OF clinical.measurements
```

```
            FOR VALUES FROM (%L) TO (%L)',
            partition_name, start_date, end_date
        );

        start_date := end_date;
    END LOOP;
END $$;

-- Medications table
CREATE TABLE clinical.medications (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patient.patients(id),
    medication_name VARCHAR(200) NOT NULL,
    generic_name VARCHAR(200),
    drug_class VARCHAR(100),
    dose NUMERIC(10,3),
    dose_unit VARCHAR(20),
    frequency VARCHAR(50),
    route VARCHAR(20),
    start_date DATE NOT NULL,
    end_date DATE,
    discontinued_reason TEXT,
    prescriber_id UUID,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Create indexes
CREATE INDEX idx_medications_patient_active ON clinical.medications(patient_id, is_active);
CREATE INDEX idx_medications_drug_class ON clinical.medications(drug_class);

-- Conditions table
CREATE TABLE clinical.conditions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patient.patients(id),
    condition_code VARCHAR(20),
    condition_system VARCHAR(20), -- 'ICD10', 'SNOMED'
    condition_display TEXT NOT NULL,
    clinical_status VARCHAR(20), -- 'active', 'resolved', 'inactive'
    verification_status VARCHAR(20), -- 'confirmed', 'provisional', 'differential'
    onset_date DATE,
    abatement_date DATE,
    recorded_date DATE NOT NULL,
```

```
    recorder_id UUID,
    notes TEXT,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Laboratory results
CREATE TABLE clinical.lab_results (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patient.patients(id),
    lab_type VARCHAR(50) NOT NULL,
    loinc_code VARCHAR(20),
    value VARCHAR(50) NOT NULL,
    value_numeric NUMERIC(10,3),
    unit VARCHAR(20),
    reference_range VARCHAR(50),
    interpretation VARCHAR(20), -- 'normal', 'high', 'low', 'critical'
    collected_at TIMESTAMPTZ NOT NULL,
    resulted_at TIMESTAMPTZ,
    performing_lab VARCHAR(100),
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Create indexes for common queries
CREATE INDEX idx_lab_results_patient_type_date ON clinical.lab_results(patient_id, lab_type,
collected_at DESC);
CREATE INDEX idx_lab_results_interpretation ON clinical.lab_results(interpretation) WHERE
interpretation != 'normal';

-- Audit log table
CREATE TABLE audit.audit_log (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL,
    patient_id UUID,
    action VARCHAR(50) NOT NULL,
    resource_type VARCHAR(50) NOT NULL,
    resource_id UUID,
    changes JSONB,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Create indexes for audit queries
CREATE INDEX idx_audit_log_user_date ON audit.audit_log(user_id, created_at DESC);
```

```
CREATE INDEX idx_audit_log_patient_date ON audit.audit_log(patient_id, created_at DESC);
```

Neo4j Knowledge Graph Schema

```
// Create constraints for unique identifiers
CREATE CONSTRAINT patient_id IF NOT EXISTS ON (p:Patient) ASSERT p.id IS UNIQUE;
CREATE CONSTRAINT medication_id IF NOT EXISTS ON (m:Medication) ASSERT m.id IS
UNIQUE;
CREATE CONSTRAINT condition_id IF NOT EXISTS ON (c:Condition) ASSERT c.id IS
UNIQUE;
CREATE CONSTRAINT guideline_id IF NOT EXISTS ON (g:Guideline) ASSERT g.id IS
UNIQUE;

// Create indexes for performance
CREATE INDEX patient_external_id IF NOT EXISTS FOR (p:Patient) ON (p.external_id);
CREATE INDEX medication_name IF NOT EXISTS FOR (m:Medication) ON (m.name);
CREATE INDEX condition_code IF NOT EXISTS FOR (c:Condition) ON (c.code);

// Example knowledge graph structure
// Create patient node
CREATE (p:Patient {
    id: 'uuid-here',
    external_id: 'EXT123',
    age_group: '45-54',
    diabetes_type: 'type2',
    duration_years: 5
})

// Create condition nodes
CREATE (dm:Condition {
    id: 'uuid-dm',
    code: 'E11.9',
    system: 'ICD10',
    name: 'Type 2 diabetes mellitus without complications'
})

CREATE (htn:Condition {
    id: 'uuid-htn',
    code: 'I10',
    system: 'ICD10',
    name: 'Essential hypertension'
})

// Create relationships
```

```
CREATE (p)-[:HAS_CONDITION {diagnosed_date: date('2019-01-15')}]->(dm)
CREATE (p)-[:HAS_CONDITION {diagnosed_date: date('2020-03-20')}]->(htn)

// Create medication nodes and relationships
CREATE (met:Medication {
    id: 'uuid-met',
    name: 'Metformin',
    drug_class: 'Biguanide',
    mechanism: 'Decreases hepatic glucose production'
})

CREATE (p)-[:TAKES_MEDICATION {
    dose: 1000,
    unit: 'mg',
    frequency: 'BID',
    start_date: date('2019-02-01'),
    adherence_rate: 0.85
}]->(met)

// Create similar patient relationships for collaborative filtering
MATCH (p1:Patient), (p2:Patient)
WHERE p1.id <> p2.id
  AND abs(p1.age_group - p2.age_group) <= 1
  AND p1.diabetes_type = p2.diabetes_type
  AND abs(p1.duration_years - p2.duration_years) <= 2
CREATE (p1)-[:SIMILAR_TO {
    similarity_score: 0.85,
    calculated_at: datetime()
}]->(p2)

// Guideline relationships
CREATE (g:Guideline {
    id: 'uuid-guide',
    name: 'ADA Standards of Care 2024',
    version: '2024.1',
    section: 'Pharmacologic Approaches'
})

CREATE (met)-[:RECOMMENDED_BY {
    recommendation_strength: 'strong',
    evidence_level: 'A'
}]->(g)

// Contraindication relationships
```

```
CREATE (ckd:Condition {
    id: 'uuid-ckd',
    code: 'N18.5',
    name: 'Chronic kidney disease, stage 5'
})

CREATE (met)-[:CONTRAINDICATED_IN {
    absolute: true,
    reason: 'eGFR < 30'
}]->(ckd)
```

API Endpoint Implementation

Patient Management Endpoints

```
# services/clinical-api/app/api/v1/endpoints/patients.py
from typing import List, Optional
from fastapi import APIRouter, Depends, HTTPException, Query
from sqlalchemy.ext.asyncio import AsyncSession

from app.api.deps import get_db, get_current_user
from app.models.user import User
from app.schemas.patient import (
    PatientCreate,
    PatientUpdate,
    PatientResponse,
    PatientListResponse
)
from app.crud.patient import patient_crud
from app.core.security import check_permission

router = APIRouter()

@router.post("/", response_model=PatientResponse)
async def create_patient(
    *,
    db: AsyncSession = Depends(get_db),
    patient_in: PatientCreate,
    current_user: User = Depends(get_current_user)
) -> PatientResponse:
    """Create new patient record"""

    # Check permissions
```

```
    if not check_permission(current_user, "patient:create"):
        raise HTTPException(status_code=403, detail="Insufficient permissions")

    # Check for duplicate
    existing = await patient_crud.get_by_external_id(
        db, external_id=patient_in.external_id
    )
    if existing:
        raise HTTPException(
            status_code=400,
            detail="Patient with this external ID already exists"
        )

    # Create patient
    patient = await patient_crud.create_with_owner(
        db, obj_in=patient_in, owner_id=current_user.id
    )

    # Log audit event
    await audit_log.create_entry(
        db,
        user_id=current_user.id,
        action="create_patient",
        resource_type="patient",
        resource_id=patient.id
    )

    return patient

@router.get("/{patient_id}", response_model=PatientResponse)
async def get_patient(
    *,
    db: AsyncSession = Depends(get_db),
    patient_id: str,
    current_user: User = Depends(get_current_user)
) -> PatientResponse:
    """Get patient by ID"""

    # Get patient
    patient = await patient_crud.get(db, id=patient_id)
    if not patient:
        raise HTTPException(status_code=404, detail="Patient not found")

    # Check access permissions
```

```
    if not check_patient_access(current_user, patient):
        raise HTTPException(status_code=403, detail="Access denied")

    return patient

@router.get("/", response_model=PatientListResponse)
async def list_patients(
    db: AsyncSession = Depends(get_db),
    skip: int = Query(0, ge=0),
    limit: int = Query(100, ge=1, le=1000),
    search: Optional[str] = None,
    current_user: User = Depends(get_current_user)
) -> PatientListResponse:
    """List patients with pagination and search"""

    # Check permissions
    if not check_permission(current_user, "patient:list"):
        raise HTTPException(status_code=403, detail="Insufficient permissions")

    # Get patients based on user role
    if current_user.role == "clinician":
        # Only show assigned patients
        patients = await patient_crud.get_by_clinician(
            db,
            clinician_id=current_user.id,
            skip=skip,
            limit=limit,
            search=search
        )
    else:
        # Admin can see all
        patients = await patient_crud.get_multi(
            db, skip=skip, limit=limit, search=search
        )

    return PatientListResponse(
        items=patients,
        total=len(patients),
        skip=skip,
        limit=limit
    )
```

Clinical Recommendations Endpoint

```
# services/clinical-api/app/api/v1/endpoints/recommendations.py
from fastapi import APIRouter, Depends, HTTPException, BackgroundTasks
from sqlalchemy.ext.asyncio import AsyncSession
import asyncio

from app.api.deps import get_db, get_current_user
from app.models.user import User
from app.schemas.recommendations import (
    RecommendationRequest,
    RecommendationResponse,
    RecommendationFeedback
)
from app.services.clinical_engine import ClinicalDecisionEngine
from app.services.cache import recommendation_cache
from app.core.monitoring import metrics

router = APIRouter()

@router.post("/{patient_id}/generate", response_model=RecommendationResponse)
async def generate_recommendations(
    *,
    db: AsyncSession = Depends(get_db),
    patient_id: str,
    request: RecommendationRequest,
    background_tasks: BackgroundTasks,
    current_user: User = Depends(get_current_user)
) -> RecommendationResponse:
    """Generate clinical recommendations for patient"""

    # Start timing for metrics
    start_time = asyncio.get_event_loop().time()

    try:
        # Check patient access
        patient = await get_patient_with_access_check(
            db, patient_id, current_user
        )

        # Check cache first
        cache_key = f"recommendations:{patient_id}:{request.hash()}"
        cached = await recommendation_cache.get(cache_key)
        if cached and not request.force_refresh:
            metrics.cache_hits.inc()
            return cached
```

```
        # Get clinical data
        clinical_data = await get_clinical_data(db, patient_id)

        # Initialize decision engine
        engine = ClinicalDecisionEngine()

        # Generate recommendations
        recommendations = await engine.generate_recommendations(
            patient=patient,
            clinical_data=clinical_data,
            request_context={
                "user_id": current_user.id,
                "request_params": request.dict(),
                "timestamp": datetime.utcnow()
            }
        )

        # Cache results
        await recommendation_cache.set(
            cache_key,
            recommendations,
            expire=300  # 5 minutes
        )

        # Background tasks
        background_tasks.add_task(
            log_recommendation_generation,
            patient_id=patient_id,
            user_id=current_user.id,
            recommendations=recommendations
        )

        # Record metrics
        duration = asyncio.get_event_loop().time() - start_time
        metrics.recommendation_generation_time.observe(duration)
        metrics.recommendations_generated.inc()

        return RecommendationResponse(**recommendations)

    except Exception as e:
        metrics.recommendation_errors.inc()
        raise HTTPException(
            status_code=500,
```

```
            detail=f"Error generating recommendations: {str(e)}"
        )

@router.post("/{patient_id}/recommendations/{recommendation_id}/feedback")
async def submit_recommendation_feedback(
    *,
    db: AsyncSession = Depends(get_db),
    patient_id: str,
    recommendation_id: str,
    feedback: RecommendationFeedback,
    current_user: User = Depends(get_current_user)
) -> dict:
    """Submit feedback on a recommendation"""

