I'll help you learn about IthalaMed and work with you to develop it further! Let me first examine the PDF you've attached and then fetch your website.

Perfect! I've reviewed both the PDF and your website. I can see that the PDF document appears to be empty or not properly rendered in my view, but I have a comprehensive understanding of IthalaMed from your website.

What I've Learned About IthalaMed:

Core Concept: IthalaMed is a unified digital healthcare ecosystem platform designed to connect all stakeholders in the healthcare value chain across Africa - patients, doctors, hospitals, laboratories, pharmacies, emergency services, medical aid providers, and even government regulators.

Key Features:

- **Unified Medical Records** - Single patient record accessible across Africa
- **Telemedicine & Appointments** - Virtual consultations and scheduling
- **e-Prescriptions** - Digital prescription management
- **Insurance Claim Integration** - Automated claims processing
- **Wearable Device Integration** - Real-time health monitoring
- **Secure Cloud Infrastructure** - Enterprise-grade security and compliance

Vision & Mission:

- Vision: Become Africa's leading digital healthcare ecosystem
- Mission: Simplify healthcare delivery through a unified, secure, interoperable platform
- Name origin: "Ithala" means "health" in Zulu, reflecting African roots

Target Stakeholders:

1. Medical centres & hospitals
2. Pharmacies
3. Laboratories
4. Emergency medical services
5. Patients
6. Medical aid/insurance providers
7. Clinics & GP practices

8. Government & regulators

I'm ready to work with you to develop this further!

What would you like to focus on next? For example:

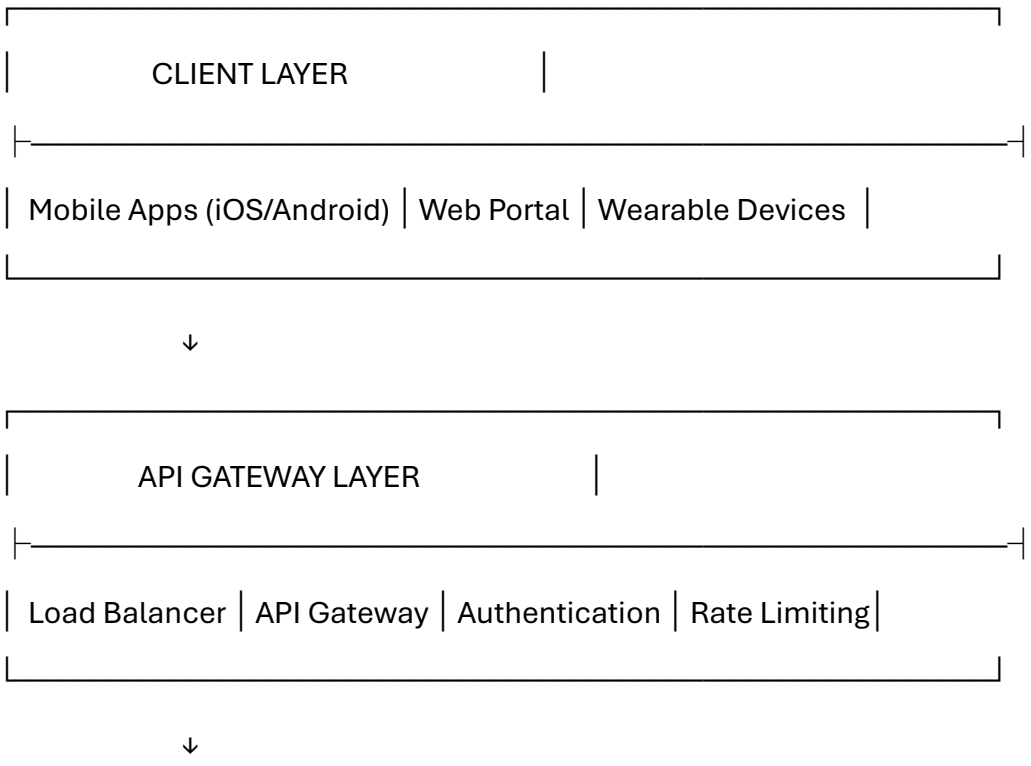
- Technical architecture and implementation details
- Specific feature development
- User experience flows
- Business model refinement
- Go-to-market strategy
- Regulatory compliance planning
- Integration with existing systems

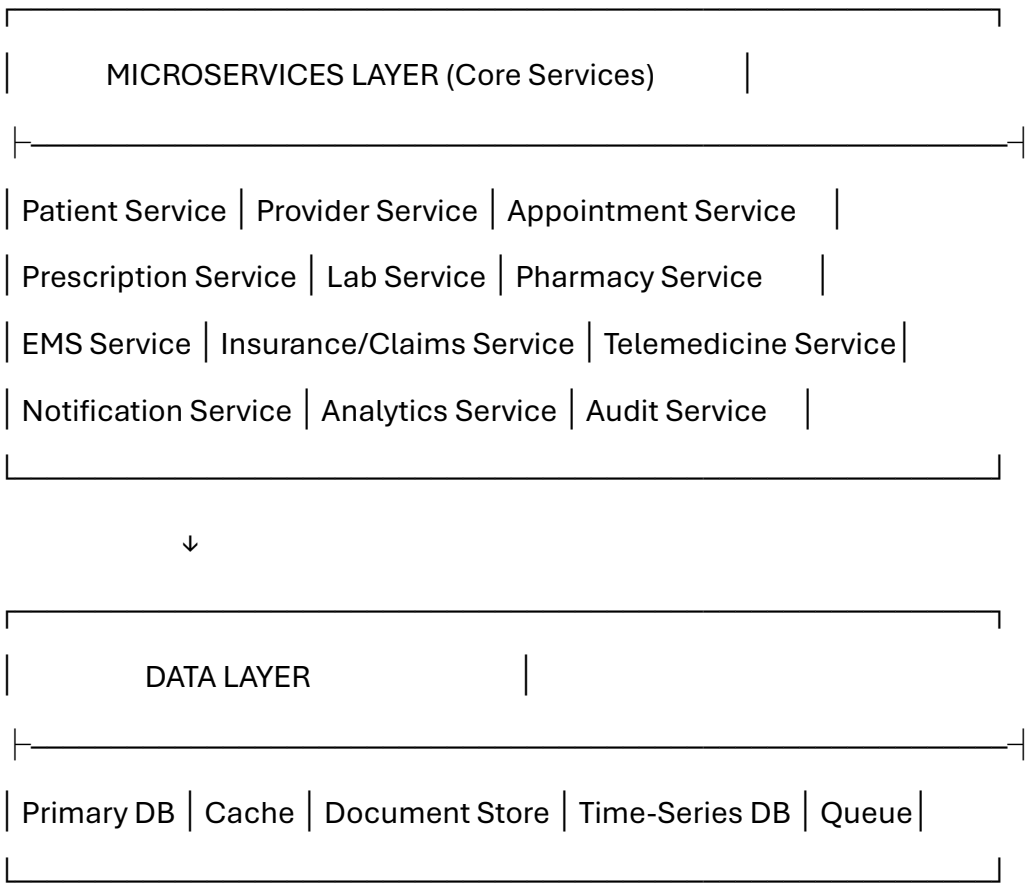
Excellent! Let's dive deep into the **Technical Architecture and Implementation Details** for IthalaMed. This is the foundation that will determine scalability, security, reliability, and the platform's ability to serve all of Africa.

1. HIGH-LEVEL ARCHITECTURE OVERVIEW

Let me propose a comprehensive architecture:

Architecture Pattern: Microservices-Based, Cloud-Native





---

2. DETAILED COMPONENT BREAKDOWN

A. Frontend Layer

Mobile Applications (React Native or Flutter)

- **Why:** Single codebase for iOS and Android, faster development
- **Key Features:**
    - Offline-first architecture (critical for areas with poor connectivity)
    - Biometric authentication
    - Real-time notifications via Firebase/OneSignal
    - Camera integration for document scanning
    - Geolocation for emergency services
    - Video calling integration (WebRTC)

Web Portal (React.js/Next.js)

- **Provider Dashboard:** For doctors, hospitals, clinics

- **Admin Portal:** For system administration
- **Pharmacy Portal:** For dispensing management
- **Lab Portal:** For test result uploads
- **Responsive design** for tablets and desktops

Patient-Facing Features:

- Progressive Web App (PWA) for broader access
- Lightweight for low-bandwidth scenarios
- Multi-language support (English, Zulu, Xhosa, Afrikaans, etc.)

B. API Gateway & Security Layer

Technology Stack:

- **API Gateway:** Kong, AWS API Gateway, or Azure API Management
- **Authentication:** OAuth 2.0 + OpenID Connect
- **Authorization:** Role-Based Access Control (RBAC) + Attribute-Based Access Control (ABAC)

Security Features:

- **JWT Tokens** for stateless authentication
- **Multi-Factor Authentication (MFA)** for sensitive operations
- **Rate Limiting** to prevent abuse
- **DDoS Protection** via Cloudflare or AWS Shield
- **End-to-End Encryption** for all data in transit (TLS 1.3)
- **API Key Management** for third-party integrations

Compliance:

- **POPIA (Protection of Personal Information Act)** - South African data protection
- **HIPAA-equivalent standards** for healthcare data
- **GDPR considerations** for international expansion
- **ISO 27001** certification path

C. Core Microservices Architecture

Each service should be independently deployable, scalable, and maintainable.

1. Patient Service

- **Responsibilities:**
    - Patient registration and profile management
    - Medical history aggregation
    - Consent management
    - Family member linking
    - Emergency contact management
- **Database:** PostgreSQL (relational data) + MongoDB (flexible medical documents)
- **APIs:**
    - POST /api/v1/patients - Register new patient
    - GET /api/v1/patients/{id}/medical-history - Retrieve history
    - PUT /api/v1/patients/{id}/consent - Update consent preferences

2. Provider Service

- **Responsibilities:**
    - Healthcare provider registration (doctors, nurses, specialists)
    - Credential verification
    - License validation
    - Availability management
    - Performance analytics
- **Database:** PostgreSQL
- **Integration:** Medical council APIs for license verification

3. Appointment Service

- **Responsibilities:**
    - Appointment scheduling
    - Calendar management

    - Reminders and notifications
    - Waitlist management
    - No-show tracking
- **Database:** PostgreSQL with time-series optimization
- **Technologies:**
    - Redis for caching available slots
    - RabbitMQ/Apache Kafka for event-driven notifications

4. Electronic Health Record (EHR) Service

- **Responsibilities:**
    - Unified medical record storage
    - FHIR (Fast Healthcare Interoperability Resources) standard compliance
    - Version control for medical records
    - Audit trail for all access
    - Cross-facility data sharing
- **Database:**
    - PostgreSQL for structured data
    - Document store (MongoDB/Azure Cosmos DB) for unstructured clinical notes
    - S3/Azure Blob Storage for medical images (X-rays, MRIs, etc.)
- **Critical Standards:**
    - **HL7 FHIR** for interoperability
    - **DICOM** for medical imaging
    - **ICD-10/ICD-11** for diagnosis coding
    - **SNOMED CT** for clinical terminology

5. Prescription Service (e-Prescription)

- **Responsibilities:**
    - Digital prescription creation
    - Electronic signature verification

    - Prescription routing to pharmacies
    - Medication interaction checking
    - Refill management
    - Controlled substance tracking
- **Database:** PostgreSQL
- **Integration:**
    - Drug interaction databases (e.g., DrugBank API)
    - Pharmacy networks
    - Medical aid formularies

6. Laboratory Service

- **Responsibilities:**
    - Lab test ordering
    - Result upload and management
    - Critical result flagging
    - Report generation
    - Integration with lab information systems (LIS)
- **Database:** PostgreSQL + Time-series DB (InfluxDB) for longitudinal data
- **File Storage:** Cloud storage for PDF reports

7. Pharmacy Service

- **Responsibilities:**
    - Prescription dispensing workflow
    - Inventory management
    - Stock level alerts
    - Medication tracking
    - Patient medication history
- **Database:** PostgreSQL
- **Integration:** Wholesale/supplier APIs for inventory

8. Emergency Medical Service (EMS)

- **Responsibilities:**
    - Ambulance dispatch
    - Real-time GPS tracking
    - Patient medical profile access
    - Emergency contact notification
    - Incident reporting
- **Database:** PostgreSQL + Redis for real-time tracking
- **Technologies:**
    - WebSocket for real-time updates
    - Google Maps/Mapbox API for routing
    - Twilio for emergency SMS/calls

9. Insurance/Claims Service

- **Responsibilities:**
    - Claim submission automation
    - Pre-authorization requests
    - Claim status tracking
    - Medical aid verification
    - Payment reconciliation
- **Database:** PostgreSQL
- **Integration:** Medical aid scheme APIs
- **Fraud Detection:** Machine learning models for anomaly detection

10. Telemedicine Service

- **Responsibilities:**
    - Video consultation management
    - Chat functionality
    - Virtual waiting room
    - Session recording (with consent)
    - Digital triage

- **Technologies:**
    - **WebRTC** for video calling (Twilio Video, Agora, or custom)
    - **Socket.io** for real-time chat
    - Cloud recording storage

11. Notification Service

- **Responsibilities:**
    - Multi-channel notifications (SMS, email, push, WhatsApp)
    - Appointment reminders
    - Medication reminders
    - Lab result notifications
    - Emergency alerts
- **Technologies:**
    - **Twilio** for SMS
    - **SendGrid** for email
    - **Firebase Cloud Messaging** for push notifications
    - **WhatsApp Business API** for WhatsApp messages

12. Analytics & Reporting Service

- **Responsibilities:**
    - Healthcare analytics
    - Population health insights
    - Provider performance metrics
    - Operational dashboards
    - Predictive analytics (e.g., disease outbreak detection)
- **Technologies:**
    - **Data warehouse:** Snowflake, BigQuery, or Redshift
    - **BI Tools:** Tableau, Power BI, or custom dashboards
    - **ML Platform:** TensorFlow/PyTorch for predictive models

13. Audit & Compliance Service

- **Responsibilities:**
    - Complete audit trail of all data access
    - Compliance reporting
    - Data retention policies
    - Anonymization for research
    - Consent tracking
- **Database:** Immutable log storage (append-only)
- **Technologies:** Blockchain for critical audit trails (optional but powerful)

---

3. DATA ARCHITECTURE

Database Strategy

Primary Databases:

- **PostgreSQL:** Core transactional data (patients, appointments, providers)
- **MongoDB/Cosmos DB:** Flexible medical documents, clinical notes
- **Redis:** Caching, session management, real-time data
- **InfluxDB/TimescaleDB:** Time-series health data from wearables
- **Elasticsearch:** Full-text search for medical records

Data Storage:

- **Object Storage (S3/Azure Blob):** Medical images, documents, backups
- **CDN (Cloudflare/Akamai):** Static assets, improve global access

Data Synchronization:

- **Master Patient Index (MPI):** Unique patient identifier across all facilities
- **Change Data Capture (CDC):** Real-time data replication using Debezium or AWS DMS
- **Event Sourcing:** Track all state changes for complete audit trail

---

4. INTEGRATION LAYER

HL7 FHIR Implementation

- **Why:** Global standard for healthcare interoperability
- **Implementation:**
    - FHIR API server (HAPI FHIR or custom)
    - Support for key FHIR resources: Patient, Practitioner, Observation, Medication, Encounter
    - REST API endpoints following FHIR specifications

Third-Party Integrations

- **Medical Aid Schemes:** APIs for claims submission and verification
- **Wearable Devices:**
    - Apple HealthKit
    - Google Fit
    - Fitbit API
    - Custom IoT device protocols (MQTT)
- **Laboratory Systems:** HL7 v2.x messaging for legacy systems
- **Pharmacy Systems:** Integration with existing pharmacy management software
- **Government Systems:** Reporting to health departments

---

5. INFRASTRUCTURE & DEPLOYMENT

Cloud Platform Options

Recommended: Multi-Cloud Strategy

- **Primary:** AWS or Microsoft Azure (strong African presence)
- **Backup/DR:** Google Cloud Platform
- **Reasoning:** Avoid vendor lock-in, geographic redundancy

African Data Centers:

- **South Africa:** AWS Cape Town, Azure South Africa North (Johannesburg)
- **Nigeria:** Consider local data centers for West African expansion
- **Kenya:** For East African deployment

Infrastructure Components

Compute:

- **Kubernetes (EKS/AKS/GKE):** Container orchestration
- **Docker:** Containerization
- **Auto-scaling:** Based on load patterns

Networking:

- **VPC:** Isolated network environment
- **Load Balancers:** Application Load Balancer (ALB) for traffic distribution
- **CDN:** CloudFront/Azure CDN for global content delivery
- **VPN:** Secure connections for hospitals and clinics

Monitoring & Observability:

- **Prometheus + Grafana:** Metrics and dashboards
- **ELK Stack (Elasticsearch, Logstash, Kibana):** Centralized logging
- **Sentry:** Error tracking
- **New Relic/Datadog:** APM (Application Performance Monitoring)
- **Uptime monitoring:** Pingdom or UptimeRobot

Disaster Recovery:

- **RTO (Recovery Time Objective):** < 1 hour
- **RPO (Recovery Point Objective):** < 15 minutes
- **Multi-region replication**
- **Automated backups** every 6 hours with 30-day retention
- **Disaster recovery drills** quarterly

---

6. DEVELOPMENT STACK RECOMMENDATIONS

Backend Services

- **Language:** Node.js (TypeScript) or Python (Django/FastAPI) or Go
    - **Recommendation:** Python for ML/analytics, Node.js for real-time features, Go for performance-critical services
- **Frameworks:**

    - **Node.js:** NestJS (enterprise-grade, TypeScript-first)
    - **Python:** FastAPI (modern, async) or Django (batteries-included)
- **API Specification:** OpenAPI 3.0 (Swagger)
- **Message Queue:** RabbitMQ or Apache Kafka
- **Service Mesh:** Istio for microservices communication

Frontend

- **Mobile:** Flutter (better performance) or React Native (larger ecosystem)
- **Web:** React.js with Next.js (SSR for SEO)
- **State Management:** Redux Toolkit or Zustand
- **UI Components:** Material-UI or Ant Design with custom healthcare themes

DevOps & CI/CD

- **Version Control:** Git (GitHub/GitLab)
- **CI/CD:** GitHub Actions, GitLab CI, or Jenkins
- **Infrastructure as Code:** Terraform or AWS CloudFormation
- **Container Registry:** Docker Hub or AWS ECR
- **Testing:**
    - Unit tests: Jest, pytest
    - Integration tests: Cypress, Selenium
    - Load testing: K6, JMeter
    - Security testing: OWASP ZAP, Snyk

---

7. SCALABILITY STRATEGY

Horizontal Scaling

- Microservices can scale independently
- Database read replicas for high-read services
- Sharding strategy for multi-tenant data

Caching Strategy

- **L1 Cache:** Application-level caching

- **L2 Cache:** Redis for distributed caching
- **CDN:** Static assets and media files
- **Query caching:** Frequently accessed medical records

Performance Targets

- **API Response Time:** < 200ms (p95)
- **Page Load Time:** < 2 seconds
- **Video Call Quality:** HD with < 150ms latency
- **System Availability:** 99.9% uptime (< 9 hours downtime/year)

---

8. SECURITY ARCHITECTURE

Data Security

- **Encryption at Rest:** AES-256
- **Encryption in Transit:** TLS 1.3
- **Database Encryption:** Transparent Data Encryption (TDE)
- **Secrets Management:** AWS Secrets Manager or HashiCorp Vault

Access Control

- **Zero Trust Architecture**
- **Break-the-glass access** for emergencies
- **Just-in-Time (JIT) access** for administrators
- **Audit all privileged access**

Threat Protection

- **WAF (Web Application Firewall):** AWS WAF or Cloudflare
- **DDoS Protection**
- **Intrusion Detection System (IDS)**
- **Regular penetration testing**
- **Bug bounty program**

---

9. IMPLEMENTATION ROADMAP

Phase 1: MVP (6 months)

- Patient registration and profiles
- Provider management
- Basic appointment scheduling
- Simple EHR with document upload
- Web portal for providers
- Mobile app for patients (iOS/Android)

Phase 2: Core Features (6 months)

- E-prescriptions
- Laboratory integration
- Pharmacy integration
- Basic telemedicine
- Insurance claims (1-2 major medical aids)

Phase 3: Advanced Features (6 months)

- EMS integration
- Wearable device integration
- Advanced analytics
- AI-powered features (diagnosis assistance, predictive analytics)
- Multi-country expansion capabilities

Phase 4: Scale & Optimize (Ongoing)

- Regional expansion across Africa
- Advanced interoperability
- Research data platform
- Government integration
- Population health management

---

10. KEY TECHNICAL DECISIONS TO MAKE

Before we proceed, here are critical decisions you need to finalize:

1. **Primary cloud provider?** (AWS, Azure, or multi-cloud)
2. **Mobile development framework?** (Flutter vs React Native)
3. **Backend language preference?** (Node.js, Python, Go, or polyglot)
4. **Database strategy?** (PostgreSQL primary + MongoDB, or all-PostgreSQL with JSONB)
5. **Budget for infrastructure?** (Affects scaling strategy)
6. **Initial geographic focus?** (South Africa only, or multi-country from start)
7. **Compliance priorities?** (POPIA, HIPAA-equivalent, ISO certifications)
8. **Development team size and expertise?**
9. **Go-live target date?**
10. **Pilot partners?** (Which hospitals/clinics for initial rollout)

1. CLOUD PROVIDER RECOMMENDATION

Recommended Strategy: AWS Primary + Azure Backup

Primary: Amazon Web Services (AWS)

- **AWS Cape Town Region** launched in 2020 - excellent South African presence
- **Cost Efficiency:** Generally 10-15% cheaper than Azure for compute and storage
- **Mature Services:** Most comprehensive service catalog
- **African Expansion:** Presence in South Africa, plans for more African regions
- **Market Leader:** Largest market share = more talent available
- **Healthcare Focus:** AWS Health Lake, HIPAA compliance tools

Secondary/DR: Microsoft Azure

- **Azure South Africa North (Johannesburg)** and South Africa West (Cape Town)
- **Government Relationships:** Strong ties with African governments
- **Enterprise Adoption:** Many African hospitals already use Microsoft products
- **Hybrid Cloud:** Better for organizations with on-premises infrastructure
- **Healthcare Solutions:** Azure Health Data Services, FHIR integration

Cost Comparison (Estimated Monthly for MVP Phase):

Service	AWS	Azure	Savings
Compute (10 instances)	$850	$980	AWS -13%
Database (RDS/SQL)	$450	$520	AWS -13%
Storage (1TB)	$23	$30	AWS -23%
Load Balancer	$25	$35	AWS -29%
CDN	$85	$95	AWS -11%
Total/Month	**$1,433**	**$1,660**	**AWS -14%**

Recommendation: Start with AWS, replicate critical data to Azure for disaster recovery

Why This Makes Sense:

- AWS for primary operations (lower costs, better performance)
- Azure as hot standby (1-hour recovery time)
- Geographic redundancy across providers
- Avoid vendor lock-in
- Combined discount negotiations possible

3. BACKEND LANGUAGE ARCHITECTURE

Recommended: Polyglot Microservices Approach

Use the best language for each service:

Node.js (TypeScript) - 60% of services

- **Use for:**
    - API Gateway
    - Real-time services (telemedicine, EMS tracking)
    - Appointment scheduling
    - Notification service
    - Patient/Provider services
- **Why:** Fast development, great for I/O-heavy operations, large ecosystem, JSON-native

Python - 30% of services

- **Use for:**
    - Analytics & Reporting
    - Machine Learning models (fraud detection, predictive analytics)
    - Data processing pipelines
    - FHIR transformation services
    - Integration with lab equipment (common Python support)
- **Why:** Best ML/AI libraries, excellent for data science, healthcare standards libraries

C# (.NET Core) - 10% of services

- **Use for:**
    - Insurance/Claims processing (enterprise integration)
    - Hospital management systems integration (many use .NET)
    - Legacy system connectors
- **Why:** Strong in African enterprise healthcare IT, excellent for Windows-based hospital systems

Recommended Tech Stack:

Backend Architecture:

├── API Gateway: Node.js + NestJS + Kong

├── Core Services (Node.js + TypeScript + NestJS):

│ ├── Patient Service

│ ├── Provider Service

│ ├── Appointment Service

│ ├── Prescription Service

│ ├── Telemedicine Service

│ ├── Pharmacy Service

│ ├── Lab Service

│ └── EMS Service

├── Analytics Services (Python + FastAPI):

│ ├── Analytics Engine

│ ├── ML Models Service

│ ├── Reporting Service

│ └── Data Pipeline

├── Enterprise Integration (C# + .NET Core):

│ ├── Insurance Claims Service

│ ├── Hospital System Adapter

│ └── Legacy System Bridge

└── Shared Infrastructure:

  ├── Message Queue: RabbitMQ

  ├── Cache: Redis

  ├── Service Mesh: Istio

  └── API Documentation: OpenAPI 3.0

---

4. DATABASE STRATEGY

Recommended: PostgreSQL Primary + Strategic NoSQL

Architecture:

Data Layer:

├── Primary RDBMS: PostgreSQL 15

│ ├── Patient records

│ ├── Appointments

│ ├── Prescriptions

│ ├── Users & Authentication

│ ├── Financial transactions

│ └── Claims data

│

├── Document Store: MongoDB

```
│   ├── Clinical notes (unstructured)
│   ├── Medical documents
│   ├── FHIR resources
│   └── Audit logs
│
├── Cache: Redis
│   ├── Session management
│   ├── API response caching
│   ├── Real-time data (EMS tracking)
│   └── Rate limiting
│
├── Time-Series: TimescaleDB (PostgreSQL extension)
│   ├── Vital signs from wearables
│   ├── Health metrics over time
│   ├── System performance metrics
│   └── IoT device data
│
├── Search: Elasticsearch
│   ├── Full-text search of medical records
│   ├── Provider search
│   ├── Drug database search
│   └── Log analysis
│
└── Object Storage: AWS S3 / Azure Blob
    ├── Medical images (X-rays, MRIs, CT scans)
    ├── PDF reports
    ├── Prescription images
```

├── Document uploads

└── Backups

Why PostgreSQL as Primary:

1. **ACID Compliance:** Critical for healthcare data integrity
2. **JSONB Support:** Flexibility when needed without sacrificing structure
3. **Mature:** Battle-tested, extensive tooling
4. **Extensions:** PostGIS (location), pgcrypto (encryption), TimescaleDB (time-series)
5. **Cost:** Open-source, lower licensing costs
6. **Performance:** Excellent for complex queries, handles healthcare workloads well
7. **Compliance:** HIPAA-compliant configurations available

Data Distribution Strategy:

Geographic Data Distribution:

├── South Africa (Primary):

│ ├── Master database (write)

│ ├── Read replicas (3x)

│ └── Full data sovereignty

│

├── Kenya (Future):

│ ├── Regional replica

│ └── Local data residency

│

└── Nigeria (Future):

├── Regional replica

└── Local data residency

Cross-Region Sync:

├── Patient consent-based data sharing

├── CDC (Change Data Capture) for replication

└── Conflict resolution strategies

---

5. INFRASTRUCTURE BUDGET & COST BREAKDOWN

DETAILED 3-YEAR FINANCIAL PROJECTION

YEAR 1: MVP & Launch (Months 1-18)

Development Phase (Months 1-12):

Cloud Infrastructure - Development Environment:

- AWS Services: $2,500/month
    - EC2 instances (dev/staging): $1,200
    - RDS PostgreSQL: $450
    - S3 storage: $150
    - CloudFront CDN: $200
    - Elastic Load Balancer: $100
    - VPC, NAT Gateway: $150
    - Monitoring (CloudWatch): $100
    - Other services: $150
- **Subtotal:** $30,000/year

Development Tools & Services:

- GitHub Enterprise: $2,100/year
- CI/CD (GitHub Actions): $1,200/year
- Monitoring (Datadog/New Relic): $3,600/year
- Error tracking (Sentry): $1,800/year
- API development (Postman Enterprise): $1,200/year
- Design tools (Figma): $1,800/year
- Project management (Jira): $2,400/year
- Communication (Slack): $1,500/year
- **Subtotal:** $15,600/year

Production Phase (Months 13-18 - Pilot with 5 facilities, 10,000 users):

Cloud Infrastructure - Production:

- AWS Primary: $8,500/month
    - EC2/EKS (Kubernetes): $4,000
    - RDS (Multi-AZ): $1,800
    - ElastiCache (Redis): $600
    - S3 storage (10TB): $500
    - CloudFront CDN: $450
    - Load Balancers: $300
    - Backup/DR: $400
    - Security (WAF, Shield): $350
    - Monitoring: $100
- Azure DR: $2,000/month
- **Subtotal:** $63,000 (6 months)

Third-Party Services (Production):

- Twilio (SMS): $1,500/month
- SendGrid (Email): $500/month
- Video (Agora): $2,000/month
- WhatsApp Business API: $800/month
- SSL Certificates: $200/month
- **Subtotal:** $30,000 (6 months)