    # Verify access
    patient = await get_patient_with_access_check(
        db, patient_id, current_user
    )

    # Store feedback
    await store_recommendation_feedback(
        db,
        patient_id=patient_id,
        recommendation_id=recommendation_id,
        feedback=feedback,
        user_id=current_user.id
    )

    # Update ML models if accepted/rejected
    if feedback.action in ["accepted", "rejected"]:
        await queue_model_update(
            recommendation_id=recommendation_id,
            action=feedback.action,
            reason=feedback.reason
        )

    return {"status": "feedback_recorded"}
```

Frontend Implementation

React Component Architecture

```
// frontend/src/components/clinical/ClinicalDashboard.tsx
```

```
import React, { useState, useEffect } from 'react'
import { useQuery, useMutation } from '@tanstack/react-query'
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle
} from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { Alert, AlertDescription } from '@/components/ui/alert'
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs'

import { PatientOverview } from './PatientOverview'
import { GlycemicControl } from './GlycemicControl'
import { MedicationManagement } from './MedicationManagement'
import { RiskAssessment } from './RiskAssessment'
import { ClinicalRecommendations } from './ClinicalRecommendations'
import { usePatientStore } from '@/stores/patientStore'
import { api } from '@/lib/api'

interface ClinicalDashboardProps {
  patientId: string
}

export const ClinicalDashboard: React.FC<ClinicalDashboardProps> = ({
  patientId
}) => {
  const [activeTab, setActiveTab] = useState('overview')
  const { setCurrentPatient } = usePatientStore()

  // Fetch patient data
  const { data: patient, isLoading: patientLoading } = useQuery({
    queryKey: ['patient', patientId],
    queryFn: () => api.patients.get(patientId),
    onSuccess: (data) => setCurrentPatient(data)
  })

  // Fetch clinical data
  const { data: clinicalData, isLoading: clinicalLoading } = useQuery({
    queryKey: ['clinical-data', patientId],
    queryFn: () => api.clinical.getData(patientId),
    refetchInterval: 5 * 60 * 1000 // Refetch every 5 minutes
  })
```

```
// Generate recommendations mutation
const generateRecommendations = useMutation({
  mutationFn: () => api.recommendations.generate(patientId),
  onSuccess: () => {
    queryClient.invalidateQueries(['recommendations', patientId])
  }
})

if (patientLoading || clinicalLoading) {
  return <DashboardSkeleton />
}

return (
  <div className="space-y-6">
    {/* Patient Header */}
    <Card>
      <CardHeader>
        <div className="flex items-center justify-between">
          <div>
            <CardTitle className="text-2xl">
              {patient.firstName} {patient.lastName}
            </CardTitle>
            <CardDescription>
              MRN: {patient.mrn} | {calculateAge(patient.dateOfBirth)} years old
            </CardDescription>
          </div>
          <Button
            onClick={() => generateRecommendations.mutate()}
            disabled={generateRecommendations.isLoading}
          >
            {generateRecommendations.isLoading
              ? 'Generating...'
              : 'Generate Recommendations'}
          </Button>
        </div>
      </CardHeader>
    </Card>

    {/* Main Dashboard Tabs */}
    <Tabs value={activeTab} onValueChange={setActiveTab}>
      <TabsList className="grid w-full grid-cols-5">
        <TabsTrigger value="overview">Overview</TabsTrigger>
        <TabsTrigger value="glycemic">Glycemic Control</TabsTrigger>
```

```
          <TabsTrigger value="medications">Medications</TabsTrigger>
          <TabsTrigger value="risk">Risk Assessment</TabsTrigger>
          <TabsTrigger value="recommendations">Recommendations</TabsTrigger>
        </TabsList>

        <TabsContent value="overview" className="space-y-4">
          <PatientOverview
            patient={patient}
            clinicalData={clinicalData}
          />
        </TabsContent>

        <TabsContent value="glycemic" className="space-y-4">
          <GlycemicControl
            patientId={patientId}
            clinicalData={clinicalData}
          />
        </TabsContent>

        <TabsContent value="medications" className="space-y-4">
          <MedicationManagement
            patientId={patientId}
            medications={clinicalData.medications}
          />
        </TabsContent>

        <TabsContent value="risk" className="space-y-4">
          <RiskAssessment
            patientId={patientId}
            riskScores={clinicalData.riskScores}
          />
        </TabsContent>

        <TabsContent value="recommendations" className="space-y-4">
          <ClinicalRecommendations
            patientId={patientId}
          />
        </TabsContent>
      </Tabs>
    </div>
  )
}
```

Glycemic Control Visualization Component

```
// frontend/src/components/clinical/GlycemicControl.tsx
import React from 'react'
import {
  LineChart,
  Line,
  AreaChart,
  Area,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  ResponsiveContainer,
  ReferenceLine
} from 'recharts'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Badge } from '@/components/ui/badge'

interface GlycemicControlProps {
  patientId: string
  clinicalData: ClinicalData
}

export const GlycemicControl: React.FC<GlycemicControlProps> = ({
  patientId,
  clinicalData
}) => {
  const latestHbA1c = clinicalData.labs.hba1c[0]
  const cgmData = processCGMData(clinicalData.cgm)
  const timeInRange = calculateTimeInRange(cgmData)

  return (
    <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-3">
      {/* HbA1c Card */}
      <Card>
        <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
          <CardTitle className="text-sm font-medium">HbA1c</CardTitle>
          <Badge variant={getHbA1cVariant(latestHbA1c.value)}>
            {getHbA1cStatus(latestHbA1c.value)}
          </Badge>
        </CardHeader>
        <CardContent>
          <div className="text-2xl font-bold">{latestHbA1c.value}%</div>
```

```
    <p className="text-xs text-muted-foreground">
     {formatDate(latestHbA1c.date)}
    </p>
    <div className="mt-4">
     <ResponsiveContainer width="100%" height={100}>
      <LineChart data={clinicalData.labs.hba1c}>
       <Line
        type="monotone"
        dataKey="value"
        stroke="#8884d8"
        strokeWidth={2}
        dot={{ r: 4 }}
       />
       <ReferenceLine y={7} stroke="#ef4444" strokeDasharray="3 3" />
       <Tooltip />
      </LineChart>
     </ResponsiveContainer>
    </div>
  </CardContent>
</Card>

{/* Time in Range Card */}
<Card>
  <CardHeader>
   <CardTitle className="text-sm font-medium">Time in Range</CardTitle>
  </CardHeader>
  <CardContent>
   <div className="space-y-2">
    <TimeRangeBar
     label="Above Range (>180)"
     value={timeInRange.above}
     color="text-orange-500"
     target="<25%"
    />
    <TimeRangeBar
     label="In Range (70-180)"
     value={timeInRange.inRange}
     color="text-green-500"
     target=">70%"
    />
    <TimeRangeBar
     label="Below Range (<70)"
     value={timeInRange.below}
     color="text-red-500"
```

```
        target="<4%"
      />
    </div>
  </CardContent>
</Card>

{/* CGM Trace */}
<Card className="col-span-full">
  <CardHeader>
    <CardTitle>Continuous Glucose Monitor - 24 Hour View</CardTitle>
  </CardHeader>
  <CardContent>
    <ResponsiveContainer width="100%" height={300}>
      <AreaChart data={cgmData}>
        <CartesianGrid strokeDasharray="3 3" />
        <XAxis
          dataKey="time"
          tickFormatter={(time) => formatTime(time)}
        />
        <YAxis domain={[40, 400]} />
        <Tooltip
          labelFormatter={(time) => formatDateTime(time)}
          formatter={(value) => [`${value} mg/dL`, 'Glucose']}
        />

        {/* Target range background */}
        <ReferenceLine y={180} stroke="#ef4444" strokeDasharray="3 3" />
        <ReferenceLine y={70} stroke="#ef4444" strokeDasharray="3 3" />

        {/* Glucose trace */}
        <Area
          type="monotone"
          dataKey="glucose"
          stroke="#8884d8"
          fill="#8884d8"
          fillOpacity={0.3}
        />

        {/* Meal markers */}
        {cgmData.meals?.map((meal, idx) => (
          <ReferenceLine
            key={idx}
            x={meal.time}
            stroke="#10b981"
```

```
              strokeWidth={2}
              label="Meal"
            />
          ))}
        </AreaChart>
      </ResponsiveContainer>
    </CardContent>
  </Card>
 </div>
 )
}
```

Document 5: Product Roadmap

Phase 1: Foundation

Goal: Establish core infrastructure and basic clinical functionality

Technical Milestones

- ✅ Core microservices architecture
- ✅ PostgreSQL and Neo4j setup
- ✅ Basic authentication/authorization
- ✅ FHIR R4 integration framework
- ✅ Initial API development

Clinical Features

- ✅ Patient registration and management
- ✅ Basic medication tracking
- ✅ Lab result integration
- ✅ Simple clinical alerts
- ✅ Basic reporting

Regulatory

- ✅ HIPAA compliance framework
- ✅ Initial FDA pre-submission meeting
- ✅ Quality management system setup

Phase 2: AI Integration

Goal: Implement core AI capabilities and clinical decision support

AI/ML Features

- ⬜ Multi-agent architecture implementation
- ⬜ Clinical assessment agent
- ⬜ Guideline compliance agent
- ⬜ Basic recommendation generation
- ⬜ Explanation framework

Clinical Enhancements

- ⬜ CGM integration (Dexcom, FreeStyle Libre)
- ⬜ Insulin dosing calculator
- ⬜ Hypoglycemia prediction (2-hour)
- ⬜ Medication optimization suggestions

Integration

- ⬜ Epic FHIR integration
- ⬜ Cerner PowerChart integration
- ⬜ Apple HealthKit integration

Phase 3: Advanced Analytics

Goal: Sophisticated predictive models and personalization

Predictive Analytics

- ⬜ 4-hour glucose prediction
- ⬜ HbA1c trajectory modeling
- ⬜ Complication risk scoring
- ⬜ Treatment response prediction

Personalization

- ⬜ Patient similarity matching
- ⬜ Personalized targets
- ⬜ Lifestyle factor integration
- ⬜ Behavioral nudges

Clinical Features

- ⬜ Automated insulin adjustment
- ⬜ Carb counting assistance
- ⬜ Exercise impact prediction
- ⬜ Sick day management

Phase 4: Scale & Optimize

Goal: Production readiness and optimization

Scalability

- ⬜ Kubernetes autoscaling
- ⬜ Multi-region deployment
- ⬜ Performance optimization
- ⬜ Load testing (1M users)

Advanced Features

- ⬜ Federated learning implementation
- ⬜ Real-time collaboration tools
- ⬜ Voice interface
- ⬜ AR medication administration

Regulatory

- ⬜ FDA 510(k) submission
- ⬜ CE marking process
- ⬜ Clinical trial initiation

Phase 5: Expansion

Goal: Broader disease coverage and global deployment

Disease Expansion

- ⬜ Type 1 diabetes full support
- ⬜ Gestational diabetes module
- ⬜ Prediabetes prevention
- ⬜ Pediatric diabetes

Geographic Expansion

- ⬜ EU deployment
- ⬜ Localization (10 languages)
- ⬜ Regional guideline integration
- ⬜ Local EHR integrations

Advanced AI

- ⬜ Multimodal learning
- ⬜ Causal inference engine
- ⬜ Automated clinical trials
- ⬜ Digital twin modeling

Long-term Vision

Platform Evolution

- Comprehensive metabolic disease platform
- Integration with digital therapeutics
- Closed-loop automated care
- Population health management

Technology Advancement

- Quantum computing for drug discovery
- Advanced robotics integration
- Brain-computer interfaces
- Nano-sensor integration

Global Impact

- 10M+ patients managed
- 50+ country deployment
- 100+ health system integrations
- Measurable population health improvement

Document 6: Project Knowledge Base

1. Domain Knowledge

Diabetes Pathophysiology

Type 2 Diabetes:
- Insulin resistance + relative insulin deficiency
- Progressive beta cell dysfunction
- Hepatic glucose overproduction
- Incretin deficiency/resistance
- Accelerated lipolysis
- Glucagon hypersecretion
- Renal glucose reabsorption increase
- Brain insulin resistance

Type 1 Diabetes:
- Autoimmune beta cell destruction
- Absolute insulin deficiency
- Presence of autoantibodies (GAD, IA-2, ZnT8)
- Risk of diabetic ketoacidosis
- Usually juvenile onset (but LADA exists)

Key Clinical Concepts

Glycemic Targets:
- HbA1c: Average glucose over 3 months
- Time in Range: % time glucose 70-180 mg/dL
- Glycemic Variability: Coefficient of variation <36%
- Hypoglycemia: <70 mg/dL (Level 1), <54 mg/dL (Level 2)

Complications:
Microvascular:
- Retinopathy (leading cause of blindness)
- Nephropathy (leading cause of ESRD)
- Neuropathy (leading cause of amputation)

Macrovascular:
- 2-4x increased CV risk
- Stroke risk doubled
- Peripheral arterial disease

Acute:
- Hypoglycemia
- Hyperglycemic crises (DKA, HHS)

Treatment Principles

Lifestyle:
- Medical nutrition therapy
- 150 min/week moderate exercise
- Weight loss 5-7% if overweight
- Smoking cessation

Medications:
- Metformin first-line unless contraindicated
- Early combination therapy if HbA1c >1.5% above target
- Consider CV/renal benefits (SGLT2i, GLP-1)
- Avoid hypoglycemia in elderly
- Cost considerations

Monitoring:
- SMBG if on insulin or sulfonylureas
- CGM for all Type 1, intensive insulin
- HbA1c every 3 months if not at goal
- Annual complication screening

2. Technical Knowledge

System Architecture Patterns

Microservices Benefits:
- Independent deployment
- Technology diversity
- Fault isolation
- Scalability
- Team autonomy

Event-Driven Architecture:
- Loose coupling
- Real-time processing
- Event sourcing
- CQRS pattern
- Audit trail

API Design:
- RESTful principles
- GraphQL for complex queries
- gRPC for internal services
- OpenAPI documentation
- Versioning strategy

AI/ML Concepts

Multi-Agent Systems:
- Specialized agents for subtasks
- Orchestration layer
- Inter-agent communication
- Consensus mechanisms
- Explanation aggregation

Clinical NLP:
- Named entity recognition (medications, conditions)
- Relation extraction
- Negation detection
- Temporal reasoning
- Section segmentation

Predictive Models:
- Time series forecasting
- Survival analysis
- Causal inference
- Uncertainty quantification
- Model interpretability

Security & Compliance

HIPAA Requirements:
- Access controls
- Audit trails
- Encryption (at rest & transit)
- Minimum necessary
- Business Associate Agreements

FDA SaMD:
- Design controls
- Risk management (ISO 14971)
- Clinical validation
- Post-market surveillance
- Change control

Zero Trust Security:
- Never trust, always verify
- Least privilege access

- Micro-segmentation
- Continuous monitoring
- Identity-centric

3. Implementation Best Practices

Code Quality