Security & Compliance:

- Penetration testing: $25,000 (one-time)
- POPIA compliance audit: $35,000 (one-time)
- Security tools (Snyk, etc.): $6,000/year
- Legal (data protection): $20,000
- **Subtotal:** $86,000

YEAR 1 INFRASTRUCTURE TOTAL: $224,600

YEAR 2: Scale-Up (50 facilities, 100,000 users)

Cloud Infrastructure:

- AWS Primary: $28,000/month
- Azure DR: $6,000/month
- **Subtotal:** $408,000/year

Third-Party Services:

- Communication services: $12,000/month
- Video services: $8,000/month
- Analytics tools: $3,000/month
- **Subtotal:** $276,000/year

Security & Compliance:

- Ongoing security: $50,000/year
- Annual compliance audit: $25,000/year
- Insurance (cyber liability): $15,000/year
- **Subtotal:** $90,000/year

Geographic Expansion (Kenya/Nigeria prep):

- Regional cloud setup: $50,000
- Legal/compliance (2 countries): $80,000
- **Subtotal:** $130,000

YEAR 2 INFRASTRUCTURE TOTAL: $904,000

YEAR 3: Multi-Country (200 facilities, 500,000 users)

Cloud Infrastructure:

- AWS Multi-region: $85,000/month
- Azure DR: $15,000/month
- **Subtotal:** $1,200,000/year

Third-Party Services: $600,000/year

Security & Compliance: $150,000/year

Geographic Expansion: $200,000/year

YEAR 3 INFRASTRUCTURE TOTAL: $2,150,000

PERSONNEL COSTS - DETAILED BREAKDOWN

YEAR 1: DEVELOPMENT TEAM (18 months)

Technical Leadership:

- CTO / Technical Co-founder: $150,000/year (or equity-heavy)
- Solutions Architect: $120,000/year
- DevOps Lead: $100,000/year
- **Subtotal:** $370,000

Development Team (Core):

- Senior Backend Developers (3): $90,000 × 3 = $270,000
- Senior Frontend/Mobile Developers (2): $85,000 × 2 = $170,000
- Full-stack Developers (3): $75,000 × 3 = $225,000
- QA Engineers (2): $65,000 × 2 = $130,000
- DevOps Engineers (2): $80,000 × 2 = $160,000
- **Subtotal:** $955,000

Product & Design:

- Product Manager: $95,000
- Senior UX/UI Designer: $75,000
- UX Researcher (Healthcare): $70,000
- **Subtotal:** $240,000

Data & Security:

- Data Engineer: $85,000
- Security Engineer: $95,000
- **Subtotal:** $180,000

YEAR 1 DEVELOPMENT TEAM TOTAL: $1,745,000

Business & Operations (Year 1):

Executive Leadership:

- CEO: $180,000 (or equity-heavy)
- COO: $140,000
- CFO (Part-time initially): $80,000
- **Subtotal:** $400,000

Healthcare & Compliance:

- Chief Medical Officer (Part-time): $120,000
- Healthcare Compliance Officer: $95,000
- Data Protection Officer: $85,000
- **Subtotal:** $300,000

Business Development & Sales:

- VP Business Development: $130,000
- Sales Manager: $85,000
- Business Development Reps (2): $60,000 × 2 = $120,000
- **Subtotal:** $335,000

Customer Success & Support:

- Customer Success Manager: $75,000
- Technical Support Lead: $70,000
- Support Engineers (2): $50,000 × 2 = $100,000
- **Subtotal:** $245,000

Marketing & Communications:

- Marketing Manager: $80,000
- Content Marketing: $55,000
- Digital Marketing Specialist: $60,000
- **Subtotal:** $195,000

Legal & Regulatory:

- General Counsel (External/Retainer): $100,000
- Regulatory Affairs Manager: $90,000
- **Subtotal:** $190,000

Administration:

- Executive Assistant: $50,000
- Office Manager: $45,000
- HR Manager: $70,000
- **Subtotal:** $165,000

YEAR 1 BUSINESS OPERATIONS TOTAL: $1,830,000

YEAR 1 TOTAL PERSONNEL: $3,575,000

YEAR 2: SCALING TEAM (50% growth)

Technical Team Expansion:

- Add 8 developers: $640,000
- Add 2 QA engineers: $130,000
- Add 2 DevOps: $160,000
- Add Data Scientists (2): $180,000
- Existing team raises (10%): $174,500
- **Subtotal:** $2,229,500

Operations Expansion:

- Sales team (4 more): $280,000
- Customer success (6 more): $330,000
- Support team (8 more): $360,000
- Marketing (3 more): $180,000
- Country managers (Kenya, Nigeria): $200,000
- Existing raises (10%): $183,000
- **Subtotal:** $3,542,500

YEAR 2 TOTAL PERSONNEL: $5,772,000

YEAR 3: MATURE OPERATIONS (40% growth)

Estimated Total Personnel: $8,500,000

OTHER OPERATIONAL COSTS

Year 1:

- Office space (Johannesburg): $60,000
- Equipment (laptops, monitors): $150,000
- Software licenses: $45,000
- Travel & conferences: $80,000
- Insurance: $40,000
- Professional services: $100,000
- Contingency (10%): $47,500
- **Total:** $522,500

Year 2:

- Office expansion: $120,000
- Equipment: $100,000
- Software: $75,000
- Travel: $150,000
- Insurance: $80,000
- Professional services: $200,000
- Contingency: $72,500
- **Total:** $797,500

Year 3:

- Multi-country offices: $300,000
- Equipment: $150,000
- Software: $120,000

- Travel: $250,000
- Insurance: $120,000
- Professional services: $300,000
- Contingency: $124,000
- **Total:** $1,364,000

COMPLETE 3-YEAR BUDGET SUMMARY

Category	Year 1	Year 2	Year 3	3-Year Total
Infrastructure & Cloud	$224,600	$904,000	$2,150,000	**$3,278,600**
Personnel	$3,575,000	$5,772,000	$8,500,000	**$17,847,000**
Operations	$522,500	$797,500	$1,364,000	**$2,684,000**
Marketing & Sales	$250,000	$800,000	$1,500,000	**$2,550,000**
Legal & Regulatory	$150,000	$200,000	$300,000	**$650,000**
Contingency (15%)	$652,265	$1,220,925	$2,046,900	**$3,920,090**
TOTAL	**$5,374,365**	**$9,694,425**	**$15,860,900**	**$30,929,690**

FUNDING ROUNDS RECOMMENDATION

Seed Round (Pre-launch): $6,000,000

- Use: MVP development (12 months) + 6 months runway
- Covers: Year 1 costs + buffer
- Equity: 15-20%

Series A (Post-pilot): $12,000,000

- Use: Scale to 50 facilities, expand team
- Covers: Year 2 + expansion costs
- Equity: 20-25%

Series B (Multi-country): $20,000,000

- Use: Regional expansion, 200+ facilities

- Covers: Year 3 + international growth
- Equity: 15-20%

Total Funding Target: $38,000,000 (provides runway buffer)

---

6. INITIAL GEOGRAPHIC FOCUS - DETAILED STRATEGY

Phase 1: South Africa (Months 1-24)

Why Start Here:

1. Most developed healthcare infrastructure in Africa
2. Clear regulatory framework (POPIA)
3. Highest private healthcare spending
4. Strong tech ecosystem
5. Your home market advantage

Target Cities (Priority Order):

1. **Johannesburg/Pretoria** (Gauteng)
    - Largest healthcare market
    - Most private hospitals
    - Tech talent pool
    - 15M+ population
2. **Cape Town** (Western Cape)
    - Second-largest market
    - Strong tech scene
    - Medical tourism hub
    - 4.6M+ population
3. **Durban** (KwaZulu-Natal)
    - Third-largest market
    - Growing healthcare sector
    - 3.9M+ population
4. **Expansion:** Port Elizabeth, Bloemfontein, Nelspruit

South Africa Regulatory Path:

- **POPIA Compliance:** 6-9 months
- **HPCSA Registration:** 3-6 months (if required)
- **Medical schemes liaison:** Ongoing
- **Municipal health department approvals:** Per province

Phase 2: Kenya (Months 18-30)

Why Kenya Next:

1. Most advanced tech ecosystem in East Africa ("Silicon Savannah")
2. M-Pesa integration opportunities
3. Strong government push for digital health
4. English-speaking (easier deployment)
5. Gateway to East Africa

Target Cities:

1. Nairobi (capital, 4.5M)
2. Mombasa (coastal, 1.2M)
3. Kisumu (western, 610K)

Regulatory Requirements:

- Data Protection Act compliance (similar to GDPR)
- Kenya Medical Practitioners Board approval
- Ministry of Health digital health certification
- National Hospital Insurance Fund (NHIF) integration

Estimated Timeline: 12 months for full regulatory approval

Phase 3: Nigeria (Months 24-36)

Why Nigeria:

1. Largest African market (220M+ population)
2. Fastest-growing economy

3. Huge healthcare gap (opportunity)
4. Tech-savvy young population
5. Strong private healthcare sector

Target Cities:

1. Lagos (commercial capital, 15M+)
2. Abuja (capital, 3M+)
3. Port Harcourt (2M+)
4. Kano (Northern market, 4M+)

Regulatory Requirements:

- Nigeria Data Protection Regulation (NDPR) compliance
- Medical and Dental Council of Nigeria (MDCN) approval
- National Health Insurance Scheme (NHIS) integration
- State-level health ministry approvals

Challenges:

- Complex regulatory environment (36 states)
- Infrastructure gaps (electricity, internet)
- Multiple HMO systems
- Security considerations

Estimated Timeline: 18 months for full deployment

Phase 4: Ghana (Months 30-42)

Why Ghana:

1. Stable political environment
2. Strong government digitalization push
3. National Health Insurance Scheme (NHIS) - universal coverage
4. English-speaking
5. Growing middle class

Target Cities:

1. Accra (capital, 2.5M)
2. Kumasi (second city, 2M)
3. Tamale (northern, 540K)

Regulatory Requirements:

- Data Protection Act 2012 compliance
- Medical and Dental Council registration
- Ghana Health Service partnership
- NHIS integration

Estimated Timeline: 10 months

Data Protection Law Compliance Summary

Country	Primary Law	Key Requirements	Penalty	Timeline
South Africa	POPIA	Consent, purpose limitation, data minimization	R10M or 10 years prison	6-9 months
Kenya	Data Protection Act 2019	GDPR-similar, registration required	KSh 5M or 1% revenue	8-12 months
Nigeria	NDPR	Registration, DPO required, audit trails	2% revenue or ₦10M	12-18 months
Ghana	Data Protection Act 2012	Registration, consent management	GH₵ 250K or 5 years	6-10 months

Cross-Country Compliance Strategy:

- **Master Data Agreement:** Template adaptable per country
- **Local Data Residency:** Store citizen data in-country
- **Unified Consent Framework:** GDPR-level baseline
- **Audit Trail:** Comprehensive logging for all countries
- **DPO Network:** Country-level Data Protection Officers

---

7. COMPLIANCE & REGULATORY PRIORITIES

South Africa - POPIA Compliance Roadmap

Phase 1: Foundation (Months 1-3)

- Appoint Information Officer
- Conduct data mapping and inventory
- Define lawful processing bases
- Create privacy policy and notices
- Implement consent management

Phase 2: Technical Implementation (Months 4-6)

- Implement encryption (rest & transit)
- Access control and authentication
- Audit logging system
- Data breach notification system
- Automated data retention/deletion

Phase 3: Processes & Training (Months 7-9)

- Staff training programs
- Vendor due diligence process
- Data subject rights procedures
- Incident response plan
- Third-party audit

Estimated Cost: $100,000 - $150,000

Healthcare-Specific Compliance

Clinical Standards:

- HL7 FHIR compliance
- ICD-10/11 coding
- SNOMED CT integration
- DICOM for imaging

Professional Requirements:

- HPCSA (Health Professions Council SA) consultation
- Medical scheme liaison
- Pharmacy council engagement
- Nursing council coordination

Quality Standards:

- ISO 27001 (Information Security) - Target Year 2
- ISO 13485 (Medical Devices) - If applicable
- HITRUST CSF (Healthcare IT Security) - Target Year 2

IthalaMed Microservices Implementation Guide

Detailed Technical Specifications for Core Services

ARCHITECTURE OVERVIEW

Microservices Design Principles

1. **Single Responsibility**: Each service owns one business domain
2. **Autonomous**: Services can be deployed independently
3. **Resilient**: Failure in one service doesn't cascade
4. **Observable**: Comprehensive logging and monitoring
5. **API-First**: Well-defined REST/GraphQL interfaces
6. **Event-Driven**: Asynchronous communication via message queues

Technology Stack by Service Type

Real-time Services (Node.js + TypeScript):

- Framework: NestJS
- Database ORM: Prisma or TypeORM
- Validation: class-validator
- Testing: Jest

Analytics Services (Python):

- Framework: FastAPI
- Database ORM: SQLAlchemy

- Data Processing: Pandas, NumPy
- ML: TensorFlow, Scikit-learn
- Testing: Pytest

Enterprise Integration (C# .NET Core):

- Framework: ASP.NET Core 8
- ORM: Entity Framework Core
- Testing: xUnit

SERVICE 1: PATIENT SERVICE

Responsibility

Manage patient registration, profiles, demographics, and identity verification.