```
# Example: Type hints and documentation
from typing import List, Dict, Optional, Union
from datetime import datetime
from pydantic import BaseModel, Field

class GlucoseReading(BaseModel):
    """Represents a blood glucose measurement.

    Attributes:
        value: Glucose value in mg/dL
        timestamp: When the reading was taken
        meal_tag: Optional meal context
        notes: Optional user notes
    """
    value: float = Field(..., ge=20, le=600, description="Glucose in mg/dL")
    timestamp: datetime
    meal_tag: Optional[str] = Field(None, regex="^(before|after)_(breakfast|lunch|dinner|snack)$")
    notes: Optional[str] = Field(None, max_length=500)

    class Config:
        schema_extra = {
            "example": {
                "value": 120,
                "timestamp": "2024-01-15T08:30:00Z",
                "meal_tag": "before_breakfast",
                "notes": "Feeling good"
            }
        }
```

Testing Strategies

```
# Example: Comprehensive test structure
import pytest
from unittest.mock import Mock, AsyncMock
from datetime import datetime, timedelta
```

```
class TestGlucosePrediction:
    """Test suite for glucose prediction functionality"""

    @pytest.fixture
    def mock_cgm_data(self):
        """Generate mock CGM data for testing"""
        base_time = datetime.now()
        return [
            {"timestamp": base_time - timedelta(minutes=5*i),
             "glucose": 120 + 10*np.sin(i/10)}
            for i in range(288)  # 24 hours of data
        ]

    @pytest.mark.asyncio
    async def test_prediction_accuracy(self, mock_cgm_data):
        """Test that predictions are within acceptable range"""
        predictor = GlucosePredictor()
        predictions = await predictor.predict(mock_cgm_data, horizon=60)

        assert predictions is not None
        assert len(predictions) == 12  # 60 minutes / 5 minute intervals
        assert all(20 <= p <= 400 for p in predictions)

    @pytest.mark.parametrize("horizon,expected_length", [
        (30, 6),
        (60, 12),
        (120, 24),
        (240, 48)
    ])
    async def test_prediction_horizons(self, mock_cgm_data, horizon, expected_length):
        """Test predictions for different time horizons"""
        predictor = GlucosePredictor()
        predictions = await predictor.predict(mock_cgm_data, horizon=horizon)
        assert len(predictions) == expected_length
```

Performance Optimization

```
# Example: Caching and async optimization
from functools import lru_cache
from typing import List
import asyncio
import aioredis
```

```
class OptimizedClinicalService:
    def __init__(self):
        self.redis = None
        self._cache = {}

    async def initialize(self):
        """Initialize Redis connection"""
        self.redis = await aioredis.create_redis_pool(
            'redis://localhost',
            encoding='utf-8'
        )

    @lru_cache(maxsize=1000)
    def calculate_risk_score(self, patient_id: str) -> float:
        """Calculate risk score with LRU caching"""
        # Expensive calculation cached in memory
        return self._complex_risk_calculation(patient_id)

    async def get_patient_data_batch(self, patient_ids: List[str]) -> List[dict]:
        """Fetch multiple patients efficiently"""
        # Create tasks for parallel execution
        tasks = [self.get_patient_data(pid) for pid in patient_ids]

        # Execute in parallel with semaphore to limit concurrency
        sem = asyncio.Semaphore(10)
        async def bounded_fetch(patient_id):
            async with sem:
                return await self.get_patient_data(patient_id)

        bounded_tasks = [bounded_fetch(pid) for pid in patient_ids]
        return await asyncio.gather(*bounded_tasks)

    async def get_patient_data(self, patient_id: str) -> dict:
        """Get patient data with Redis caching"""
        # Check Redis cache first
        cached = await self.redis.get(f"patient:{patient_id}")
        if cached:
            return json.loads(cached)

        # Fetch from database
        data = await self._fetch_from_db(patient_id)

        # Cache with expiration
        await self.redis.setex(
```

```
            f"patient:{patient_id}",
            300,  # 5 minute TTL
            json.dumps(data)
        )

        return data
```

Monitoring and Observability

```
# Example: Comprehensive monitoring setup
from prometheus_client import Counter, Histogram, Gauge
from opentelemetry import trace
from opentelemetry.exporter.jaeger import JaegerExporter
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor

# Metrics
recommendation_counter = Counter(
    'clinical_recommendations_total',
    'Total number of clinical recommendations generated',
    ['recommendation_type', 'status']
)

recommendation_latency = Histogram(
    'clinical_recommendation_duration_seconds',
    'Time spent generating recommendations',
    ['recommendation_type']
)

active_users = Gauge(
    'active_users_total',
    'Number of active users in the system'
)

# Tracing
trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer(__name__)

jaeger_exporter = JaegerExporter(
    agent_host_name="localhost",
    agent_port=6831,
)

span_processor = BatchSpanProcessor(jaeger_exporter)
```

```
trace.get_tracer_provider().add_span_processor(span_processor)

class MonitoredClinicalService:
    @recommendation_latency.time()
    async def generate_recommendation(self, patient_id: str) -> dict:
        """Generate recommendation with monitoring"""
        with tracer.start_as_current_span("generate_recommendation") as span:
            span.set_attribute("patient.id", patient_id)

            try:
                # Generate recommendation
                recommendation = await self._generate(patient_id)

                # Record metrics
                recommendation_counter.labels(
                    recommendation_type=recommendation['type'],
                    status='success'
                ).inc()

                span.set_attribute("recommendation.type", recommendation['type'])
                span.set_attribute("recommendation.confidence", recommendation['confidence'])

                return recommendation

            except Exception as e:
                recommendation_counter.labels(
                    recommendation_type='unknown',
                    status='error'
                ).inc()

                span.record_exception(e)
                span.set_status(trace.Status(trace.StatusCode.ERROR))
                raise
```

4. Troubleshooting Guide

Common Issues and Solutions

Issue: High latency in recommendation generation

```
# Diagnosis steps
1. Check service metrics
   curl http://localhost:9090/metrics | grep recommendation_duration
```

```
2. Analyze slow queries
   SELECT query, mean_exec_time, calls
   FROM pg_stat_statements
   ORDER BY mean_exec_time DESC
   LIMIT 10;

3. Profile Python code
   python -m cProfile -s cumulative api_server.py

4. Check Redis cache hit rate
   redis-cli INFO stats | grep keyspace

# Solutions
- Add database indexes on frequently queried columns
- Implement query result caching
- Use connection pooling
- Optimize N+1 queries with eager loading
- Add CDN for static assets
```

Issue: Memory leaks in ML model serving

```
# Detection
import tracemalloc
import gc

tracemalloc.start()

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')

print("[ Top 10 memory allocations ]")
for stat in top_stats[:10]:
    print(stat)

# Solution: Proper cleanup
class ModelServer:
    def __init__(self):
        self.models = {}

    def load_model(self, model_name: str):
        """Load model with proper memory management"""
```

```
        # Clear any existing model
        if model_name in self.models:
            del self.models[model_name]
            gc.collect()
            torch.cuda.empty_cache()  # If using GPU

        # Load new model
        self.models[model_name] = load_model(model_name)

    def __del__(self):
        """Cleanup on deletion"""
        for model in self.models.values():
            del model
        gc.collect()
```

5. Development Resources

Useful Libraries and Tools

```
Python:
  web_frameworks:
    - FastAPI: Modern async web framework
    - Django: Full-featured web framework
    - Flask: Lightweight web framework

  ml_libraries:
    - PyTorch: Deep learning framework
    - scikit-learn: Traditional ML
    - XGBoost: Gradient boosting
    - Ray: Distributed computing

  data_processing:
    - Pandas: Data manipulation
    - NumPy: Numerical computing
    - Polars: Fast DataFrame library
    - Dask: Parallel computing

  testing:
    - pytest: Testing framework
    - hypothesis: Property-based testing
    - locust: Load testing
    - pytest-asyncio: Async testing
```

JavaScript/TypeScript:
  frameworks:
    - Next.js: React framework
    - Remix: Full-stack framework
    - Vite: Build tool

  ui_libraries:
    - Radix UI: Headless components
    - Tailwind CSS: Utility CSS
    - Framer Motion: Animations
    - Recharts: Data visualization

  testing:
    - Jest: Testing framework
    - React Testing Library: Component testing
    - Cypress: E2E testing
    - Playwright: Browser automation

Infrastructure:
  containers:
    - Docker: Containerization
    - Kubernetes: Orchestration
    - Helm: K8s package manager

  monitoring:
    - Prometheus: Metrics
    - Grafana: Visualization
    - Jaeger: Distributed tracing
    - ELK Stack: Logging

  databases:
    - PostgreSQL: Relational DB
    - Neo4j: Graph DB
    - Redis: Cache/Message broker
    - ClickHouse: Analytics DB

API Documentation Standards

openapi: 3.0.0
info:
  title: EASY-Diabetes Clinical API
  version: 2.0.0
  description: Clinical decision support API for diabetes management
  contact:

```
    email: api-support@easy-diabetes.com
  license:
    name: Proprietary

servers:
  - url: https://api.easy-diabetes.com/v2
    description: Production server
  - url: https://staging-api.easy-diabetes.com/v2
    description: Staging server

security:
  - bearerAuth: []
  - oauth2: [read, write]

paths:
  /patients/{patientId}/recommendations:
    post:
      summary: Generate clinical recommendations
      operationId: generateRecommendations
      tags:
        - Clinical Decision Support
      parameters:
        - name: patientId
          in: path
          required: true
          schema:
            type: string
            format: uuid
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/RecommendationRequest'
      responses:
        '200':
          description: Recommendations generated successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/RecommendationResponse'
        '400':
          $ref: '#/components/responses/BadRequest'
        '401':
```

```
      $ref: '#/components/responses/Unauthorized'
    '404':
      $ref: '#/components/responses/NotFound'
    '500':
      $ref: '#/components/responses/InternalError'
```

Git Workflow

```
# Feature branch workflow
git checkout -b feature/EASY-123-glucose-prediction

# Make changes with conventional commits
git commit -m "feat(prediction): add 4-hour glucose forecasting"
git commit -m "test(prediction): add unit tests for glucose predictor"
git commit -m "docs(prediction): update API documentation"

# Keep branch updated
git checkout main
git pull origin main
git checkout feature/EASY-123-glucose-prediction
git rebase main

# Push and create PR
git push origin feature/EASY-123-glucose-prediction

# After review and approval
git checkout main
git merge --no-ff feature/EASY-123-glucose-prediction
git push origin main

# Tag releases
git tag -a v2.1.0 -m "Release version 2.1.0"
git push origin v2.1.0
```

Deployment Checklist

```
## Pre-Deployment Checklist

### Code Quality
- [ ] All tests passing (unit, integration, e2e)
- [ ] Code coverage >80%
- [ ] No critical security vulnerabilities
- [ ] Performance benchmarks met
```

- [ ] Documentation updated

Database
- [ ] Migration scripts tested
- [ ] Rollback plan prepared
- [ ] Indexes optimized
- [ ] Backup completed

Infrastructure
- [ ] Resource limits configured
- [ ] Autoscaling policies set
- [ ] Health checks defined
- [ ] Monitoring alerts configured
- [ ] SSL certificates valid

Security
- [ ] Security scan completed
- [ ] Secrets rotated
- [ ] Access controls verified
- [ ] Audit logging enabled
- [ ] WAF rules updated

Business
- [ ] Stakeholders notified
- [ ] Downtime window scheduled
- [ ] Support team briefed
- [ ] Rollback criteria defined
- [ ] Success metrics identified

Document 7: Development Guidelines and Best Practices

Code Style Guide

Python Style Guide

"""
EASY-Diabetes Python Style Guide
Based on PEP 8 with project-specific conventions

"""

# Imports
# Standard library imports first
import os
import sys
from datetime import datetime
from typing import List, Dict, Optional, Union

# Third-party imports
import numpy as np
import pandas as pd
from fastapi import Fast

"""
EASY-Diabetes Python Style Guide (continued)
Based on PEP 8 with project-specific conventions
"""

# Imports (continued)
# Standard library imports first
import os
import sys
from datetime import datetime
from typing import List, Dict, Optional, Union

# Third-party imports
import numpy as np
import pandas as pd
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from sqlalchemy.ext.asyncio import AsyncSession

# Local imports
from app.core.config import settings
from app.models.patient import Patient
from app.services.clinical_engine import ClinicalEngine

# Class definitions
class ClinicalRecommendation(BaseModel):
    """
    Clinical recommendation model.

```
    Attributes:
        id: Unique identifier
        patient_id: Patient UUID
        recommendation_type: Type of recommendation
        priority: Clinical priority (high, medium, low)
        evidence_level: Strength of evidence (A, B, C, D)
        confidence_score: ML model confidence (0-1)
    """
    id: str = Field(..., description="Unique recommendation ID")
    patient_id: str = Field(..., description="Patient UUID")
    recommendation_type: str = Field(..., regex="^(medication|lifestyle|monitoring)$")
    priority: str = Field(..., regex="^(high|medium|low)$")
    evidence_level: str = Field(..., regex="^[A-D]$")
    confidence_score: float = Field(..., ge=0, le=1)

    class Config:
        schema_extra = {
            "example": {
                "id": "rec_123456",
                "patient_id": "550e8400-e29b-41d4-a716-446655440000",
                "recommendation_type": "medication",
                "priority": "high",
                "evidence_level": "A",
                "confidence_score": 0.92
            }
        }


# Function definitions
async def calculate_insulin_dose(
    current_glucose: float,
    target_glucose: float,
    insulin_sensitivity_factor: float,
    active_insulin: float = 0
) -> float:
    """
    Calculate insulin correction dose.

    Args:
        current_glucose: Current blood glucose in mg/dL
        target_glucose: Target blood glucose in mg/dL
        insulin_sensitivity_factor: 1 unit drops BG by X mg/dL
        active_insulin: Units of insulin still active
```

```
    Returns:
        Recommended insulin dose in units

    Raises:
        ValueError: If glucose values are out of valid range

    Example:
        >>> dose = await calculate_insulin_dose(
        ...     current_glucose=180,
        ...     target_glucose=120,
        ...     insulin_sensitivity_factor=50,
        ...     active_insulin=0.5
        ... )
        >>> print(f"Recommended dose: {dose} units")
        Recommended dose: 0.7 units
    """
    if not 40 <= current_glucose <= 600:
        raise ValueError(f"Current glucose {current_glucose} out of valid range")

    if not 70 <= target_glucose <= 180:
        raise ValueError(f"Target glucose {target_glucose} out of valid range")

    # Calculate correction dose
    glucose_delta = current_glucose - target_glucose
    correction_dose = glucose_delta / insulin_sensitivity_factor

    # Account for active insulin
    net_dose = max(0, correction_dose - active_insulin)

    # Round to nearest 0.5 unit
    return round(net_dose * 2) / 2


# Constants and configuration
class DiabetesConstants:
    """Central location for diabetes-related constants."""

    # Glucose thresholds (mg/dL)
    HYPOGLYCEMIA_LEVEL_1 = 70
    HYPOGLYCEMIA_LEVEL_2 = 54
    TARGET_RANGE_LOW = 70
    TARGET_RANGE_HIGH = 180
    HYPERGLYCEMIA_LEVEL_1 = 180
    HYPERGLYCEMIA_LEVEL_2 = 250
```

```
    # HbA1c targets (%)
    GENERAL_A1C_TARGET = 7.0
    INTENSIVE_A1C_TARGET = 6.5
    RELAXED_A1C_TARGET = 8.0

    # Time in range targets (%)
    TIR_TARGET_MINIMUM = 70
    TBR_TARGET_MAXIMUM = 4
    TAR_TARGET_MAXIMUM = 25


# Error handling
class ClinicalError(Exception):
    """Base exception for clinical errors."""
    pass


class ContraindicationError(ClinicalError):
    """Raised when a contraindication is detected."""

    def __init__(self, medication: str, reason: str):
        self.medication = medication
        self.reason = reason
        super().__init__(f"Contraindication for {medication}: {reason}")


class InsufficientDataError(ClinicalError):
    """Raised when insufficient data for clinical decision."""

    def __init__(self, data_type: str, minimum_required: int, actual: int):
        self.data_type = data_type
        self.minimum_required = minimum_required
        self.actual = actual
        super().__init__(
            f"Insufficient {data_type} data: required {minimum_required}, got {actual}"
        )


# Async context managers
class DatabaseTransaction:
    """Async context manager for database transactions."""

    def __init__(self, session: AsyncSession):
```

```
        self.session = session

    async def __aenter__(self):
        await self.session.begin()
        return self.session

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        if exc_type:
            await self.session.rollback()
        else:
            await self.session.commit()
        await self.session.close()


# Type hints for complex types
from typing import TypedDict, Protocol, Literal

class GlucoseReading(TypedDict):
    """Type definition for glucose readings."""
    timestamp: datetime
    value: float
    unit: Literal["mg/dL", "mmol/L"]
    source: Literal["cgm", "fingerstick", "lab"]


class PredictionModel(Protocol):
    """Protocol for prediction models."""

    async def predict(self, data: pd.DataFrame) -> np.ndarray:
        """Generate predictions from input data."""
        ...

    def get_feature_importance(self) -> Dict[str, float]:
        """Return feature importance scores."""
        ...
```

TypeScript/React Style Guide

```
/**
 * EASY-Diabetes TypeScript/React Style Guide
 * Based on Airbnb style guide with project conventions
 */

// Import organization
```

```
// 1. React imports
import React, { useState, useEffect, useCallback, useMemo } from 'react'
import { useRouter } from 'next/router'

// 2. Third-party libraries
import { useQuery, useMutation } from '@tanstack/react-query'
import { z } from 'zod'
import { format, parseISO } from 'date-fns'

// 3. Local components
import { Button } from '@/components/ui/button'
import { Card, CardContent, CardHeader } from '@/components/ui/card'

// 4. Utils and helpers
import { api } from '@/lib/api'
import { cn } from '@/lib/utils'

// 5. Types
import type { Patient, ClinicalData, Medication } from '@/types/clinical'


// Component definition with proper typing
interface PatientDashboardProps {
  patientId: string
  initialData?: Patient
  onUpdate?: (patient: Patient) => void
}

export const PatientDashboard: React.FC<PatientDashboardProps> = ({
  patientId,
  initialData,
  onUpdate
}) => {
  // State management
  const [isEditing, setIsEditing] = useState(false)
  const [selectedMedication, setSelectedMedication] = useState<Medication | null>(null)

  // Custom hooks
  const router = useRouter()

  // Data fetching with React Query
  const { data: patient, isLoading, error } = useQuery({
    queryKey: ['patient', patientId],
    queryFn: () => api.patients.get(patientId),
```

```
  initialData,
  staleTime: 5 * 60 * 1000, // 5 minutes
})

// Mutations
const updatePatient = useMutation({
  mutationFn: (data: Partial<Patient>) =>
    api.patients.update(patientId, data),
  onSuccess: (updatedPatient) => {
    onUpdate?.(updatedPatient)
    setIsEditing(false)
  },
})

// Memoized calculations
const riskScore = useMemo(() => {
  if (!patient) return null
  return calculateRiskScore(patient)
}, [patient])

// Callbacks to prevent re-renders
const handleEdit = useCallback(() => {
  setIsEditing(true)
}, [])

const handleSave = useCallback(async (data: Partial<Patient>) => {
  await updatePatient.mutateAsync(data)
}, [updatePatient])

// Effects
useEffect(() => {
  if (error) {
    console.error('Failed to load patient:', error)
    router.push('/patients')
  }
}, [error, router])

// Loading state
if (isLoading) {
  return <PatientDashboardSkeleton />
}

// Error state
if (error) {
```

```
    return (
      <Card>
        <CardContent>
          <p className="text-destructive">
            Failed to load patient data. Please try again.
          </p>
        </CardContent>
      </Card>
    )
  }

  // Main render
  return (
    <div className="space-y-6">
      <PatientHeader
        patient={patient!}
        isEditing={isEditing}
        onEdit={handleEdit}
        onSave={handleSave}
      />

      <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
        <GlycemicControlCard patientId={patientId} />
        <MedicationsCard
          medications={patient!.medications}
          onSelect={setSelectedMedication}
        />
        <RiskAssessmentCard score={riskScore} />
      </div>

      {selectedMedication && (
        <MedicationDetailsDialog
          medication={selectedMedication}
          onClose={() => setSelectedMedication(null)}
        />
      )}
    </div>
  )
}

// Subcomponents with proper prop types
interface GlycemicControlCardProps {
  patientId: string
}
```

```
const GlycemicControlCard: React.FC<GlycemicControlCardProps> = ({
  patientId
}) => {
  const { data: glucoseData } = useQuery({
    queryKey: ['glucose', patientId, 'recent'],
    queryFn: () => api.glucose.getRecent(patientId),
  })

  return (
    <Card>
      <CardHeader>
        <h3 className="text-lg font-semibold">Glycemic Control</h3>
      </CardHeader>
      <CardContent>
        {glucoseData ? (
          <GlucoseChart data={glucoseData} />
        ) : (
          <div className="h-32 animate-pulse bg-muted rounded" />
        )}
      </CardContent>
    </Card>
  )
}