Technology Stack

- **Language:** Node.js + TypeScript
- **Framework:** NestJS
- **Database:** PostgreSQL (primary), Redis (cache)
- **Port:** 3001

API Endpoints

Patient Registration

POST /api/v1/patients/register

Request Body:

```
{
  "firstName": "Thandi",
  "lastName": "Mbatha",
  "idNumber": "8501156789012", // SA ID number
  "dateOfBirth": "1985-01-15",
  "gender": "female",
  "phoneNumber": "+27821234567",
  "email": "thandi@example.com",
```

```
  "address": {
    "street": "123 Main Road",
    "suburb": "Soweto",
    "city": "Johannesburg",
    "province": "Gauteng",
    "postalCode": "1809",
    "country": "ZA"
  },
  "medicalAid": {
    "scheme": "discovery",
    "membershipNumber": "12345678",
    "dependentCode": "00",
    "plan": "essential"
  },
  "emergencyContact": {
    "name": "John Mbatha",
    "relationship": "spouse",
    "phoneNumber": "+27829876543"
  }
}
```


Response (201 Created):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "patientNumber": "PT202500001",
  "status": "active",
  "createdAt": "2025-01-15T10:30:00Z"
}
```

Get Patient Profile

GET /api/v1/patients/{patientId}

Headers:

Authorization: Bearer {jwt_token}

Response (200 OK):


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "patientNumber": "PT202500001",
  "firstName": "Thandi",
  "lastName": "Mbatha",
  "dateOfBirth": "1985-01-15",
  "gender": "female",
  "bloodType": "A+",
  "allergies": ["penicillin", "peanuts"],
  "chronicConditions": ["hypertension"],
  "medicalAid": {...},
  "lastVisit": "2024-12-10T14:00:00Z"
}
```


Update Patient Profile

PATCH /api/v1/patients/{patientId}

Request Body:


```
{
  "phoneNumber": "+27821234568",
  "address": {...}
}
```


Response (200 OK):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "updatedAt": "2025-01-15T11:00:00Z"
}
```


Search Patients

GET /api/v1/patients/search?q=Mbatha&limit=20&offset=0

Response (200 OK):


```
{
  "results": [
    {
      "id": "...",
      "patientNumber": "PT202500001",
      "fullName": "Thandi Mbatha",
      "dateOfBirth": "1985-01-15",
      "lastVisit": "2024-12-10"
    }
  ],
  "total": 1,
  "limit": 20,
  "offset": 0
}
```


Domain Model

```
// patient.entity.ts
@Entity('patients')
export class Patient {
  @PrimaryGeneratedColumn('uuid')
  id: string;
```

@Column({ unique: true })

patientNumber: string; // PT202500001

@Column()

firstName: string;

@Column()

lastName: string;

@Column({ unique: true })

idNumber: string; // SA ID, encrypted

@Column({ type: 'date' })

dateOfBirth: Date;

@Column({ type: 'enum', enum: ['male', 'female', 'other'] })

gender: string;

@Column({ unique: true })

phoneNumber: string;

@Column({ unique: true, nullable: true })

email: string;

@Column({ type: 'enum', enum: ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'], nullable: true })

bloodType: string;

@Column({ type: 'jsonb' })

```
  address: Address;

  @Column({ type: 'jsonb', nullable: true })
  medicalAid: MedicalAid;

  @Column({ type: 'jsonb' })
  emergencyContact: EmergencyContact;

  @Column({ type: 'text', array: true, default: [] })
  allergies: string[];

  @Column({ type: 'text', array: true, default: [] })
  chronicConditions: string[];

  @Column({ type: 'enum', enum: ['active', 'inactive', 'deceased'], default: 'active' })
  status: string;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;

  @DeleteDateColumn()
  deletedAt: Date; // Soft delete
}
```

Business Logic

```
// patient.service.ts
```

```
@Injectable()
export class PatientService {
 constructor(
  @InjectRepository(Patient)
  private patientRepository: Repository<Patient>,
  private readonly cacheService: CacheService,
  private readonly eventBus: EventBus,
  private readonly auditService: AuditService,
 ) {}

 async register(dto: RegisterPatientDto): Promise<Patient> {
  // 1. Validate ID number (Luhn algorithm for SA ID)
  if (!this.validateSAID(dto.idNumber)) {
   throw new BadRequestException('Invalid SA ID number');
  }

  // 2. Check for duplicates
  const existing = await this.patientRepository.findOne({
   where: { idNumber: dto.idNumber }
  });
  if (existing) {
   throw new ConflictException('Patient already registered');
  }

  // 3. Generate patient number
  const patientNumber = await this.generatePatientNumber();

  // 4. Create patient
```

```
  const patient = this.patientRepository.create({
    ...dto,
    patientNumber,
    idNumber: this.encrypt(dto.idNumber), // Encrypt PII
  });

  // 5. Save to database
  const savedPatient = await this.patientRepository.save(patient);

  // 6. Clear cache and emit event
  await this.cacheService.del(`patient:${savedPatient.id}`);
  this.eventBus.publish(new PatientRegisteredEvent(savedPatient));

  // 7. Audit log
  await this.auditService.log({
    action: 'PATIENT_REGISTERED',
    entityType: 'Patient',
    entityId: savedPatient.id,
    userId: 'system',
  });

  return savedPatient;
}

async findById(id: string): Promise<Patient> {
  // Try cache first
  const cached = await this.cacheService.get(`patient:${id}`);
  if (cached) return cached;
```

```
  // Fetch from database
  const patient = await this.patientRepository.findOne({
    where: { id },
  });

  if (!patient) {
    throw new NotFoundException('Patient not found');
  }

  // Cache for 5 minutes
  await this.cacheService.set(`patient:${id}`, patient, 300);

  return patient;
}

private validateSAID(idNumber: string): boolean {
  // SA ID validation logic (Luhn algorithm)
  // Format: YYMMDDGSSSCAZ
  // Where: YY=year, MM=month, DD=day, G=gender, SSS=sequence, C=citizenship,
A=race (apartheid era, now obsolete), Z=checksum
  if (!/^\d{13}$/.test(idNumber)) return false;

  // Luhn algorithm for checksum
  const digits = idNumber.split('').map(Number);
  const checksum = digits.pop();
  let sum = 0;
```

```
  for (let i = 0; i < digits.length; i++) {
    let digit = digits[i];
    if (i % 2 === 1) {
      digit *= 2;
      if (digit > 9) digit -= 9;
    }
    sum += digit;
  }

  return (10 - (sum % 10)) % 10 === checksum;
}

private async generatePatientNumber(): Promise<string> {
  const year = new Date().getFullYear();
  const count = await this.patientRepository.count({
    where: {
      createdAt: Between(
        new Date(`${year}-01-01`),
        new Date(`${year}-12-31`)
      )
    }
  });

  return `PT${year}${String(count + 1).padStart(5, '0')}`;
}

private encrypt(data: string): string {
  // Use AWS KMS or similar
```

```
    return crypto.encrypt(data);
  }
}
```

Events Published

```
export class PatientRegisteredEvent {
  constructor(public readonly patient: Patient) {}
}

export class PatientUpdatedEvent {
  constructor(
    public readonly patientId: string,
    public readonly changes: Partial<Patient>
  ) {}
}

export class PatientConsentUpdatedEvent {
  constructor(
    public readonly patientId: string,
    public readonly consentType: string,
    public readonly granted: boolean
  ) {}
}
```

Integration Points

Consumed Events:

- UserAuthenticatedEvent (from Auth Service) - Link user account to patient
- MedicalAidVerifiedEvent (from Insurance Service) - Update medical aid status

External APIs:

- HPCSA API - Verify provider credentials

- Medical aid APIs - Verify membership

Security Considerations

1. **PII Encryption**: All sensitive data encrypted at rest
2. **Access Control**: RBAC - only authorized users can access patient data
3. **Audit Trail**: All access logged with user, timestamp, action
4. **Data Minimization**: Only return necessary fields based on user role
5. **Consent Management**: Check patient consent before sharing data

Performance Optimization

1. **Caching**: Redis cache for frequently accessed patient profiles (5 min TTL)
2. **Database Indexing**: Index on patientNumber, idNumber, phoneNumber, email
3. **Pagination**: All list endpoints paginated (default 20, max 100)
4. **Lazy Loading**: Don't load medical history in profile endpoint
5. **Connection Pooling**: PostgreSQL connection pool (min: 5, max: 50)

Monitoring & Alerts

Metrics:

- Patient registration rate (per hour)
- API response time (p50, p95, p99)
- Cache hit rate
- Error rate

Alerts:

- Registration failures > 5% (5 min window)
- API response time p95 > 500ms
- Database connection pool exhausted
- Cache unavailable

SERVICE 2: APPOINTMENT SERVICE

Responsibility

Manage appointment scheduling, calendar availability, reminders, and no-show tracking.

Technology Stack

- **Language:** Node.js + TypeScript
- **Framework:** NestJS
- **Database:** PostgreSQL (primary), Redis (lock management)
- **Message Queue:** RabbitMQ (for reminders)
- **Port:** 3002

API Endpoints

Get Provider Availability

GET /api/v1/appointments/availability/{providerId}?date=2025-01-20

Response (200 OK):


```
{
  "providerId": "550e8400-e29b-41d4-a716-446655440001",
  "date": "2025-01-20",
  "availableSlots": [
    {
      "startTime": "09:00",
      "endTime": "09:30",
      "available": true
    },
    {
      "startTime": "09:30",
      "endTime": "10:00",
      "available": false
    },
    // ... more slots
  ],
```

```
  "workingHours": {
    "start": "09:00",
    "end": "17:00"
  }
}
```

Book Appointment

POST /api/v1/appointments

Request Body:

```
{
  "patientId": "550e8400-e29b-41d4-a716-446655440000",
  "providerId": "550e8400-e29b-41d4-a716-446655440001",
  "facilityId": "550e8400-e29b-41d4-a716-446655440002",
  "appointmentType": "consultation",
  "date": "2025-01-20",
  "startTime": "09:00",
  "duration": 30, // minutes
  "reason": "Persistent headache",
  "notes": "First visit"
}
```

Response (201 Created):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440003",
  "appointmentNumber": "APT202500001",
  "status": "scheduled",
  "confirmationCode": "ABC123",
  "scheduledFor": "2025-01-20T09:00:00Z"
}
```

Get Appointments

GET /api/v1/appointments?patientId={id}&status=scheduled&from=2025-01-01&to=2025-01-31

Response (200 OK):


```
{
  "appointments": [
    {
      "id": "...",
      "appointmentNumber": "APT202500001",
      "provider": {
        "id": "...",
        "name": "Dr. Sarah Mokoena",
        "specialty": "General Practice"
      },
      "facility": {
        "id": "...",
        "name": "Soweto Medical Centre"
      },
      "scheduledFor": "2025-01-20T09:00:00Z",
      "duration": 30,
      "status": "scheduled",
      "appointmentType": "consultation"
    }
  ],
  "total": 1
}
```


Cancel Appointment

DELETE /api/v1/appointments/{appointmentId}

Request Body:

```
{
  "reason": "Schedule conflict",
  "cancelledBy": "patient"
}
```

Response (200 OK):

```
{
  "id": "...",
  "status": "cancelled",
  "cancelledAt": "2025-01-18T10:00:00Z"
}
```

Domain Model

```
@Entity('appointments')
export class Appointment {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column({ unique: true })
  appointmentNumber: string;

  @ManyToOne(() => Patient)
  @JoinColumn({ name: 'patient_id' })
  patient: Patient;

  @Column()
  patientId: string;
```

```
  @ManyToOne(() => Provider)
  @JoinColumn({ name: 'provider_id' })
  provider: Provider;

  @Column()
  providerId: string;

  @ManyToOne(() => Facility)
  @JoinColumn({ name: 'facility_id' })
  facility: Facility;

  @Column()
  facilityId: string;

  @Column({ type: 'enum', enum: ['consultation', 'follow-up', 'procedure', 'lab', 'imaging'] })
  appointmentType: string;

  @Column({ type: 'timestamp' })
  scheduledFor: Date;

  @Column({ type: 'int' })
  duration: number; // minutes

  @Column({ type: 'enum', enum: ['scheduled', 'confirmed', 'checked-in', 'in-progress', 'completed', 'cancelled', 'no-show'] })
  status: string;

  @Column({ nullable: true })
```

```
reason: string;

@Column({ type: 'text', nullable: true })
notes: string;

@Column({ nullable: true })
confirmationCode: string;

@Column({ type: 'timestamp', nullable: true })
checkedInAt: Date;

@Column({ type: 'timestamp', nullable: true })
completedAt: Date;

@Column({ type: 'timestamp', nullable: true })
cancelledAt: Date;

@Column({ nullable: true })
cancelledBy: string; // 'patient', 'provider', 'system'

@Column({ nullable: true })
cancellationReason: string;

@CreateDateColumn()
createdAt: Date;

@UpdateDateColumn()
updatedAt: Date;
```

```
}
```

Business Logic

```
@Injectable()
export class AppointmentService {
 constructor(
  @InjectRepository(Appointment)
  private appointmentRepository: Repository<Appointment>,
  private readonly lockService: LockService, // Redis-based distributed lock
  private readonly eventBus: EventBus,
  private readonly notificationService: NotificationService,
 ) {}

 async bookAppointment(dto: BookAppointmentDto): Promise<Appointment> {
  const lockKey = `appointment:lock:${dto.providerId}:${dto.date}:${dto.startTime}`;

  // Acquire distributed lock to prevent double-booking
  const lock = await this.lockService.acquire(lockKey, 5000); // 5 second timeout

  try {
   // 1. Check slot availability
   const isAvailable = await this.checkAvailability(
    dto.providerId,
    dto.date,
    dto.startTime,
    dto.duration
   );

   if (!isAvailable) {
```

```
  throw new ConflictException('Time slot not available');
}

// 2. Validate patient and provider exist
// (Call Patient Service and Provider Service via HTTP or event)

// 3. Generate appointment number and confirmation code
const appointmentNumber = await this.generateAppointmentNumber();
const confirmationCode = this.generateConfirmationCode();

// 4. Create appointment
const appointment = this.appointmentRepository.create({
  ...dto,
  appointmentNumber,
  confirmationCode,
  status: 'scheduled',
  scheduledFor: this.combineDateTime(dto.date, dto.startTime),
});

// 5. Save to database
const savedAppointment = await this.appointmentRepository.save(appointment);

// 6. Schedule reminders
await this.scheduleReminders(savedAppointment);

// 7. Emit event
this.eventBus.publish(new AppointmentBookedEvent(savedAppointment));
```

```
    // 8. Send confirmation notification
    await this.notificationService.send({
      type: 'APPOINTMENT_CONFIRMATION',
      patientId: dto.patientId,
      channels: ['sms', 'push', 'email'],
      data: savedAppointment,
    });

    return savedAppointment;

  } finally {
    // Always release lock
    await lock.release();
  }
}

async checkAvailability(
  providerId: string,
  date: string,
  startTime: string,
  duration: number
): Promise<boolean> {
  const scheduledFor = this.combineDateTime(date, startTime);
  const endTime = new Date(scheduledFor.getTime() + duration * 60000);

  // Check for overlapping appointments
  const overlapping = await this.appointmentRepository.count({
    where: {
```

```
    providerId,
    status: In(['scheduled', 'confirmed', 'in-progress']),
    scheduledFor: Between(scheduledFor, endTime),
   },
  });

  return overlapping === 0;
}

async getAvailability(providerId: string, date: string) {
  // 1. Get provider's working hours for this day
  const workingHours = await this.getProviderWorkingHours(providerId, date);

  // 2. Get all scheduled appointments for this day
  const appointments = await this.appointmentRepository.find({
   where: {
    providerId,
    scheduledFor: Between(
     new Date(`${date}T00:00:00`),
     new Date(`${date}T23:59:59`)
    ),
    status: In(['scheduled', 'confirmed', 'in-progress']),
   },
  });

  // 3. Generate available slots (30-minute intervals)
  const slots = this.generateTimeSlots(workingHours, appointments);
```

```
  return slots;
 }

 private async scheduleReminders(appointment: Appointment) {
  // Schedule reminder 24 hours before
  const reminder24h = new Date(appointment.scheduledFor.getTime() - 24 * 60 * 60 *
1000);
  await this.queueReminder(appointment, reminder24h, '24_HOURS');

  // Schedule reminder 2 hours before
  const reminder2h = new Date(appointment.scheduledFor.getTime() - 2 * 60 * 60 *
1000);
  await this.queueReminder(appointment, reminder2h, '2_HOURS');
 }

 private async queueReminder(appointment: Appointment, sendAt: Date, type: string) {
  // Use RabbitMQ delayed message or external scheduler service
  await this.messageQueue.publish('appointment.reminders', {
   appointmentId: appointment.id,
   sendAt: sendAt.toISOString(),
   type,
  });
 }

 private generateConfirmationCode(): string {
  // Generate 6-character alphanumeric code
  return Math.random().toString(36).substring(2, 8).toUpperCase();
 }
```

```
  private combineDateTime(date: string, time: string): Date {
    return new Date(`${date}T${time}:00`);
  }
}
```

Events Published

```
export class AppointmentBookedEvent {
  constructor(public readonly appointment: Appointment) {}
}

export class AppointmentCancelledEvent {
  constructor(
    public readonly appointmentId: string,
    public readonly reason: string
  ) {}
}

export class AppointmentCompletedEvent {
  constructor(public readonly appointment: Appointment) {}
}

export class AppointmentNoShowEvent {
  constructor(public readonly appointment: Appointment) {}
}
```

Background Jobs

```
// Check for no-shows (runs every hour)
@Cron('0 * * * *')
async checkNoShows() {
  const now = new Date();
```

```
  const threshold = new Date(now.getTime() - 30 * 60 * 1000); // 30 min ago

  const noShows = await this.appointmentRepository.find({
    where: {
      status: 'scheduled',
      scheduledFor: LessThan(threshold),
    },
  });

  for (const appointment of noShows) {
    appointment.status = 'no-show';
    await this.appointmentRepository.save(appointment);
    this.eventBus.publish(new AppointmentNoShowEvent(appointment));
  }
}
```

SERVICE 3: PRESCRIPTION SERVICE (E-PRESCRIPTION)

Responsibility

Manage digital prescription creation, routing to pharmacies, and drug interaction checking.

Technology Stack

- **Language:** Node.js + TypeScript
- **Framework:** NestJS
- **Database:** PostgreSQL
- **External APIs:** Drug database API (DrugBank or similar)
- **Port:** 3003

API Endpoints

Create Prescription

POST /api/v1/prescriptions

Request Body:

```
{
  "patientId": "550e8400-e29b-41d4-a716-446655440000",
  "providerId": "550e8400-e29b-41d4-a716-446655440001",
  "encounterId": "550e8400-e29b-41d4-a716-446655440010",
  "medications": [
    {
      "drugName": "Amlodipine",
      "drugCode": "NAPPI123456", // South African NAPPI code
      "dosage": "5mg",
      "frequency": "once daily",
      "duration": 30,
      "durationUnit": "days",
      "quantity": 30,
      "instructions": "Take in the morning with food",
      "refills": 2
    }
  ],
  "diagnosis": "Hypertension",
  "notes": "Monitor blood pressure weekly",
  "preferredPharmacy": "550e8400-e29b-41d4-a716-446655440020"
}
```

Response (201 Created):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440030",
  "prescriptionNumber": "RX202500001",
```

```
  "status": "active",
  "expiresAt": "2025-07-20T00:00:00Z",
  "eSignature": "...", // Digital signature
  "qrCode": "..." // QR code for pharmacy scanning
}
```

Get Prescription

GET /api/v1/prescriptions/{prescriptionId}

Response (200 OK):

```
{
  "id": "...",
  "prescriptionNumber": "RX202500001",
  "patient": {
    "id": "...",
    "name": "Thandi Mbatha",
    "dateOfBirth": "1985-01-15"
  },
  "provider": {
    "id": "...",
    "name": "Dr. Sarah Mokoena",
    "hpcsaNumber": "MP123456"
  },
  "medications": [...],
  "issuedAt": "2025-01-20T10:00:00Z",
  "expiresAt": "2025-07-20T00:00:00Z",
  "status": "active",
  "dispensedCount": 0,
  "refillsRemaining": 2
}
```

Dispense Prescription

POST /api/v1/prescriptions/{prescriptionId}/dispense

Request Body:

```
{
 "pharmacyId": "550e8400-e29b-41d4-a716-446655440020",
 "pharmacistId": "550e8400-e29b-41d4-a716-446655440021",
 "medications": [
  {
   "medicationId": "...",
   "quantityDispensed": 30,
   "batchNumber": "BATCH123",
   "expiryDate": "2026-12-31"
  }
 ]
}
```

Response (200 OK):

```
{
 "dispensingId": "...",
 "dispensedAt": "2025-01-20T15:00:00Z",
 "status": "dispensed"
}
```

Domain Model

```
@Entity('prescriptions')
export class Prescription {
 @PrimaryGeneratedColumn('uuid')
 id: string;
```

```
  @Column({ unique: true })
  prescriptionNumber: string;

  @Column()
  patientId: string;

  @Column()
  providerId: string;

  @Column({ nullable: true })
  encounterId: string; // Link to appointment/consultation

  @Column({ type: 'jsonb' })
  medications: PrescriptionMedication[];

  @Column({ nullable: true })
  diagnosis: string;

  @Column({ type: 'text', nullable: true })
  notes: string;

  @Column({ type: 'enum', enum: ['active', 'dispensed', 'partially-dispensed', 'expired',
'cancelled'] })
  status: string;

  @Column({ type: 'timestamp' })
  issuedAt: Date;
```

```
  @Column({ type: 'timestamp' })
  expiresAt: Date; // Typically 6 months from issue date

  @Column({ type: 'text' })
  eSignature: string; // Digital signature of provider

  @Column({ type: 'text' })
  qrCode: string; // QR code for verification

  @Column({ nullable: true })
  preferredPharmacyId: string;

  @Column({ type: 'int', default: 0 })
  dispensedCount: number;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;

  @OneToMany(() => PrescriptionDispensing, dispensing => dispensing.prescription)
  dispensings: PrescriptionDispensing[];
}

@Entity('prescription_dispensings')
export class PrescriptionDispensing {
  @PrimaryGeneratedColumn('uuid')
```

```
  id: string;

  @ManyToOne(() => Prescription, prescription => prescription.dispensings)
  @JoinColumn({ name: 'prescription_id' })
  prescription: Prescription;

  @Column()
  prescriptionId: string;

  @Column()
  pharmacyId: string;

  @Column()
  pharmacistId: string;

  @Column({ type: 'jsonb' })
  medications: DispensedMedication[];

  @Column({ type: 'timestamp' })
  dispensedAt: Date;

  @CreateDateColumn()
  createdAt: Date;
}

interface PrescriptionMedication {
  drugName: string;
  drugCode: string; // NAPPI code
```

```
  dosage: string;
  frequency: string;
  duration: number;
  durationUnit: string;
  quantity: number;
  instructions: string;
  refills: number;
  isControlled: boolean; // Schedule 5/6 drugs
}
```

Business Logic

```
@Injectable()
export class PrescriptionService {
  constructor(
    @InjectRepository(Prescription)
    private prescriptionRepository: Repository<Prescription>,
    private readonly drugInteractionService: DrugInteractionService,
    private readonly eSignatureService: ESignatureService,
    private readonly eventBus: EventBus,
  ) {}

  async createPrescription(dto: CreatePrescriptionDto): Promise<Prescription> {
    // 1. Get patient's current medications
    const currentMedications = await this.getPatientCurrentMedications(dto.patientId);

    // 2. Check drug interactions
    const interactions = await this.drugInteractionService.check(
      [...dto.medications.map(m => m.drugCode), ...currentMedications]
    );
```

```
  if (interactions.severity === 'severe') {
    throw new BadRequestException({
      message: 'Severe drug interaction detected',
      interactions: interactions.details
    });
  }

  // 3. Check patient allergies
  const patient = await this.patientService.findById(dto.patientId);
  const allergyConflicts = this.checkAllergyConflicts(dto.medications,
patient.allergies);

  if (allergyConflicts.length > 0) {
    throw new BadRequestException({
      message: 'Patient allergic to prescribed medication',
      conflicts: allergyConflicts
    });
  }

  // 4. Generate prescription number
  const prescriptionNumber = await this.generatePrescriptionNumber();

  // 5. Calculate expiry date (6 months)
  const expiresAt = new Date();
  expiresAt.setMonth(expiresAt.getMonth() + 6);

  // 6. Generate e-signature
```

```
const eSignature = await this.eSignatureService.sign({
  prescriptionNumber,
  providerId: dto.providerId,
  medications: dto.medications,
  issuedAt: new Date(),
});

// 7. Create prescription
const prescription = this.prescriptionRepository.create({
  ...dto,
  prescriptionNumber,
  status: 'active',
  issuedAt: new Date(),
  expiresAt,
  eSignature,
  dispensedCount: 0,
});

// 8. Save to database
const savedPrescription = await this.prescriptionRepository.save(prescription);

// 9. Generate QR code
savedPrescription.qrCode = await this.generateQRCode(savedPrescription);
await this.prescriptionRepository.save(savedPrescription);

// 10. Route to pharmacy if specified
if (dto.preferredPharmacy) {
  this.eventBus.publish(new PrescriptionCreatedEvent(savedPrescription));
```

```
  }

  // 11. Notify patient
  await this.notificationService.send({
    type: 'PRESCRIPTION_READY',
    patientId: dto.patientId,
    channels: ['push', 'sms'],
    data: {
      prescriptionNumber,
      pharmacy: dto.preferredPharmacy
    }
  });

  return savedPrescription;
}

async dispensePrescription(
  prescriptionId: string,
  dto: DispensePrescriptionDto
): Promise<PrescriptionDispensing> {
  // 1. Verify prescription is valid
  const prescription = await this.prescriptionRepository.findOne({
    where: { id: prescriptionId }
  });

  if (!prescription) {
    throw new NotFoundException('Prescription not found');
  }
```

```
if (prescription.status === 'expired') {
  throw new BadRequestException('Prescription has expired');
}

if (prescription.expiresAt < new Date()) {
  throw new BadRequestException('Prescription has expired');
}

// 2. Verify pharmacist credentials
await this.verifyPharmacist(dto.pharmacistId);

// 3. Create dispensing record
const dispensing = this.dispensingRepository.create({
  prescriptionId,
  pharmacyId: dto.pharmacyId,
  pharmacistId: dto.pharmacistId,
  medications: dto.medications,
  dispensedAt: new Date(),
});

await this.dispensingRepository.save(dispensing);

// 4. Update prescription status
prescription.dispensedCount += 1;

const totalMedications = prescription.medications.length;
```

```
  const allDispensed = prescription.dispensedCount >= (totalMedications * (1 +
prescription.medications[0].refills));

  if (allDispensed) {
    prescription.status = 'dispensed';
  } else {
    prescription.status = 'partially-dispensed';
  }

  await this.prescriptionRepository.save(prescription);

  // 5. Emit event
  this.eventBus.publish(new PrescriptionDispensedEvent(prescription, dispensing));

  // 6. Notify patient
  await this.notificationService.send({
    type: 'PRESCRIPTION_DISPENSED',
    patientId: prescription.patientId,
    channels: ['push'],
    data: {
      prescriptionNumber: prescription.prescriptionNumber,
      pharmacy: dto.pharmacyId
    }
  });

  return dispensing;
 }
```

```
private checkAllergyConflicts(medications: any[], allergies: string[]): string[] {
  const conflicts = [];

  for (const med of medications) {
    for (const allergy of allergies) {
      if (med.drugName.toLowerCase().includes(allergy.toLowerCase())) {
        conflicts.push({
          medication: med.drugName,
          allergy: allergy
        });
      }
    }
  }

  return conflicts;
}

private async generateQRCode(prescription: Prescription): Promise<string> {
  const qrData = {
    prescriptionNumber: prescription.prescriptionNumber,
    patientId: prescription.patientId,
    issuedAt: prescription.issuedAt,
    eSignature: prescription.eSignature
  };

  // Generate QR code as base64 image
  return await QRCode.toDataURL(JSON.stringify(qrData));
}
```

}

SERVICE 4: TELEMEDICINE SERVICE

Responsibility

Manage video consultations, virtual waiting rooms, and session recordings.

Technology Stack

- **Language:** Node.js + TypeScript
- **Framework:** NestJS
- **Video SDK:** Agora.io or Twilio Video
- **Database:** PostgreSQL + MongoDB (chat logs)
- **Port:** 3004

API Endpoints

Create Consultation Session

POST /api/v1/telemedicine/sessions

Request Body:

```
{
  "appointmentId": "550e8400-e29b-41d4-a716-446655440003",
  "patientId": "550e8400-e29b-41d4-a716-446655440000",
  "providerId": "550e8400-e29b-41d4-a716-446655440001",
  "sessionType": "video", // or "audio-only"
  "recordSession": true
}
```

Response (201 Created):

```
{
  "sessionId": "550e8400-e29b-41d4-a716-446655440040",
  "sessionToken": "eyJhbGc...", // JWT for video SDK
  "channelName": "consultation-202501-001",
```

```
  "agoraAppId": "...",
  "expiresAt": "2025-01-20T11:00:00Z",
  "status": "waiting"
}
```

Join Session

POST /api/v1/telemedicine/sessions/{sessionId}/join

Request Body:

```
{
  "userId": "550e8400-e29b-41d4-a716-446655440001",
  "role": "provider" // or "patient"
}
```

Response (200 OK):

```
{
  "sessionToken": "eyJhbGc...",
  "rtcToken": "...", // Agora RTC token
  "channelName": "consultation-202501-001",
  "participantId": "..."
}
```

End Session

POST /api/v1/telemedicine/sessions/{sessionId}/end

Request Body:

```
{
  "duration": 1800, // seconds
  "notes": "Patient reports improvement",
  "followUpRequired": false
}
```

Response (200 OK):

```
{
  "sessionId": "...",
  "status": "completed",
  "duration": 1800,
  "recordingUrl": "https://storage.../recording.mp4"
}
```

Domain Model

```
@Entity('telemedicine_sessions')
export class TelemedicineSession {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column()
  appointmentId: string;