// Utility functions with proper typing
function calculateRiskScore(patient: Patient): number {
  let score = 0

  // Age factor
  const age = calculateAge(patient.dateOfBirth)
  if (age > 65) score += 2
  else if (age > 45) score += 1

  // Comorbidity factors
  if (patient.conditions.includes('hypertension')) score += 1
  if (patient.conditions.includes('dyslipidemia')) score += 1

  // Recent HbA1c
  const recentA1c = patient.labs.hba1c[0]?.value
  if (recentA1c > 9) score += 3
  else if (recentA1c > 8) score += 2
  else if (recentA1c > 7) score += 1
```

```
  return Math.min(score, 10) // Cap at 10
}

// Custom hooks
export function useGlucosePrediction(patientId: string) {
  const [prediction, setPrediction] = useState<GlucosePrediction | null>(null)
  const [isLoading, setIsLoading] = useState(false)

  const generatePrediction = useCallback(async () => {
    setIsLoading(true)
    try {
      const result = await api.predictions.generateGlucose(patientId)
      setPrediction(result)
    } catch (error) {
      console.error('Prediction failed:', error)
    } finally {
      setIsLoading(false)
    }
  }, [patientId])

  useEffect(() => {
    generatePrediction()

    // Refresh every 5 minutes
    const interval = setInterval(generatePrediction, 5 * 60 * 1000)
    return () => clearInterval(interval)
  }, [generatePrediction])

  return { prediction, isLoading, refresh: generatePrediction }
}

// Zod schemas for validation
const MedicationFormSchema = z.object({
  name: z.string().min(1, 'Medication name is required'),
  dose: z.number().positive('Dose must be positive'),
  unit: z.enum(['mg', 'g', 'mcg', 'units', 'mL']),
  frequency: z.enum(['daily', 'bid', 'tid', 'qid', 'prn']),
  route: z.enum(['oral', 'subcutaneous', 'intravenous', 'intramuscular']),
  startDate: z.date(),
  endDate: z.date().optional(),
})

type MedicationFormData = z.infer<typeof MedicationFormSchema>
```

```
// Constants
export const GLUCOSE_THRESHOLDS = {
  hypoglycemia: {
    level1: 70,
    level2: 54,
  },
  target: {
    low: 70,
    high: 180,
  },
  hyperglycemia: {
    level1: 180,
    level2: 250,
  },
} as const

// Enums with const assertion
export const MedicationClass = {
  BIGUANIDE: 'biguanide',
  SULFONYLUREA: 'sulfonylurea',
  DPP4_INHIBITOR: 'dpp4_inhibitor',
  SGLT2_INHIBITOR: 'sglt2_inhibitor',
  GLP1_AGONIST: 'glp1_agonist',
  INSULIN: 'insulin',
} as const

export type MedicationClass = typeof MedicationClass[keyof typeof MedicationClass]
```

SQL Style Guide

```
-- EASY-Diabetes SQL Style Guide
-- Consistent formatting for maintainable database code

-- Table creation with proper structure
CREATE TABLE clinical.patient_medications (
    -- Primary key first
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),

    -- Foreign keys
    patient_id UUID NOT NULL REFERENCES patient.patients(id) ON DELETE CASCADE,
    medication_id UUID NOT NULL REFERENCES clinical.medications(id),
    prescriber_id UUID REFERENCES auth.users(id),

    -- Required fields
```

```
    dose NUMERIC(10,3) NOT NULL CHECK (dose > 0),
    dose_unit VARCHAR(20) NOT NULL,
    frequency VARCHAR(50) NOT NULL,
    route VARCHAR(30) NOT NULL,
    start_date DATE NOT NULL,

    -- Optional fields
    end_date DATE,
    discontinuation_reason TEXT,
    instructions TEXT,

    -- Metadata
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    created_by UUID REFERENCES auth.users(id),

    -- Constraints
    CONSTRAINT valid_date_range CHECK (end_date IS NULL OR end_date >= start_date),
    CONSTRAINT valid_dose_unit CHECK (dose_unit IN ('mg', 'g', 'mcg', 'units', 'mL'))
);

-- Indexes for performance
CREATE INDEX idx_patient_medications_patient_active
    ON clinical.patient_medications(patient_id, is_active)
    WHERE is_active = true;

CREATE INDEX idx_patient_medications_dates
    ON clinical.patient_medications(patient_id, start_date, end_date);

-- Comments for documentation
COMMENT ON TABLE clinical.patient_medications IS 'Tracks all patient medications with
dosing information';
COMMENT ON COLUMN clinical.patient_medications.dose IS 'Medication dose amount';
COMMENT ON COLUMN clinical.patient_medications.frequency IS 'Dosing frequency (e.g.,
daily, BID, TID)';

-- Complex queries with CTEs for readability
WITH active_medications AS (
    -- Get all active medications for patients
    SELECT
        pm.patient_id,
        pm.medication_id,
        m.generic_name,
```

```
        m.drug_class,
        pm.dose,
        pm.dose_unit,
        pm.frequency,
        pm.start_date,
        DATE_PART('day', CURRENT_DATE - pm.start_date) AS days_on_medication
    FROM clinical.patient_medications pm
    INNER JOIN clinical.medications m ON m.id = pm.medication_id
    WHERE pm.is_active = true
      AND pm.end_date IS NULL
),
medication_counts AS (
    -- Count medications by class
    SELECT
        patient_id,
        drug_class,
        COUNT(*) AS medication_count,
        STRING_AGG(generic_name, ', ' ORDER BY generic_name) AS medications
    FROM active_medications
    GROUP BY patient_id, drug_class
),
patient_summary AS (
    -- Summarize patient medication profile
    SELECT
        p.id AS patient_id,
        p.external_id,
        COUNT(DISTINCT am.medication_id) AS total_medications,
        COUNT(DISTINCT am.drug_class) AS unique_drug_classes,
        MAX(am.days_on_medication) AS longest_medication_duration
    FROM patient.patients p
    LEFT JOIN active_medications am ON am.patient_id = p.id
    WHERE p.is_active = true
    GROUP BY p.id, p.external_id
)
-- Final query combining CTEs
SELECT
    ps.external_id,
    ps.total_medications,
    ps.unique_drug_classes,
    ps.longest_medication_duration,
    mc.drug_class,
    mc.medication_count,
    mc.medications
FROM patient_summary ps
```

```
LEFT JOIN medication_counts mc ON mc.patient_id = ps.patient_id
ORDER BY ps.external_id, mc.drug_class;

-- Function with proper error handling
CREATE OR REPLACE FUNCTION clinical.calculate_medication_adherence(
    p_patient_id UUID,
    p_medication_id UUID,
    p_start_date DATE DEFAULT CURRENT_DATE - INTERVAL '30 days',
    p_end_date DATE DEFAULT CURRENT_DATE
)
RETURNS TABLE (
    adherence_rate NUMERIC,
    expected_doses INTEGER,
    actual_doses INTEGER,
    missed_doses INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_frequency VARCHAR(50);
    v_daily_doses INTEGER;
BEGIN
    -- Validate inputs
    IF p_end_date <= p_start_date THEN
        RAISE EXCEPTION 'End date must be after start date';
    END IF;

    -- Get medication frequency
    SELECT pm.frequency INTO v_frequency
    FROM clinical.patient_medications pm
    WHERE pm.patient_id = p_patient_id
      AND pm.medication_id = p_medication_id
      AND pm.is_active = true;

    IF v_frequency IS NULL THEN
        RAISE EXCEPTION 'No active medication found for patient % and medication %',
            p_patient_id, p_medication_id;
    END IF;

    -- Calculate daily doses based on frequency
    v_daily_doses := CASE v_frequency
        WHEN 'daily' THEN 1
        WHEN 'bid' THEN 2
        WHEN 'tid' THEN 3
```

```
        WHEN 'qid' THEN 4
        ELSE 1 -- Default to daily
    END;

    -- Calculate adherence
    RETURN QUERY
    WITH dose_counts AS (
        SELECT
            COUNT(*) AS actual_doses
        FROM clinical.medication_administrations ma
        WHERE ma.patient_id = p_patient_id
          AND ma.medication_id = p_medication_id
          AND ma.administered_at::date BETWEEN p_start_date AND p_end_date
    )
    SELECT
        ROUND((dc.actual_doses::numeric /
               (v_daily_doses * (p_end_date - p_start_date + 1))::numeric) * 100, 2) AS
adherence_rate,
        v_daily_doses * (p_end_date - p_start_date + 1) AS expected_doses,
        dc.actual_doses::integer AS actual_doses,
        (v_daily_doses * (p_end_date - p_start_date + 1) - dc.actual_doses)::integer AS
missed_doses
    FROM dose_counts dc;
END;
$$;

-- Grant appropriate permissions
GRANT EXECUTE ON FUNCTION clinical.calculate_medication_adherence TO clinical_user;
```

Testing Strategy

Unit Testing

```
# tests/unit/test_glucose_prediction.py
import pytest
from datetime import datetime, timedelta
import numpy as np
from unittest.mock import Mock, patch

from app.services.prediction import GlucosePredictor
from app.models.glucose import GlucoseReading
from app.exceptions import InsufficientDataError
```

```
class TestGlucosePredictor:
    """Comprehensive unit tests for glucose prediction."""

    @pytest.fixture
    def predictor(self):
        """Create predictor instance for testing."""
        return GlucosePredictor()

    @pytest.fixture
    def sample_cgm_data(self):
        """Generate sample CGM data for testing."""
        base_time = datetime.now()
        readings = []

        # Generate 24 hours of 5-minute readings
        for i in range(288):
            time = base_time - timedelta(minutes=5*i)
            # Simulate realistic glucose pattern
            base_glucose = 120
            meal_effect = 30 * np.exp(-((i-60)**2)/1000) if 50 < i < 70 else 0
            circadian = 10 * np.sin(2*np.pi*i/288)
            noise = np.random.normal(0, 5)

            glucose = base_glucose + meal_effect + circadian + noise
            glucose = np.clip(glucose, 40, 400)

            readings.append(GlucoseReading(
                timestamp=time,
                value=glucose,
                source='cgm'
            ))

        return readings

    def test_prediction_with_sufficient_data(self, predictor, sample_cgm_data):
        """Test prediction with adequate historical data."""
        # Act
        predictions = predictor.predict(
            historical_data=sample_cgm_data,
            horizon_minutes=60
        )

        # Assert
```

```
    assert len(predictions) == 12  # 60 min / 5 min intervals
    assert all(40 <= p.value <= 400 for p in predictions)
    assert all(p.confidence > 0.7 for p in predictions[:6])  # Higher confidence for near term

def test_prediction_with_insufficient_data(self, predictor):
    """Test that prediction fails with insufficient data."""
    # Arrange
    insufficient_data = [
        GlucoseReading(
            timestamp=datetime.now(),
            value=120,
            source='cgm'
        )
    ]

    # Act & Assert
    with pytest.raises(InsufficientDataError) as exc_info:
        predictor.predict(insufficient_data, horizon_minutes=60)

    assert exc_info.value.minimum_required == 12  # 1 hour of data minimum
    assert exc_info.value.actual == 1

@pytest.mark.parametrize("horizon,expected_confidence", [
    (15, 0.95),  # 15 min - very high confidence
    (30, 0.90),  # 30 min - high confidence
    (60, 0.80),  # 60 min - moderate confidence
    (120, 0.65), # 120 min - lower confidence
    (240, 0.50), # 240 min - uncertainty threshold
])
def test_confidence_decreases_with_horizon(
    self, predictor, sample_cgm_data, horizon, expected_confidence
):
    """Test that prediction confidence decreases with longer horizons."""
    # Act
    predictions = predictor.predict(sample_cgm_data, horizon)

    # Assert
    avg_confidence = np.mean([p.confidence for p in predictions])
    assert abs(avg_confidence - expected_confidence) < 0.1

def test_hypoglycemia_detection(self, predictor):
    """Test that predictor detects impending hypoglycemia."""
    # Arrange - Create declining glucose trend
    base_time = datetime.now()
```

```
        declining_data = []

        for i in range(24):  # 2 hours of data
            time = base_time - timedelta(minutes=5*i)
            glucose = 120 - (3 * i)  # Declining by 3 mg/dL every 5 min
            declining_data.append(GlucoseReading(
                timestamp=time,
                value=max(glucose, 40),
                source='cgm'
            ))

        # Act
        predictions = predictor.predict(declining_data, horizon_minutes=30)
        alerts = predictor.detect_hypoglycemia_risk(predictions)

        # Assert
        assert any(alert.severity == 'high' for alert in alerts)
        assert any(p.value < 70 for p in predictions)

    @patch('app.services.prediction.ml_model')
    def test_model_fallback_on_error(self, mock_model, predictor, sample_cgm_data):
        """Test fallback to statistical model if ML model fails."""
        # Arrange
        mock_model.predict.side_effect = RuntimeError("Model error")

        # Act
        predictions = predictor.predict(sample_cgm_data, horizon_minutes=30)

        # Assert
        assert len(predictions) == 6  # Still returns predictions
        assert all(p.model_type == 'statistical_fallback' for p in predictions)
```

Integration Testing

```
# tests/integration/test_clinical_workflow.py
import pytest
from httpx import AsyncClient
from sqlalchemy.ext.asyncio import AsyncSession

from app.main import app
from app.models.patient import Patient
from app.models.clinical import ClinicalData
from tests.factories import PatientFactory, ClinicalDataFactory
```

```
@pytest.mark.asyncio
class TestClinicalWorkflow:
    """Integration tests for complete clinical workflows."""

    async def test_complete_recommendation_workflow(
        self,
        async_client: AsyncClient,
        db_session: AsyncSession,
        authenticated_headers: dict
    ):
        """Test end-to-end recommendation generation workflow."""
        # Arrange - Create test patient with clinical data
        patient = await PatientFactory.create(db_session)
        clinical_data = await ClinicalDataFactory.create_batch(
            db_session,
            size=10,
            patient_id=patient.id
        )

        # Act - Generate recommendations
        response = await async_client.post(
            f"/api/v1/patients/{patient.id}/recommendations",
            json={
                "include_categories": ["medication", "lifestyle", "monitoring"],
                "urgency": "routine",
                "context": "regular_checkup"
            },
            headers=authenticated_headers
        )

        # Assert - Check response
        assert response.status_code == 200
        recommendations = response.json()

        assert "recommendations" in recommendations
        assert len(recommendations["recommendations"]) > 0
        assert all(
            rec["evidence_level"] in ["A", "B", "C", "D"]
            for rec in recommendations["recommendations"]
        )
        assert recommendations["risk_assessment"] is not None
        assert recommendations["confidence_scores"]["overall"] > 0.7
```

```
    # Verify audit trail created
    audit_response = await async_client.get(
        f"/api/v1/audit/patient/{patient.id}",
        headers=authenticated_headers
    )
    assert audit_response.status_code == 200
    audit_entries = audit_response.json()
    assert any(
        entry["action"] == "generate_recommendations"
        for entry in audit_entries
    )

async def test_medication_optimization_with_contraindications(
    self,
    async_client: AsyncClient,
    db_session: AsyncSession,
    authenticated_headers: dict
):
    """Test medication optimization handles contraindications correctly."""
    # Arrange - Patient with CKD contraindicating metformin
    patient = await PatientFactory.create(
        db_session,
        conditions=[{
            "code": "N18.5",
            "display": "Chronic kidney disease, stage 5"
        }]
    )

    # Current medications including metformin
    await db_session.execute(
        """
        INSERT INTO clinical.patient_medications
        (patient_id, medication_id, dose, dose_unit, frequency, route, start_date)
        VALUES
        (:patient_id, :medication_id, 1000, 'mg', 'bid', 'oral', CURRENT_DATE)
        """,
        {
            "patient_id": patient.id,
            "medication_id": "med_metformin_uuid"
        }
    )
    await db_session.commit()

    # Act - Request medication optimization
```

```
    response = await async_client.post(
        f"/api/v1/patients/{patient.id}/medications/optimize",
        headers=authenticated_headers
    )

    # Assert
    assert response.status_code == 200
    optimization = response.json()

    # Should recommend discontinuing metformin
    assert any(
        rec["action"] == "discontinue" and
        rec["medication"] == "metformin" and
        "contraindicated" in rec["reason"].lower()
        for rec in optimization["recommendations"]
    )

    # Should suggest alternatives
    alternatives = [
        rec for rec in optimization["recommendations"]
        if rec["action"] == "initiate"
    ]
    assert len(alternatives) > 0
    assert all(
        alt["safety_check"]["renal_safe"]
        for alt in alternatives
    )
```

End-to-End Testing

```
// tests/e2e/clinical-dashboard.spec.ts
import { test, expect } from '@playwright/test'
import { loginAs } from './helpers/auth'
import { createTestPatient } from './helpers/data'

test.describe('Clinical Dashboard E2E', () => {
  test.beforeEach(async ({ page }) => {
    // Login as clinician
    await loginAs(page, 'clinician@test.com', 'testpass123')
  })

  test('complete clinical workflow', async ({ page }) => {
    // Create test patient
    const patient = await createTestPatient({
```

```
    name: 'John Doe',
    mrn: 'TEST-12345',
    diagnosis: 'Type 2 Diabetes',
    hba1c: 8.5
  })

  // Navigate to patient dashboard
  await page.goto(`/patients/${patient.id}`)

  // Verify patient information displayed
  await expect(page.locator('h1')).toContainText('John Doe')
  await expect(page.locator('[data-testid="mrn"]')).toContainText('TEST-12345')

  // Check glycemic control section
  await page.click('text=Glycemic Control')
  await expect(page.locator('[data-testid="current-hba1c"]')).toContainText('8.5%')

  // Generate recommendations
  await page.click('button:has-text("Generate Recommendations")')

  // Wait for recommendations to load
  await expect(page.locator('[data-testid="recommendations-loading"]')).toBeVisible()
  await expect(page.locator('[data-testid="recommendations-list"]')).toBeVisible({
    timeout: 10000
  })

  // Verify recommendations displayed
  const recommendations = page.locator('[data-testid="recommendation-card"]')
  await expect(recommendations).toHaveCount(3, { timeout: 5000 })

  // Test accepting a recommendation
  await recommendations.first().locator('button:has-text("Accept")').click()

  // Verify confirmation dialog
  await expect(page.locator('[role="dialog"]')).toContainText('Confirm Recommendation')
  await page.click('button:has-text("Confirm")')

  // Verify success message
  await expect(page.locator('[data-testid="toast"]')).toContainText('Recommendation accepted')

  // Check that recommendation is marked as accepted
  await expect(recommendations.first()).toHaveAttribute('data-status', 'accepted')
})
```

```
  test('real-time glucose monitoring', async ({ page }) => {
    const patient = await createTestPatient({
      hasCGM: true,
      cgmDevice: 'Dexcom G7'
    })

    await page.goto(`/patients/${patient.id}/glucose`)

    // Verify CGM data is displayed
    await expect(page.locator('[data-testid="cgm-chart"]')).toBeVisible()
    await expect(page.locator('[data-testid="current-glucose"]')).toContainText(/\d+ mg\/dL/)

    // Test prediction feature
    await page.click('button:has-text("Show Prediction")')
    await expect(page.locator('[data-testid="glucose-prediction"]')).toBeVisible()

    // Verify prediction confidence
    const confidenceText = await
page.locator('[data-testid="prediction-confidence"]').textContent()
    const confidence = parseFloat(confidenceText?.match(/(\d+)%/)?.[1] || '0')
    expect(confidence).toBeGreaterThan(70)

    // Test alert settings
    await page.click('button:has-text("Alert Settings")')
    await page.fill('#low-alert-threshold', '75')
    await page.fill('#high-alert-threshold', '170')
    await page.click('button:has-text("Save Settings")')

    await expect(page.locator('[data-testid="toast"]')).toContainText('Alert settings updated')
  })
})
```

Deployment and Operations

Docker Configuration

```
# Base image for Python services
FROM python:3.11-slim as python-base

# Install system dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
```

```
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Set environment variables
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PIP_NO_CACHE_DIR=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

# Install Python dependencies
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Production image for clinical API
FROM python-base as clinical-api

# Copy application code
COPY ./services/clinical-api /app

# Create non-root user
RUN useradd -m -u 1000 appuser && chown -R appuser:appuser /app
USER appuser

# Health check
HEALTHCHECK --interval=30s --timeout=3s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

# Run application
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

# ML model serving image
FROM python-base as model-server

# Install ML-specific dependencies
RUN pip install --no-cache-dir \
    torch==2.1.0 \
    transformers==4.35.0 \
    onnxruntime==1.16.0

# Copy model files and code
COPY ./services/model-server /app
COPY ./models /models
```

```
# Create non-root user
RUN useradd -m -u 1000 appuser && chown -R appuser:appuser /app /models
USER appuser

# Configure model cache
ENV TRANSFORMERS_CACHE=/models/cache \
    TORCH_HOME=/models/torch

# Run model server
CMD ["python", "-m", "app.server", "--port", "8001"]
```

Kubernetes Manifests

```
# kubernetes/base/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: clinical-api
  labels:
    app: clinical-api
    version: v2.0.0
spec:
  replicas: 3
  selector:
    matchLabels:
      app: clinical-api
  template:
    metadata:
      labels:
        app: clinical-api
        version: v2.0.0
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8000"
        prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: clinical-api
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 1000
      containers:
      - name: clinical-api
        image: easy-diabetes/clinical-api:v2.0.0
```

```
imagePullPolicy: IfNotPresent
ports:
- name: http
  containerPort: 8000
  protocol: TCP
env:
- name: ENV
  value: "production"
- name: DATABASE_URL
  valueFrom:
    secretKeyRef:
      name: clinical-api-secrets
      key: database-url
- name: REDIS_URL
  valueFrom:
    secretKeyRef:
      name: clinical-api-secrets
      key: redis-url
resources:
  requests:
    memory: "512Mi"
    cpu: "250m"
  limits:
    memory: "1Gi"
    cpu: "1000m"
livenessProbe:
  httpGet:
    path: /health
    port: http
  initialDelaySeconds: 30
  periodSeconds: 10
  timeoutSeconds: 3
  failureThreshold: 3
readinessProbe:
  httpGet:
    path: /ready
    port: http
  initialDelaySeconds: 5
  periodSeconds: 5
  timeoutSeconds: 3
  successThreshold: 1
volumeMounts:
- name: config
  mountPath: /app/config
```

```
          readOnly: true
      volumes:
      - name: config
        configMap:
          name: clinical-api-config
---
apiVersion: v1
kind: Service
metadata:
  name: clinical-api
  labels:
    app: clinical-api
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: clinical-api
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: clinical-api-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: clinical-api
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
```

```
      type: Utilization
      averageUtilization: 80
  - type: Pods
    pods:
      metric:
        name: http_requests_per_second
      target:
        type: AverageValue
        averageValue: "1000"
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 10
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
      - type: Percent
        value: 50
        periodSeconds: 60
      - type: Pods
        value: 2
        periodSeconds: 60
```

CI/CD Pipeline

```
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches: [main]
  workflow_dispatch:

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  test:
    runs-on: ubuntu-latest
    services:
```

```
    postgres:
      image: postgres:16
      env:
        POSTGRES_PASSWORD: testpass
      options: >-
        --health-cmd pg_isready
        --health-interval 10s
        --health-timeout 5s
        --health-retries 5
    redis:
      image: redis:7-alpine
      options: >-
        --health-cmd "redis-cli ping"
        --health-interval 10s
        --health-timeout 5s
        --health-retries 5

  steps:
  - uses: actions/checkout@v4

  - name: Set up Python
    uses: actions/setup-python@v4
    with:
      python-version: '3.11'
      cache: 'pip'