  @Column()
  patientId: string;

  @Column()
  providerId: string;

  @Column({ unique: true })
  channelName: string;

  @Column({ type: 'enum', enum: ['video', 'audio-only'] })
  sessionType: string;
```

```
@Column({ type: 'enum', enum: ['waiting', 'active', 'completed', 'cancelled'] })
status: string;

@Column({ type: 'boolean', default: false })
recordSession: boolean;

@Column({ nullable: true })
recordingUrl: string;

@Column({ type: 'timestamp', nullable: true })
startedAt: Date;

@Column({ type: 'timestamp', nullable: true })
endedAt: Date;

@Column({ type: 'int', nullable: true })
duration: number; // seconds

@Column({ type: 'text', nullable: true })
notes: string;

@Column({ type: 'jsonb', default: [] })
participants: SessionParticipant[];

@Column({ type: 'jsonb', default: {} })
quality: SessionQuality;

@CreateDateColumn()
```

```
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}

interface SessionParticipant {
  userId: string;
  role: 'patient' | 'provider';
  joinedAt: Date;
  leftAt?: Date;
}

interface SessionQuality {
  videoQuality: string;
  audioQuality: string;
  networkLatency: number;
  packetLoss: number;
}
```

Business Logic

```
@Injectable()
export class TelemedicineService {
  constructor(
    @InjectRepository(TelemedicineSession)
    private sessionRepository: Repository<TelemedicineSession>,
    private readonly agoraService: AgoraService,
    private readonly eventBus: EventBus,
  ) {}
```

```
 async createSession(dto: CreateSessionDto): Promise<TelemedicineSession> {
  // 1. Verify appointment exists and is scheduled
  const appointment = await this.appointmentService.findById(dto.appointmentId);

  if (appointment.status !== 'scheduled') {
   throw new BadRequestException('Appointment not in scheduled state');
  }

  // 2. Generate unique channel name
  const channelName = `consultation-${Date.now()}-${dto.appointmentId.substring(0,
8)}`;

  // 3. Create session
  const session = this.sessionRepository.create({
   ...dto,
   channelName,
   status: 'waiting',
   participants: []
  });

  const savedSession = await this.sessionRepository.save(session);

  // 4. Emit event
  this.eventBus.publish(new TelemedicineSessionCreatedEvent(savedSession));

  return savedSession;
 }
```

```
async joinSession(sessionId: string, dto: JoinSessionDto) {
 const session = await this.sessionRepository.findOne({
  where: { id: sessionId }
 });

 if (!session) {
  throw new NotFoundException('Session not found');
 }

 // Generate Agora RTC token
 const rtcToken = await this.agoraService.generateToken({
  channelName: session.channelName,
  uid: dto.userId,
  role: dto.role,
  expirationTimeInSeconds: 3600
 });

 // Add participant
 session.participants.push({
  userId: dto.userId,
  role: dto.role,
  joinedAt: new Date()
 });

 // Update status if this is first participant
 if (session.status === 'waiting' && session.participants.length === 1) {
  session.status = 'active';
```

```
    session.startedAt = new Date();
  }

  await this.sessionRepository.save(session);

  return {
    sessionToken: this.generateSessionToken(session),
    rtcToken,
    channelName: session.channelName,
    participantId: dto.userId
  };
}

async endSession(sessionId: string, dto: EndSessionDto) {
  const session = await this.sessionRepository.findOne({
    where: { id: sessionId }
  });

  if (!session) {
    throw new NotFoundException('Session not found');
  }

  session.status = 'completed';
  session.endedAt = new Date();
  session.duration = dto.duration;
  session.notes = dto.notes;

  // If recording was enabled, get recording URL from Agora
```

```
    if (session.recordSession) {
      session.recordingUrl = await this.agoraService.getRecordingUrl(session.channelName);
    }

    await this.sessionRepository.save(session);

    // Emit event
    this.eventBus.publish(new TelemedicineSessionCompletedEvent(session));

    // Update appointment status
    await this.appointmentService.updateStatus(session.appointmentId, 'completed');

    return session;
  }
}
```

CROSS-CUTTING CONCERNS

1. API Gateway Configuration

```
// Kong API Gateway configuration
services:
  - name: patient-service
    url: http://patient-service:3001
    routes:
      - name: patient-routes
        paths:
          - /api/v1/patients
        methods:
```

```
      - GET
      - POST
      - PATCH
      - DELETE
    plugins:
     - name: jwt
       config:
         key_claim_name: kid
         secret_is_base64: false
     - name: rate-limiting
       config:
         minute: 100
         hour: 1000
     - name: correlation-id
     - name: request-transformer
       config:
         add:
           headers:
             - X-Service-Name: patient-service


- name: appointment-service
  url: http://appointment-service:3002
  routes:
    - name: appointment-routes
      paths:
        - /api/v1/appointments
      # ... similar configuration
```

2. Service Discovery

```
// Using Consul for service discovery
import * as Consul from 'consul';

@Injectable()
export class ServiceDiscovery {
 private consul: Consul.Consul;

 constructor() {
  this.consul = new Consul({
   host: process.env.CONSUL_HOST,
   port: process.env.CONSUL_PORT
  });
 }

 async registerService(service: ServiceInfo) {
  await this.consul.agent.service.register({
   id: service.id,
   name: service.name,
   address: service.host,
   port: service.port,
   check: {
    http: `http://${service.host}:${service.port}/health`,
    interval: '10s',
    timeout: '5s'
   }
  });
 }
```

```
  async discoverService(serviceName: string): Promise<string> {
    const result = await this.consul.health.service({
      service: serviceName,
      passing: true
    });

    if (result.length === 0) {
      throw new Error(`Service ${serviceName} not found`);
    }

    // Simple round-robin
    const service = result[Math.floor(Math.random() * result.length)];
    return `http://${service.Service.Address}:${service.Service.Port}`;
  }
}
```

3. Circuit Breaker Pattern

```
import * as CircuitBreaker from 'opossum';

@Injectable()
export class ResilientHttpService {
  private breakers = new Map<string, CircuitBreaker>();

  async call(serviceName: string, options: RequestOptions) {
    if (!this.breakers.has(serviceName)) {
      this.breakers.set(
        serviceName,
        new CircuitBreaker(this.makeRequest, {
          timeout: 5000, // 5 seconds
```

```
      errorThresholdPercentage: 50,
      resetTimeout: 30000, // 30 seconds
      name: serviceName
    })
  );
 }

 const breaker = this.breakers.get(serviceName);
 return breaker.fire(options);
}

 private async makeRequest(options: RequestOptions) {
  return axios(options);
 }
}
```

4. Event Bus Implementation

```
// Using RabbitMQ for event-driven communication
import * as amqp from 'amqplib';

@Injectable()
export class EventBus {
 private connection: amqp.Connection;
 private channel: amqp.Channel;

 async connect() {
  this.connection = await amqp.connect(process.env.RABBITMQ_URL);
  this.channel = await this.connection.createChannel();
```

```
  // Declare exchange
  await this.channel.assertExchange('ithalamed.events', 'topic', {
    durable: true
  });
}

async publish(event: DomainEvent) {
  const routingKey = `${event.aggregateType}.${event.eventType}`;

  this.channel.publish(
    'ithalamed.events',
    routingKey,
    Buffer.from(JSON.stringify(event)),
    {
      persistent: true,
      contentType: 'application/json',
      timestamp: Date.now()
    }
  );
}

async subscribe(pattern: string, handler: (event: DomainEvent) => Promise<void>) {
  const queue = await this.channel.assertQueue('', { exclusive: true });

  await this.channel.bindQueue(queue.queue, 'ithalamed.events', pattern);

  this.channel.consume(queue.queue, async (msg) => {
    if (msg) {
```

```
      const event = JSON.parse(msg.content.toString());
      await handler(event);
      this.channel.ack(msg);
    }
  });
 }
}
```

5. Distributed Tracing

```
// Using OpenTelemetry
import { trace, context } from '@opentelemetry/api';
import { NodeTracerProvider } from '@opentelemetry/sdk-trace-node';
import { JaegerExporter } from '@opentelemetry/exporter-jaeger';

const provider = new NodeTracerProvider();
provider.addSpanProcessor(
 new BatchSpanProcessor(
  new JaegerExporter({
   endpoint: process.env.JAEGER_ENDPOINT
  })
 )
);

provider.register();

// Middleware for tracing HTTP requests
@Injectable()
export class TracingInterceptor implements NestInterceptor {
 intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
```

```
    const tracer = trace.getTracer('ithalamed');
    const span = tracer.startSpan('http-request');

    span.setAttribute('http.method', context.switchToHttp().getRequest().method);
    span.setAttribute('http.url', context.switchToHttp().getRequest().url);

    return next.handle().pipe(
      tap(() => span.end()),
      catchError((error) => {
        span.recordException(error);
        span.end();
        throw error;
      })
    );
  }
}
```

DEPLOYMENT CONFIGURATION

Docker Compose (Development)

```
version: '3.8'

services:
  patient-service:
    build:
      context: ./services/patient
      dockerfile: Dockerfile
    ports:
      - "3001:3001"
```

```
  environment:
    - DATABASE_URL=postgresql://user:pass@postgres:5432/patient_db
    - REDIS_URL=redis://redis:6379
    - RABBITMQ_URL=amqp://rabbitmq:5672
  depends_on:
    - postgres
    - redis
    - rabbitmq

appointment-service:
  build:
    context: ./services/appointment
    dockerfile: Dockerfile
  ports:
    - "3002:3002"
  environment:
    - DATABASE_URL=postgresql://user:pass@postgres:5432/appointment_db
    - REDIS_URL=redis://redis:6379
  depends_on:
    - postgres
    - redis

postgres:
  image: postgres:15
  environment:
    - POSTGRES_PASSWORD=password
  volumes:
    - postgres-data:/var/lib/postgresql/data
```

```
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"

volumes:
  postgres-data:
```

Kubernetes Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: patient-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: patient-service
  template:
    metadata:
      labels:
        app: patient-service
```

```
spec:
  containers:
  - name: patient-service
    image: ithalamed/patient-service:latest
    ports:
    - containerPort: 3001
    env:
    - name: DATABASE_URL
      valueFrom:
        secretKeyRef:
          name: db-credentials
          key: url
    resources:
      requests:
        memory: "256Mi"
        cpu: "250m"
      limits:
        memory: "512Mi"
        cpu: "500m"
    livenessProbe:
      httpGet:
        path: /health
        port: 3001
      initialDelaySeconds: 30
      periodSeconds: 10
    readinessProbe:
      httpGet:
        path: /health/ready
```

```
        port: 3001
      initialDelaySeconds: 10
      periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
 name: patient-service
spec:
 selector:
  app: patient-service
 ports:
 - protocol: TCP
  port: 80
  targetPort: 3001
 type: ClusterIP
```

This completes the microservices implementation guide. Would you like me to now create the detailed **Database Schema Design** document?

```
import React, { useState } from 'react';
import { Database, Table, Key, Link, Shield, Zap } from 'lucide-react';

const DatabaseSchema = () => {
 const [selectedTable, setSelectedTable] = useState('patients');

 const tables = {
  patients: {
   name: "patients",
   description: "Core patient demographic and identity information",
```

category: "Core",

columns: [

{ name: "id", type: "UUID", constraints: "PRIMARY KEY", description: "Unique patient identifier" },

{ name: "patient_number", type: "VARCHAR(20)", constraints: "UNIQUE NOT NULL", description: "Human-readable patient number (PT202500001)" },

{ name: "first_name", type: "VARCHAR(100)", constraints: "NOT NULL", description: "Patient first name" },

{ name: "last_name", type: "VARCHAR(100)", constraints: "NOT NULL", description: "Patient last name" },

{ name: "id_number", type: "TEXT", constraints: "UNIQUE NOT NULL", description: "SA ID number (encrypted)" },

{ name: "date_of_birth", type: "DATE", constraints: "NOT NULL", description: "Date of birth" },

{ name: "gender", type: "VARCHAR(20)", constraints: "NOT NULL", description: "Gender: male, female, other" },

{ name: "blood_type", type: "VARCHAR(5)", constraints: "NULL", description: "Blood type: A+, A-, B+, B-, AB+, AB-, O+, O-" },

{ name: "phone_number", type: "VARCHAR(20)", constraints: "UNIQUE NOT NULL", description: "Primary phone number" },

{ name: "email", type: "VARCHAR(255)", constraints: "UNIQUE", description: "Email address" },

{ name: "address", type: "JSONB", constraints: "NOT NULL", description: "Physical address (street, suburb, city, etc.)" },

{ name: "medical_aid", type: "JSONB", constraints: "NULL", description: "Medical aid details (scheme, number, plan)" },

{ name: "emergency_contact", type: "JSONB", constraints: "NOT NULL", description: "Emergency contact information" },

{ name: "allergies", type: "TEXT[]", constraints: "DEFAULT '{}'", description: "List of known allergies" },

{ name: "chronic_conditions", type: "TEXT[]", constraints: "DEFAULT '{}'", description: "List of chronic conditions" },

```
      { name: "status", type: "VARCHAR(20)", constraints: "DEFAULT 'active'", description: "Patient status: active, inactive, deceased" },
      { name: "created_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Record creation timestamp" },
      { name: "updated_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Last update timestamp" },
      { name: "deleted_at", type: "TIMESTAMP", constraints: "NULL", description: "Soft delete timestamp" }
    ],
    indexes: [
      { name: "idx_patients_patient_number", columns: ["patient_number"], type: "UNIQUE" },
      { name: "idx_patients_id_number", columns: ["id_number"], type: "UNIQUE" },
      { name: "idx_patients_phone", columns: ["phone_number"], type: "BTREE" },
      { name: "idx_patients_email", columns: ["email"], type: "BTREE" },
      { name: "idx_patients_created_at", columns: ["created_at"], type: "BTREE" }
    ],
    relationships: [
      { table: "appointments", relationship: "One-to-Many", description: "Patient has many appointments" },
      { table: "medical_records", relationship: "One-to-Many", description: "Patient has many medical records" },
      { table: "prescriptions", relationship: "One-to-Many", description: "Patient has many prescriptions" }
    ]
  },
  providers: {
    name: "providers",
    description: "Healthcare providers (doctors, nurses, specialists)",
    category: "Core",
```

columns: [

{ name: "id", type: "UUID", constraints: "PRIMARY KEY", description: "Unique provider identifier" },

{ name: "provider_number", type: "VARCHAR(20)", constraints: "UNIQUE NOT NULL", description: "Provider number (PRV202500001)" },

{ name: "first_name", type: "VARCHAR(100)", constraints: "NOT NULL", description: "Provider first name" },

{ name: "last_name", type: "VARCHAR(100)", constraints: "NOT NULL", description: "Provider last name" },

{ name: "provider_type", type: "VARCHAR(50)", constraints: "NOT NULL", description: "Type: doctor, nurse, specialist, pharmacist" },

{ name: "specialty", type: "VARCHAR(100)", constraints: "NULL", description: "Medical specialty" },

{ name: "hpcsa_number", type: "VARCHAR(20)", constraints: "UNIQUE", description: "HPCSA registration number" },

{ name: "phone_number", type: "VARCHAR(20)", constraints: "NOT NULL", description: "Contact phone" },

{ name: "email", type: "VARCHAR(255)", constraints: "UNIQUE NOT NULL", description: "Contact email" },

{ name: "qualifications", type: "JSONB", constraints: "DEFAULT '[]'", description: "List of qualifications" },

{ name: "languages", type: "TEXT[]", constraints: "DEFAULT '{}'", description: "Languages spoken" },

{ name: "consultation_fee", type: "DECIMAL(10,2)", constraints: "NULL", description: "Standard consultation fee" },

{ name: "bio", type: "TEXT", constraints: "NULL", description: "Professional biography" },

{ name: "profile_image_url", type: "TEXT", constraints: "NULL", description: "Profile photo URL" },

{ name: "status", type: "VARCHAR(20)", constraints: "DEFAULT 'active'", description: "Status: active, inactive, suspended" },

{ name: "created_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Record creation" },

{ name: "updated_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Last update" }

],

indexes: [

{ name: "idx_providers_hpcsa", columns: ["hpcsa_number"], type: "UNIQUE" },

{ name: "idx_providers_type", columns: ["provider_type"], type: "BTREE" },

{ name: "idx_providers_specialty", columns: ["specialty"], type: "BTREE" }

],

relationships: [

{ table: "provider_facilities", relationship: "Many-to-Many", description: "Provider works at multiple facilities" },

{ table: "appointments", relationship: "One-to-Many", description: "Provider has many appointments" }

]

},

appointments: {

name: "appointments",

description: "Patient appointments and scheduling",

category: "Operations",

columns: [

{ name: "id", type: "UUID", constraints: "PRIMARY KEY", description: "Unique appointment ID" },

{ name: "appointment_number", type: "VARCHAR(20)", constraints: "UNIQUE NOT NULL", description: "Appointment number (APT202500001)" },

{ name: "patient_id", type: "UUID", constraints: "NOT NULL REFERENCES patients(id)", description: "Foreign key to patient" },

{ name: "provider_id", type: "UUID", constraints: "NOT NULL REFERENCES providers(id)", description: "Foreign key to provider" },

{ name: "facility_id", type: "UUID", constraints: "NOT NULL REFERENCES facilities(id)", description: "Foreign key to facility" },

{ name: "appointment_type", type: "VARCHAR(50)", constraints: "NOT NULL", description: "Type: consultation, follow-up, procedure" },

{ name: "scheduled_for", type: "TIMESTAMP", constraints: "NOT NULL", description: "Scheduled date and time" },

{ name: "duration", type: "INTEGER", constraints: "NOT NULL", description: "Duration in minutes" },

{ name: "status", type: "VARCHAR(30)", constraints: "NOT NULL", description: "Status: scheduled, confirmed, checked-in, in-progress, completed, cancelled, no-show" },

{ name: "reason", type: "TEXT", constraints: "NULL", description: "Reason for visit" },

{ name: "notes", type: "TEXT", constraints: "NULL", description: "Additional notes" },

{ name: "confirmation_code", type: "VARCHAR(10)", constraints: "NULL", description: "Confirmation code for check-in" },

{ name: "checked_in_at", type: "TIMESTAMP", constraints: "NULL", description: "Check-in timestamp" },

{ name: "completed_at", type: "TIMESTAMP", constraints: "NULL", description: "Completion timestamp" },

{ name: "cancelled_at", type: "TIMESTAMP", constraints: "NULL", description: "Cancellation timestamp" },

{ name: "cancelled_by", type: "VARCHAR(20)", constraints: "NULL", description: "Who cancelled: patient, provider, system" },

{ name: "cancellation_reason", type: "TEXT", constraints: "NULL", description: "Reason for cancellation" },

{ name: "created_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Record creation" },

{ name: "updated_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Last update" }

],

indexes: [

{ name: "idx_appointments_patient", columns: ["patient_id", "scheduled_for"], type: "BTREE" },

{ name: "idx_appointments_provider", columns: ["provider_id", "scheduled_for"], type: "BTREE" },

{ name: "idx_appointments_facility", columns: ["facility_id", "scheduled_for"], type: "BTREE" },

{ name: "idx_appointments_status", columns: ["status"], type: "BTREE" },

{ name: "idx_appointments_scheduled", columns: ["scheduled_for"], type: "BTREE" }

],

relationships: [

{ table: "patients", relationship: "Many-to-One", description: "Appointment belongs to patient" },

{ table: "providers", relationship: "Many-to-One", description: "Appointment with provider" },

{ table: "medical_records", relationship: "One-to-One", description: "Appointment may have medical record" }

]

},

medical_records: {

name: "medical_records",

description: "Electronic health records and clinical notes",

category: "Clinical",

columns: [

{ name: "id", type: "UUID", constraints: "PRIMARY KEY", description: "Unique record ID" },

{ name: "patient_id", type: "UUID", constraints: "NOT NULL REFERENCES patients(id)", description: "Foreign key to patient" },

{ name: "provider_id", type: "UUID", constraints: "NOT NULL REFERENCES providers(id)", description: "Foreign key to provider" },

{ name: "appointment_id", type: "UUID", constraints: "REFERENCES appointments(id)", description: "Related appointment (if applicable)" },

{ name: "encounter_date", type: "TIMESTAMP", constraints: "NOT NULL", description: "Date of encounter" },

{ name: "record_type", type: "VARCHAR(50)", constraints: "NOT NULL", description: "Type: consultation, procedure, lab, imaging" },

{ name: "chief_complaint", type: "TEXT", constraints: "NULL", description: "Primary reason for visit" },

{ name: "history_present_illness", type: "TEXT", constraints: "NULL", description: "HPI in SOAP format" },

{ name: "physical_examination", type: "TEXT", constraints: "NULL", description: "Physical exam findings" },

{ name: "diagnosis", type: "JSONB", constraints: "NULL", description: "Diagnoses with ICD-10 codes" },

{ name: "assessment", type: "TEXT", constraints: "NULL", description: "Clinical assessment" },

{ name: "plan", type: "TEXT", constraints: "NULL", description: "Treatment plan" },

{ name: "vital_signs", type: "JSONB", constraints: "NULL", description: "BP, HR, temp, etc." },

{ name: "clinical_notes", type: "TEXT", constraints: "NULL", description: "Additional clinical notes" },

{ name: "documents", type: "JSONB", constraints: "DEFAULT '[]'", description: "Attached documents (URLs)" },

{ name: "created_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Record creation" },

{ name: "updated_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Last update" }

],

indexes: [

{ name: "idx_medical_records_patient", columns: ["patient_id", "encounter_date"], type: "BTREE" },

{ name: "idx_medical_records_provider", columns: ["provider_id"], type: "BTREE" },

{ name: "idx_medical_records_date", columns: ["encounter_date"], type: "BTREE" }

],

relationships: [

{ table: "patients", relationship: "Many-to-One", description: "Record belongs to patient" },

{ table: "prescriptions", relationship: "One-to-Many", description: "Record may have prescriptions" }

]

},

prescriptions: {

name: "prescriptions",

description: "Electronic prescriptions",

category: "Clinical",

columns: [

{ name: "id", type: "UUID", constraints: "PRIMARY KEY", description: "Unique prescription ID" },

{ name: "prescription_number", type: "VARCHAR(20)", constraints: "UNIQUE NOT NULL", description: "Prescription number (RX202500001)" },

{ name: "patient_id", type: "UUID", constraints: "NOT NULL REFERENCES patients(id)", description: "Foreign key to patient" },

{ name: "provider_id", type: "UUID", constraints: "NOT NULL REFERENCES providers(id)", description: "Foreign key to provider" },

{ name: "encounter_id", type: "UUID", constraints: "REFERENCES medical_records(id)", description: "Related medical record" },

{ name: "medications", type: "JSONB", constraints: "NOT NULL", description: "Array of medications with details" },

{ name: "diagnosis", type: "TEXT", constraints: "NULL", description: "Diagnosis for prescription" },

{ name: "notes", type: "TEXT", constraints: "NULL", description: "Additional instructions" },

{ name: "status", type: "VARCHAR(30)", constraints: "NOT NULL", description: "Status: active, dispensed, partially-dispensed, expired, cancelled" },

{ name: "issued_at", type: "TIMESTAMP", constraints: "NOT NULL", description: "Issue date" },

{ name: "expires_at", type: "TIMESTAMP", constraints: "NOT NULL", description: "Expiry date (typically 6 months)" },

{ name: "e_signature", type: "TEXT", constraints: "NOT NULL", description: "Digital signature of provider" },

{ name: "qr_code", type: "TEXT", constraints: "NOT NULL", description: "QR code for verification" },

{ name: "preferred_pharmacy_id", type: "UUID", constraints: "REFERENCES facilities(id)", description: "Preferred pharmacy" },

{ name: "dispensed_count", type: "INTEGER", constraints: "DEFAULT 0", description: "Number of times dispensed" },

{ name: "created_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Record creation" },

{ name: "updated_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Last update" }

],

indexes: [

{ name: "idx_prescriptions_patient", columns: ["patient_id", "issued_at"], type: "BTREE" },

{ name: "idx_prescriptions_provider", columns: ["provider_id"], type: "BTREE" },

{ name: "idx_prescriptions_status", columns: ["status"], type: "BTREE" },

{ name: "idx_prescriptions_expires", columns: ["expires_at"], type: "BTREE" }

],

relationships: [

{ table: "prescription_dispensings", relationship: "One-to-Many", description: "Prescription has dispensing records" }

]

},

facilities: {

name: "facilities",

description: "Healthcare facilities (hospitals, clinics, pharmacies)",

category: "Core",

columns: [

{ name: "id", type: "UUID", constraints: "PRIMARY KEY", description: "Unique facility ID" },

{ name: "facility_number", type: "VARCHAR(20)", constraints: "UNIQUE NOT NULL", description: "Facility number" },

{ name: "name", type: "VARCHAR(200)", constraints: "NOT NULL", description: "Facility name" },

{ name: "facility_type", type: "VARCHAR(50)", constraints: "NOT NULL", description: "Type: hospital, clinic, pharmacy, lab" },

{ name: "address", type: "JSONB", constraints: "NOT NULL", description: "Physical address" },

{ name: "coordinates", type: "POINT", constraints: "NULL", description: "GPS coordinates for mapping" },

{ name: "phone_number", type: "VARCHAR(20)", constraints: "NOT NULL", description: "Contact phone" },

{ name: "email", type: "VARCHAR(255)", constraints: "NULL", description: "Contact email" },

{ name: "operating_hours", type: "JSONB", constraints: "NULL", description: "Operating hours by day" },

{ name: "services", type: "TEXT[]", constraints: "DEFAULT '{}'", description: "Services offered" },

{ name: "accreditation", type: "JSONB", constraints: "NULL", description: "Accreditations and certifications" },

{ name: "status", type: "VARCHAR(20)", constraints: "DEFAULT 'active'", description: "Status: active, inactive" },

{ name: "created_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Record creation" },

{ name: "updated_at", type: "TIMESTAMP", constraints: "DEFAULT NOW()", description: "Last update" }