  - name: Install dependencies
    run: |
      python -m pip install --upgrade pip
      pip install -r requirements-test.txt

  - name: Run tests
    env:
      DATABASE_URL: postgresql://postgres:testpass@localhost/test_db
      REDIS_URL: redis://localhost:6379
    run: |
      pytest -v --cov=app --cov-report=xml

  - name: Upload coverage
    uses: codecov/codecov-action@v3
    with:
      file: ./coverage.xml

build:
```

```
  needs: test
  runs-on: ubuntu-latest
  permissions:
    contents: read
    packages: write

  steps:
  - uses: actions/checkout@v4

  - name: Set up Docker Buildx
    uses: docker/setup-buildx-action@v3

  - name: Log in to Container Registry
    uses: docker/login-action@v3
    with:
      registry: ${{ env.REGISTRY }}
      username: ${{ github.actor }}
      password: ${{ secrets.GITHUB_TOKEN }}

  - name: Extract metadata
    id: meta
    uses: docker/metadata-action@v5
    with:
      images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
      tags: |
        type=ref,event=branch
        type=ref,event=pr
        type=semver,pattern={{version}}
        type=semver,pattern={{major}}.{{minor}}
        type=sha,prefix={{branch}}-

  - name: Build and push Docker image
    uses: docker/build-push-action@v5
    with:
      context: .
      platforms: linux/amd64,linux/arm64
      push: true
      tags: ${{ steps.meta.outputs.tags }}
      labels: ${{ steps.meta.outputs.labels }}
      cache-from: type=gha
      cache-to: type=gha,mode=max

deploy:
  needs: build
```

```
    runs-on: ubuntu-latest
    environment: production

    steps:
    - uses: actions/checkout@v4

    - name: Install kubectl
      uses: azure/setup-kubectl@v3
      with:
        version: 'v1.28.0'

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: us-west-2

    - name: Update kubeconfig
      run: |
        aws eks update-kubeconfig --name easy-diabetes-prod --region us-west-2

    - name: Deploy to Kubernetes
      run: |
        kubectl set image deployment/clinical-api \
          clinical-api=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }} \
          -n production

        kubectl rollout status deployment/clinical-api -n production

    - name: Run smoke tests
      run: |
        ./scripts/smoke-tests.sh https://api.easy-diabetes.com
```

Monitoring and Alerting

```
# prometheus/alerts.yml
groups:
  - name: clinical_api_alerts
    interval: 30s
    rules:
      - alert: HighErrorRate
        expr: |
          rate(http_requests_total{status=~"5.."}[5m]) > 0.05
```

```
    for: 5m
    labels:
      severity: critical
      service: clinical-api
    annotations:
      summary: "High error rate detected"
      description: "Error rate is {{ $value | humanizePercentage }} for {{ $labels.instance }}"

  - alert: HighLatency
    expr: |
      histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m])) > 1
    for: 5m
    labels:
      severity: warning
      service: clinical-api
    annotations:
      summary: "High API latency detected"
      description: "95th percentile latency is {{ $value }}s for {{ $labels.instance }}"

  - alert: PredictionServiceDown
    expr: |
      up{job="prediction-service"} == 0
    for: 1m
    labels:
      severity: critical
      service: prediction-service
    annotations:
      summary: "Prediction service is down"
      description: "Prediction service instance {{ $labels.instance }} is down"

  - alert: DatabaseConnectionPoolExhausted
    expr: |
      postgresql_connections_active / postgresql_connections_max > 0.9
    for: 5m
    labels:
      severity: warning
      service: database
    annotations:
      summary: "Database connection pool nearly exhausted"
      description: "{{ $value | humanizePercentage }} of connections are in use"

  - alert: HighMemoryUsage
    expr: |
      container_memory_usage_bytes / container_spec_memory_limit_bytes > 0.9
```

```
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Container memory usage is high"
          description: "Memory usage is {{ $value | humanizePercentage }} for {{ $labels.pod }}"

    - name: clinical_sla_alerts
      interval: 1m
      rules:
        - alert: RecommendationGenerationSlow
          expr: |
            histogram_quantile(0.99,
rate(recommendation_generation_duration_seconds_bucket[5m])) > 5
          for: 10m
          labels:
            severity: warning
            sla: performance
          annotations:
            summary: "Recommendation generation is slow"
            description: "99th percentile generation time is {{ $value }}s"

        - alert: CriticalDataMissing
          expr: |
            increase(clinical_data_missing_total[1h]) > 10
          for: 5m
          labels:
            severity: critical
            sla: data_quality
          annotations:
            summary: "Critical clinical data missing"
            description: "{{ $value }} instances of missing critical data in the last hour"
```

Disaster Recovery Plan

```
# EASY-Diabetes Disaster Recovery Plan

## Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO)

| Service | RTO | RPO | Priority |
|---------|-----|-----|----------|
| Clinical API | 15 min | 5 min | P0 |
| Database (Primary) | 30 min | 1 min | P0 |
| Prediction Service | 1 hour | 1 hour | P1 |
```

| Frontend Application | 15 min | N/A | P0 |
| Monitoring System | 2 hours | 1 hour | P2 |

Backup Strategy

Database Backups
```bash
#!/bin/bash
# Automated backup script (runs every hour)

# Variables
BACKUP_DIR="/backups/postgres"
DB_NAME="easy_diabetes"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
S3_BUCKET="easy-diabetes-backups"

# Create backup
pg_dump -h $DB_HOST -U $DB_USER -d $DB_NAME -F c -b -v -f
"$BACKUP_DIR/backup_$TIMESTAMP.dump"

# Compress backup
gzip "$BACKUP_DIR/backup_$TIMESTAMP.dump"

# Upload to S3
aws s3 cp "$BACKUP_DIR/backup_$TIMESTAMP.dump.gz" "s3://$S3_BUCKET/postgres/"

# Clean up old local backups (keep 7 days)
find $BACKUP_DIR -name "backup_*.dump.gz" -mtime +7 -delete

# Verify backup integrity
pg_restore --list "$BACKUP_DIR/backup_$TIMESTAMP.dump.gz" > /dev/null
if [ $? -eq 0 ]; then
    echo "Backup verified successfully"
else
    echo "Backup verification failed" >&2
    exit 1
fi
```

Application State Backup

```
# Kubernetes CronJob for state backup
apiVersion: batch/v1
kind: CronJob
metadata:
```

```
  name: state-backup
spec:
  schedule: "0 */6 * * *"  # Every 6 hours
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: easy-diabetes/backup-tool:latest
            command:
            - /bin/bash
            - -c
            - |
              # Backup Redis state
              redis-cli --rdb /backup/redis-dump.rdb

              # Backup ML model cache
              tar -czf /backup/model-cache.tar.gz /models/cache/

              # Backup configuration
              kubectl get configmaps -n production -o yaml > /backup/configmaps.yaml
              kubectl get secrets -n production -o yaml > /backup/secrets.yaml

              # Upload to S3
              aws s3 sync /backup/ s3://easy-diabetes-backups/state/$(date +%Y%m%d)/
```

Recovery Procedures

1. Database Recovery

```
# Restore from backup
BACKUP_FILE="backup_20240115_120000.dump.gz"

# Download from S3
aws s3 cp s3://easy-diabetes-backups/postgres/$BACKUP_FILE /tmp/

# Decompress
gunzip /tmp/$BACKUP_FILE

# Restore database
pg_restore -h $DB_HOST -U $DB_USER -d $DB_NAME -v /tmp/${BACKUP_FILE%.gz}
```

```
# Verify restoration
psql -h $DB_HOST -U $DB_USER -d $DB_NAME -c "SELECT COUNT(*) FROM
patient.patients;"
```

2. Application Recovery

```
# Scale down current deployment
kubectl scale deployment clinical-api --replicas=0 -n production

# Update deployment with recovery image
kubectl set image deployment/clinical-api \
  clinical-api=easy-diabetes/clinical-api:recovery-$VERSION \
  -n production

# Scale up with new image
kubectl scale deployment clinical-api --replicas=3 -n production

# Monitor rollout
kubectl rollout status deployment/clinical-api -n production
```

3. Multi-Region Failover

```
# Primary region failure detected
# Switch traffic to secondary region

# Update DNS to point to secondary load balancer
aws route53 change-resource-record-sets \
  --hosted-zone-id $ZONE_ID \
  --change-batch '{
    "Changes": [{
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "api.easy-diabetes.com",
        "Type": "A",
        "AliasTarget": {
          "HostedZoneId": "Z2FDTNDATAQYW2",
          "DNSName": "secondary-lb.us-east-1.elb.amazonaws.com",
          "EvaluateTargetHealth": true
        }
      }
    }]
  }'
```

Verify failover
curl -f https://api.easy-diabetes.com/health

Communication Plan

Incident Response Team

1. **Incident Commander**: Overall coordination
2. **Technical Lead**: Technical recovery execution
3. **Communications Lead**: Stakeholder updates
4. **Customer Success Lead**: User communication

Communication Channels

- **Internal**: Slack #incident-response
- **Status Page**: status.easy-diabetes.com
- **Customer Email**: Via automated system
- **Phone Tree**: For P0 incidents

Status Update Template

Incident Status Update

Time: [TIMESTAMP]
Severity: P0/P1/P2
Impact: [Affected services and users]
Current Status: Investigating/Identified/Monitoring/Resolved
Next Update: [TIME]

Details:
[Brief description of the issue and current actions]

Customer Impact:
[What users are experiencing]

Workaround (if available):
[Temporary solutions for users]

Security Considerations

Security Architecture
```yaml
```

```
# Security policies and configurations
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: production
spec:
  mtls:
    mode: STRICT
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: clinical-api-authz
  namespace: production
spec:
  selector:
    matchLabels:
      app: clinical-api
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/production/sa/frontend"]
    to:
    - operation:
        methods: ["GET", "POST"]
        paths: ["/api/v1/*"]
  - from:
    - source:
        principals: ["cluster.local/ns/production/sa/admin"]
    to:
    - operation:
        methods: ["*"]
---
# Network policies
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: clinical-api-network-policy
  namespace: production
spec:
  podSelector:
    matchLabels:
```

```
    app: clinical-api
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: production
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8000
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: production
    ports:
    - protocol: TCP
      port: 5432  # PostgreSQL
    - protocol: TCP
      port: 6379  # Redis
  - to:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
```

HIPAA Compliance Checklist

```
# HIPAA Compliance Implementation Checklist

## Administrative Safeguards

### Security Officer and Workforce Training
- [x] Designated HIPAA Security Officer
- [x] Workforce training program implemented
- [x] Access management procedures documented
```

- [x] Sanction policy for violations
- [x] Periodic security updates training

Access Management
- [x] Unique user identification
- [x] Automatic logoff (15 min idle)
- [x] Encryption and decryption procedures

Physical Safeguards

Facility Access Controls
- [x] Data center physical security (AWS/Azure compliance)
- [x] Workstation use policies
- [x] Device and media controls

Technical Safeguards

Access Control
```python
# Implementation example
from functools import wraps
from typing import Callable
import audit_log

def hipaa_access_control(resource_type: str, action: str):
    """Decorator for HIPAA-compliant access control."""
    def decorator(func: Callable) -> Callable:
        @wraps(func)
        async def wrapper(*args, **kwargs):
            # Extract user and patient info
            user = kwargs.get('current_user')
            patient_id = kwargs.get('patient_id')

            # Check access permissions
            if not check_access_permission(user, resource_type, action, patient_id):
                # Log access denial
                await audit_log.log_access_denial(
                    user_id=user.id,
                    resource_type=resource_type,
                    action=action,
                    patient_id=patient_id
                )
                raise HTTPException(403, "Access denied")
```

```
            # Log successful access
            await audit_log.log_access(
                user_id=user.id,
                resource_type=resource_type,
                action=action,
                patient_id=patient_id
            )

            # Execute function
            result = await func(*args, **kwargs)

            return result

        return wrapper
    return decorator

# Usage
@hipaa_access_control("patient_record", "read")
async def get_patient_record(patient_id: str, current_user: User):
    # Implementation
    pass
```

Audit Controls

```
-- Audit log table with required HIPAA fields
CREATE TABLE audit.hipaa_audit_log (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    timestamp TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    user_id UUID NOT NULL,
    user_name VARCHAR(255) NOT NULL,
    user_role VARCHAR(50) NOT NULL,
    action VARCHAR(50) NOT NULL, -- CREATE, READ, UPDATE, DELETE
    resource_type VARCHAR(50) NOT NULL,
    resource_id UUID,
    patient_id UUID,
    phi_accessed TEXT[], -- Specific PHI fields accessed
    access_reason VARCHAR(255),
    ip_address INET NOT NULL,
    user_agent TEXT,
    session_id UUID,
    success BOOLEAN NOT NULL DEFAULT true,
    failure_reason TEXT,

    -- Indexes for audit queries
```

```
    INDEX idx_audit_user_time (user_id, timestamp DESC),
    INDEX idx_audit_patient_time (patient_id, timestamp DESC),
    INDEX idx_audit_action_time (action, timestamp DESC)
) PARTITION BY RANGE (timestamp);

-- Create monthly partitions for audit logs
CREATE TABLE audit.hipaa_audit_log_2024_01
PARTITION OF audit.hipaa_audit_log
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');

-- Audit report views
CREATE VIEW audit.user_access_report AS
SELECT
    user_id,
    user_name,
    COUNT(*) as total_accesses,
    COUNT(DISTINCT patient_id) as patients_accessed,
    COUNT(DISTINCT DATE(timestamp)) as days_active,
    array_agg(DISTINCT action) as actions_performed
FROM audit.hipaa_audit_log
WHERE timestamp >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY user_id, user_name;
```

Integrity Controls

```
# Data integrity verification
import hashlib
import hmac
from cryptography.fernet import Fernet

class DataIntegrityService:
    def __init__(self, secret_key: bytes):
        self.secret_key = secret_key
        self.fernet = Fernet(Fernet.generate_key())

    def create_integrity_hash(self, data: dict) -> str:
        """Create HMAC hash for data integrity."""
        # Serialize data consistently
        data_str = json.dumps(data, sort_keys=True)

        # Create HMAC
        h = hmac.new(
            self.secret_key,
            data_str.encode(),
```

```
            hashlib.sha256
        )

        return h.hexdigest()

    def verify_integrity(self, data: dict, expected_hash: str) -> bool:
        """Verify data integrity using HMAC."""
        actual_hash = self.create_integrity_hash(data)
        return hmac.compare_digest(actual_hash, expected_hash)

    def encrypt_phi(self, phi_data: str) -> str:
        """Encrypt PHI data at rest."""
        return self.fernet.encrypt(phi_data.encode()).decode()

    def decrypt_phi(self, encrypted_data: str) -> str:
        """Decrypt PHI data."""
        return self.fernet.decrypt(encrypted_data.encode()).decode()
```

Transmission Security

```
# TLS configuration for all services
from ssl import create_default_context, Purpose
import certifi

def create_secure_ssl_context():
    """Create SSL context with strict security settings."""
    context = create_default_context(
        purpose=Purpose.CLIENT_AUTH,
        cafile=certifi.where()
    )

    # Force TLS 1.2 minimum
    context.minimum_version = ssl.TLSVersion.TLSv1_2

    # Disable weak ciphers
    context.set_ciphers(
        'ECDHE+AESGCM:ECDHE+CHACHA20:DHE+AESGCM:'
        'DHE+CHACHA20:!aNULL:!MD5:!DSS'
    )

    return context
```

Performance Optimization

Database Query Optimization

```
-- Optimized query for patient clinical summary
WITH latest_labs AS (
    SELECT DISTINCT ON (patient_id, lab_type)
        patient_id,
        lab_type,
        value,
        value_numeric,
        unit,
        collected_at
    FROM clinical.lab_results
    WHERE patient_id = ANY($1::uuid[])  -- Array of patient IDs
      AND collected_at >= CURRENT_DATE - INTERVAL '6 months'
    ORDER BY patient_id, lab_type, collected_at DESC
),
active_medications AS (
    SELECT
        pm.patient_id,
        json_agg(
            json_build_object(
                'medication_name', m.generic_name,
                'drug_class', m.drug_class,
                'dose', pm.dose,
                'dose_unit', pm.dose_unit,
                'frequency', pm.frequency,
                'start_date', pm.start_date
            ) ORDER BY pm.start_date DESC
        ) as medications
    FROM clinical.patient_medications pm
    JOIN clinical.medications m ON m.id = pm.medication_id
    WHERE pm.patient_id = ANY($1::uuid[])
      AND pm.is_active = true
    GROUP BY pm.patient_id
),
recent_glucose AS (
    SELECT
        patient_id,
        percentile_cont(0.5) WITHIN GROUP (ORDER BY value) as median_glucose,
        AVG(value) as mean_glucose,
        STDDEV(value) as glucose_variability,
        COUNT(*) as reading_count
```

```
    FROM clinical.measurements
    WHERE patient_id = ANY($1::uuid[])
      AND measurement_type = 'glucose'
      AND measured_at >= CURRENT_DATE - INTERVAL '14 days'
    GROUP BY patient_id
)
SELECT
    p.id,
    p.external_id,
    p.first_name,
    p.last_name,

    -- Latest labs as JSON
    coalesce(
        json_object_agg(
            ll.lab_type,
            json_build_object(
                'value', ll.value,
                'numeric', ll.value_numeric,
                'unit', ll.unit,
                'date', ll.collected_at
            )
        ) FILTER (WHERE ll.lab_type IS NOT NULL),
        '{}'::json
    ) as latest_labs,

    -- Medications
    am.medications,

    -- Recent glucose stats
    json_build_object(
        'median', rg.median_glucose,
        'mean', rg.mean_glucose,
        'variability', rg.glucose_variability,
        'reading_count', rg.reading_count
    ) as glucose_stats

FROM patient.patients p
LEFT JOIN latest_labs ll ON ll.patient_id = p.id
LEFT JOIN active_medications am ON am.patient_id = p.id
LEFT JOIN recent_glucose rg ON rg.patient_id = p.id
WHERE p.id = ANY($1::uuid[])
GROUP BY p.id, p.external_id, p.first_name, p.last_name,
         am.medications, rg.median_glucose, rg.mean_glucose,
```

```
        rg.glucose_variability, rg.reading_count;

-- Create covering index for this query
CREATE INDEX idx_clinical_summary ON clinical.lab_results
(patient_id, lab_type, collected_at DESC)
INCLUDE (value, value_numeric, unit)
WHERE collected_at >= CURRENT_DATE - INTERVAL '6 months';
```

Caching Strategy

```
# Multi-level caching implementation
from typing import Optional, Any, Callable
import asyncio
from functools import wraps
import redis.asyncio as redis
import pickle
import hashlib

class MultiLevelCache:
    def __init__(self):
        self.memory_cache = {}  # L1: In-memory cache
        self.redis_client = None  # L2: Redis cache
        self.cache_ttl = {
            'short': 300,      # 5 minutes
            'medium': 3600,    # 1 hour
            'long': 86400,     # 24 hours
        }

    async def initialize(self, redis_url: str):
        """Initialize Redis connection."""
        self.redis_client = await redis.from_url(redis_url)

    def cache_key(self, prefix: str, *args, **kwargs) -> str:
        """Generate consistent cache key."""
        key_data = f"{prefix}:{args}:{sorted(kwargs.items())}"
        return hashlib.md5(key_data.encode()).hexdigest()

    async def get(self, key: str) -> Optional[Any]:
        """Get from cache (memory first, then Redis)."""
        # Check L1 cache
        if key in self.memory_cache:
            return self.memory_cache[key]['value']

        # Check L2 cache
```

```
    if self.redis_client:
        cached = await self.redis_client.get(key)
        if cached:
            value = pickle.loads(cached)
            # Promote to L1
            self.memory_cache[key] = {'value': value}
            return value

    return None

async def set(self, key: str, value: Any, ttl: str = 'medium'):
    """Set in both cache levels."""
    # Set in L1
    self.memory_cache[key] = {'value': value}

    # Set in L2
    if self.redis_client:
        await self.redis_client.setex(
            key,
            self.cache_ttl[ttl],
            pickle.dumps(value)
        )

def cached(self, prefix: str, ttl: str = 'medium'):
    """Decorator for caching function results."""
    def decorator(func: Callable) -> Callable:
        @wraps(func)
        async def wrapper(*args, **kwargs):
            # Generate cache key
            cache_key = self.cache_key(prefix, *args, **kwargs)

            # Check cache
            cached_value = await self.get(cache_key)
            if cached_value is not None:
                return cached_value

            # Call function
            result = await func(*args, **kwargs)

            # Cache result
            await self.set(cache_key, result, ttl)

            return result
```

```
            return wrapper
        return decorator

# Usage example
cache = MultiLevelCache()

@cache.cached("patient_summary", ttl="short")
async def get_patient_summary(patient_id: str) -> dict:
    # Expensive operation
    return await fetch_patient_data(patient_id)
```

Performance Monitoring

```
# Application performance monitoring
from prometheus_client import Counter, Histogram, Gauge
import time
from contextlib import asynccontextmanager
import structlog

logger = structlog.get_logger()

# Metrics
request_count = Counter(
    'http_requests_total',
    'Total HTTP requests',
    ['method', 'endpoint', 'status']
)

request_duration = Histogram(
    'http_request_duration_seconds',
    'HTTP request duration',
    ['method', 'endpoint']
)

active_requests = Gauge(
    'http_requests_active',
    'Active HTTP requests'
)

db_query_duration = Histogram(
    'db_query_duration_seconds',
    'Database query duration',
    ['query_type', 'table']
)
```

```
cache_operations = Counter(
    'cache_operations_total',
    'Cache operations',
    ['operation', 'result']
)

@asynccontextmanager
async def track_performance(operation: str, **tags):
    """Context manager for tracking operation performance."""
    start_time = time.time()

    # Log start
    logger.info(f"{operation}_started", **tags)

    try:
        yield

        # Record success
        duration = time.time() - start_time
        logger.info(
            f"{operation}_completed",
            duration=duration,
            **tags
        )

    except Exception as e:
        # Record failure
        duration = time.time() - start_time
        logger.error(
            f"{operation}_failed",
            duration=duration,
            error=str(e),
            **tags
        )
        raise

# Middleware for request tracking
async def performance_middleware(request, call_next):
    """Track request performance."""
    active_requests.inc()
    start_time = time.time()

    try:
```

```
        response = await call_next(request)
        duration = time.time() - start_time

        # Record metrics
        request_count.labels(
            method=request.method,
            endpoint=request.url.path,
            status=response.status_code
        ).inc()

        request_duration.labels(
            method=request.method,
            endpoint=request.url.path
        ).observe(duration)

        # Add performance headers
        response.headers["X-Response-Time"] = f"{duration:.3f}"

        return response

    finally:
        active_requests.dec()
```

Conclusion

This comprehensive documentation suite provides the foundation for developing EASY-Diabetes, a state-of-the-art clinical decision support system. The documentation covers:

1. **System Architecture**: Microservices design with scalable, fault-tolerant infrastructure
2. **Medical Knowledge Base**: Comprehensive clinical guidelines and evidence-based protocols
3. **Technical Implementation**: Detailed code examples and best practices
4. **Security and Compliance**: HIPAA-compliant security measures and audit trails
5. **Testing Strategy**: Unit, integration, and end-to-end testing approaches
6. **Deployment and Operations**: CI/CD pipelines and monitoring strategies
7. **Performance Optimization**: Caching, query optimization, and monitoring

The system is designed to handle millions of users while maintaining sub-second response times and 99.9% availability. The modular architecture allows for easy extension to other chronic conditions beyond diabetes.

Document Version: 2.0.0
Last Updated: January 2025
Next Review: January 2026
Maintainers: EASY-Diabetes Development Team