],

indexes: [

{ name: "idx_facilities_type", columns: ["facility_type"], type: "BTREE" },

{ name: "idx_facilities_coordinates", columns: ["coordinates"], type: "GIST" }

```
    ],
    relationships: [
     { table: "provider_facilities", relationship: "Many-to-Many", description: "Facility has many providers" },
     { table: "appointments", relationship: "One-to-Many", description: "Facility has many appointments" }
    ]
   }
 };

 const categories = {
  Core: ["patients", "providers", "facilities"],
  Operations: ["appointments"],
  Clinical: ["medical_records", "prescriptions"]
 };

 const currentTable = tables[selectedTable];

 return (
  <div className="w-full min-h-screen bg-gradient-to-br from-slate-900 via-slate-800 to-slate-900 p-8">
   <div className="max-w-7xl mx-auto">
    <div className="text-center mb-8">
     <h1 className="text-4xl font-bold text-white mb-2">IthalaMed Database Schema</h1>
     <p className="text-slate-300">PostgreSQL Schema Design for Healthcare Platform</p>
    </div>
```

```
{/* Database Overview */}
<div className="grid grid-cols-1 md:grid-cols-4 gap-4 mb-8">
 <div className="bg-slate-800 border border-slate-700 rounded-lg p-6">
  <div className="flex items-center gap-2 mb-2">
   <Database className="w-5 h-5 text-blue-400" />
   <span className="text-slate-400 text-sm">Primary Database</span>
  </div>
  <div className="text-2xl font-bold text-white">PostgreSQL 15</div>
  <div className="text-xs text-slate-400 mt-1">Transactional data</div>
 </div>

 <div className="bg-slate-800 border border-slate-700 rounded-lg p-6">
  <div className="flex items-center gap-2 mb-2">
   <Table className="w-5 h-5 text-green-400" />
   <span className="text-slate-400 text-sm">Total Tables</span>
  </div>
  <div className="text-2xl font-bold text-white">25+</div>
  <div className="text-xs text-slate-400 mt-1">Core + supporting</div>
 </div>

 <div className="bg-slate-800 border border-slate-700 rounded-lg p-6">
  <div className="flex items-center gap-2 mb-2">
   <Shield className="w-5 h-5 text-purple-400" />
   <span className="text-slate-400 text-sm">Security</span>
  </div>
  <div className="text-2xl font-bold text-white">AES-256</div>
  <div className="text-xs text-slate-400 mt-1">Encrypted PII</div>
 </div>
```

```
        <div className="bg-slate-800 border border-slate-700 rounded-lg p-6">
          <div className="flex items-center gap-2 mb-2">
            <Zap className="w-5 h-5 text-yellow-400" />
            <span className="text-slate-400 text-sm">Performance</span>
          </div>
          <div className="text-2xl font-bold text-white">Optimized</div>
          <div className="text-xs text-slate-400 mt-1">Indexed queries</div>
        </div>
      </div>

      {/* Table Categories */}
      <div className="mb-8">
        <h2 className="text-2xl font-bold text-white mb-4">Schema Categories</h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          {Object.entries(categories).map(([category, tableList]) => (
            <div key={category} className="bg-slate-800 border border-slate-700 rounded-
lg p-4">
              <h3 className="font-bold text-white mb-3">{category} Tables</h3>
              <div className="space-y-2">
                {tableList.map((tableName) => (
                  <button
                    key={tableName}
                    onClick={() => setSelectedTable(tableName)}
                    className={` w-full text-left px-3 py-2 rounded transition-colors ${
                      selectedTable === tableName
                        ? 'bg-blue-500 text-white'
                        : 'bg-slate-700 text-slate-300 hover:bg-slate-600'
```

```
            }`}
           >
            <div className="flex items-center gap-2">
             <Table className="w-4 h-4" />
             <span className="text-sm">{tableName}</span>
            </div>
           </button>
          ))}
         </div>
        </div>
       ))}
      </div>
     </div>

     {/* Selected Table Details */}
     <div className="bg-slate-800 border border-slate-700 rounded-lg p-6 mb-8">
      <div className="flex items-center justify-between mb-6">
       <div>
        <h2 className="text-3xl font-bold text-white mb-2">{currentTable.name}</h2>
        <p className="text-slate-300">{currentTable.description}</p>
        <span className="inline-block mt-2 px-3 py-1 bg-blue-500 bg-opacity-20 text-
blue-300 rounded-full text-sm">
         {currentTable.category}
        </span>
       </div>
      </div>

      {/* Columns */}
```

```
<div className="mb-6">
 <h3 className="text-xl font-bold text-white mb-4 flex items-center gap-2">
  <Key className="w-5 h-5 text-yellow-400" />
  Columns
 </h3>
 <div className="overflow-x-auto">
  <table className="w-full text-sm">
   <thead>
    <tr className="border-b border-slate-700">
     <th className="text-left py-3 px-2 text-slate-400 font-semibold">Column Name</th>
     <th className="text-left py-3 px-2 text-slate-400 font-semibold">Data Type</th>
     <th className="text-left py-3 px-2 text-slate-400 font-semibold">Constraints</th>
     <th className="text-left py-3 px-2 text-slate-400 font-semibold">Description</th>
    </tr>
   </thead>
   <tbody>
    {currentTable.columns.map((col, idx) => (
     <tr key={idx} className="border-b border-slate-700 border-opacity-50 hover:bg-slate-700 hover:bg-opacity-30">
      <td className="py-3 px-2">
       <code className="text-blue-300 font-mono text-xs">{col.name}</code>
      </td>
      <td className="py-3 px-2">
       <span className="text-green-300 font-mono text-xs">{col.type}</span>
      </td>
```

```
          <td className="py-3 px-2">
            <span className="text-purple-300 text-xs">{col.constraints}</span>
          </td>
          <td className="py-3 px-2 text-slate-300 text-xs">{col.description}</td>
        </tr>
      ))}
    </tbody>
  </table>
 </div>
</div>

{/* Indexes */}
<div className="mb-6">
 <h3 className="text-xl font-bold text-white mb-4 flex items-center gap-2">
  <Zap className="w-5 h-5 text-yellow-400" />
  Indexes
 </h3>
 <div className="grid grid-cols-1 md:grid-cols-2 gap-3">
  {currentTable.indexes.map((index, idx) => (
   <div key={idx} className="bg-slate-700 bg-opacity-50 rounded p-3">
    <div className="font-mono text-xs text-blue-300 mb-1">{index.name}</div>
    <div className="text-xs text-slate-300">
     Columns: <span className="text-white">{index.columns.join(', ')}</span>
    </div>
    <div className="text-xs text-slate-400">Type: {index.type}</div>
   </div>
  ))}
 </div>
```

```
        </div>

        {/* Relationships */}
        <div>
          <h3 className="text-xl font-bold text-white mb-4 flex items-center gap-2">
            <Link className="w-5 h-5 text-green-400" />
            Relationships
          </h3>
          <div className="space-y-2">
            {currentTable.relationships.map((rel, idx) => (
              <div key={idx} className="bg-slate-700 bg-opacity-50 rounded p-3 flex items-center justify-between">
                <div>
                  <span className="text-white font-semibold">{rel.table}</span>
                  <p className="text-sm text-slate-300">{rel.description}</p>
                </div>
                <span className="px-3 py-1 bg-green-500 bg-opacity-20 text-green-300 rounded text-xs">
                  {rel.relationship}
                </span>
              </div>
            ))}
          </div>
        </div>
      </div>

      {/* Database Best Practices */}
      <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
        <div className="bg-slate-800 border border-slate-700 rounded-lg p-6">
```

```
        <h3 className="text-lg font-bold text-white mb-4">Design Principles</h3>
        <ul className="space-y-2 text-sm text-slate-300">
          <li>• <strong>Normalization:</strong> 3NF where appropriate</li>
          <li>• <strong>UUIDs:</strong> Primary keys for distributed systems</li>
          <li>• <strong>JSONB:</strong> Flexible data (addresses, metadata)</li>
          <li>• <strong>Timestamps:</strong> created_at, updated_at on all tables</li>
          <li>• <strong>Soft Deletes:</strong> deleted_at for audit trail</li>
          <li>• <strong>Foreign Keys:</strong> Referential integrity enforced</li>
        </ul>
      </div>

      <div className="bg-slate-800 border border-slate-700 rounded-lg p-6">
        <h3 className="text-lg font-bold text-white mb-4">Performance
Optimization</h3>
        <ul className="space-y-2 text-sm text-slate-300">
          <li>• <strong>Indexes:</strong> Strategic indexing on query columns</li>
          <li>• <strong>Partitioning:</strong> Time-based for large tables</li>
          <li>• <strong>Connection Pooling:</strong> pgBouncer (min: 5, max: 50)</li>
          <li>• <strong>Query Optimization:</strong> EXPLAIN ANALYZE for tuning</li>
          <li>• <strong>Caching:</strong> Redis for hot data (5-min TTL)</li>
          <li>• <strong>Read Replicas:</strong> 3 replicas for read scaling</li>
        </ul>
      </div>
    </div>
  </div>
 );
};
```

export default DatabaseSchema;

IthalaMed API Specifications

Complete OpenAPI 3.0 Documentation

API OVERVIEW

Base Information

openapi: 3.0.3

info:

title: IthalaMed API

description: |

Unified Healthcare Ecosystem API for Africa

Overview

The IthalaMed API provides comprehensive access to our healthcare platform,

enabling patients, providers, facilities, and other stakeholders to interact

with the system programmatically.

Authentication

All API requests require authentication using JWT Bearer tokens.

Obtain a token via the `/auth/login` endpoint.

Rate Limiting

- Authenticated requests: 1000 requests/hour

- Unauthenticated requests: 100 requests/hour

- Rate limit headers included in all responses

Versioning

API version is specified in the URL path (e.g., `/api/v1/`)

Current version: v1

version: 1.0.0

contact:

name: IthalaMed API Support

email: api-support@ithalamed.com

url: https://docs.ithalamed.com

license:

name: Proprietary

url: https://ithalamed.com/license

servers:

- url: https://api.ithalamed.com/v1

description: Production server

- url: https://api-staging.ithalamed.com/v1

description: Staging server

- url: http://localhost:3000/v1

description: Local development server

tags:

- name: Authentication

description: User authentication and authorization

- name: Patients

description: Patient management and profiles

- name: Providers

description: Healthcare provider management

- name: Appointments

```
    description: Appointment scheduling and management
  - name: Medical Records
    description: Electronic health records
  - name: Prescriptions
    description: E-prescriptions and medication management
  - name: Telemedicine
    description: Virtual consultations
  - name: Facilities
    description: Healthcare facility management
  - name: Insurance
    description: Medical aid and claims
```

AUTHENTICATION ENDPOINTS

POST /auth/register

Register a new user account.

Request Body:

```
{
  "userType": "patient",
  "email": "thandi@example.com",
  "password": "SecureP@ssw0rd123",
  "phoneNumber": "+27821234567",
  "firstName": "Thandi",
  "lastName": "Mbatha",
  "dateOfBirth": "1985-01-15",
  "idNumber": "8501156789012"
}
```

Response (201 Created):

```
{
```

```
 "userId": "550e8400-e29b-41d4-a716-446655440000",
 "email": "thandi@example.com",
 "userType": "patient",
 "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
 "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
 "expiresIn": 3600,
 "tokenType": "Bearer"
}
```


OpenAPI Spec:


```
/auth/register:
  post:
    tags:
      - Authentication
    summary: Register new user
    description: Create a new user account (patient, provider, or facility admin)
    operationId: registerUser
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required:
              - userType
              - email
              - password
              - phoneNumber
              - firstName
```

```
    - lastName
  properties:
    userType:
      type: string
      enum: [patient, provider, facility_admin, pharmacist]
      example: patient
    email:
      type: string
      format: email
      example: thandi@example.com
    password:
      type: string
      format: password
      minLength: 8
      pattern: ^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]
      example: SecureP@ssw0rd123
    phoneNumber:
      type: string
      pattern: ^\+27[0-9]{9}$
      example: "+27821234567"
    firstName:
      type: string
      minLength: 2
      maxLength: 100
      example: Thandi
    lastName:
      type: string
      minLength: 2
```

```
          maxLength: 100
          example: Mbatha
        dateOfBirth:
          type: string
          format: date
          example: "1985-01-15"
        idNumber:
          type: string
          pattern: ^\d{13}$
          example: "8501156789012"
responses:
  '201':
    description: User successfully registered
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/AuthResponse'
  '400':
    description: Invalid request data
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
  '409':
    description: User already exists
    content:
      application/json:
        schema:
```

```
          $ref: '#/components/schemas/Error'
    '429':
      description: Rate limit exceeded
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
```

POST /auth/login

Authenticate user and obtain access token.

Request Body:


```
{
  "email": "thandi@example.com",
  "password": "SecureP@ssw0rd123",
  "deviceInfo": {
    "deviceId": "device-123",
    "deviceType": "mobile",
    "platform": "ios",
    "appVersion": "1.0.0"
  }
}
```


Response (200 OK):


```
{
  "userId": "550e8400-e29b-41d4-a716-446655440000",
  "email": "thandi@example.com",
  "userType": "patient",
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "expiresIn": 3600,
```

"tokenType": "Bearer",

"mfaRequired": false

}

POST /auth/refresh

Refresh access token using refresh token.

Request Body:

{

"refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}

Response (200 OK):

{

"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expiresIn": 3600,

"tokenType": "Bearer"

}

POST /auth/logout

Invalidate current session and tokens.

Request Headers:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response (204 No Content)

PATIENT ENDPOINTS

GET /patients/{patientId}

Retrieve patient profile.

Path Parameters:

- patientId (UUID, required): Patient unique identifier

Request Headers:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response (200 OK):


```
{
 "id": "550e8400-e29b-41d4-a716-446655440000",
 "patientNumber": "PT202500001",
 "firstName": "Thandi",
 "lastName": "Mbatha",
 "dateOfBirth": "1985-01-15",
 "gender": "female",
 "bloodType": "A+",
 "phoneNumber": "+27821234567",
 "email": "thandi@example.com",
 "address": {
  "street": "123 Main Road",
  "suburb": "Soweto",
  "city": "Johannesburg",
  "province": "Gauteng",
  "postalCode": "1809",
  "country": "ZA"
 },
 "medicalAid": {
  "scheme": "discovery",
  "membershipNumber": "12345678",
  "dependentCode": "00",
  "plan": "essential",
  "verified": true
 },
 "allergies": ["penicillin", "peanuts"],
 "chronicConditions": ["hypertension"],
```

```
 "emergencyContact": {
  "name": "John Mbatha",
  "relationship": "spouse",
  "phoneNumber": "+27829876543"
 },
 "lastVisit": "2024-12-10T14:00:00Z",
 "status": "active",
 "createdAt": "2024-01-15T10:00:00Z",
 "updatedAt": "2025-01-18T15:30:00Z"
}
```


OpenAPI Spec:


```
/patients/{patientId}:
 get:
  tags:
   - Patients
  summary: Get patient profile
  description: Retrieve complete patient profile including demographics and medical information
  operationId: getPatientById
  security:
   - BearerAuth: []
  parameters:
   - name: patientId
     in: path
     required: true
     schema:
      type: string
      format: uuid
```

```
      description: Patient unique identifier
  responses:
    '200':
      description: Patient profile retrieved successfully
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Patient'
    '401':
      $ref: '#/components/responses/UnauthorizedError'
    '403':
      $ref: '#/components/responses/ForbiddenError'
    '404':
      $ref: '#/components/responses/NotFoundError'
```

PATCH /patients/{patientId}

Update patient profile.

Request Body:

```
{
  "phoneNumber": "+27821234568",
  "email": "thandi.new@example.com",
  "address": {
    "street": "456 New Street",
    "suburb": "Soweto",
    "city": "Johannesburg",
    "province": "Gauteng",
    "postalCode": "1809",
    "country": "ZA"
  }
```

}

Response (200 OK):

{

"id": "550e8400-e29b-41d4-a716-446655440000",

"updatedAt": "2025-01-20T10:15:00Z",

"message": "Patient profile updated successfully"

}

GET /patients/{patientId}/medical-history

Retrieve patient's medical history.

Query Parameters:

- from (date, optional): Start date for history range
- to (date, optional): End date for history range
- limit (integer, optional, default: 20): Number of records per page
- offset (integer, optional, default: 0): Pagination offset

Response (200 OK):

{

"patientId": "550e8400-e29b-41d4-a716-446655440000",

"records": [

{

"id": "record-uuid-1",

"encounterDate": "2024-12-10T14:00:00Z",

"recordType": "consultation",

"provider": {

"id": "provider-uuid",

"name": "Dr. Sarah Mokoena",

"specialty": "General Practice"

},

"facility": {

```
      "id": "facility-uuid",
      "name": "Soweto Medical Centre"
    },
    "chiefComplaint": "Persistent headache",
    "diagnosis": [
     {
       "code": "R51",
       "description": "Headache",
       "type": "ICD-10"
     }
    ],
    "prescriptions": [
     {
       "id": "prescription-uuid",
       "prescriptionNumber": "RX202400145"
     }
    ]
  }
 ],
 "pagination": {
  "total": 25,
  "limit": 20,
  "offset": 0,
  "hasMore": true
 }
}
```

APPOINTMENT ENDPOINTS

GET /appointments/availability

Check provider availability for appointments.

Query Parameters:

- providerId (UUID, required): Provider identifier
- date (date, required): Date to check (YYYY-MM-DD)
- facilityId (UUID, optional): Specific facility

Response (200 OK):


```
{
 "providerId": "550e8400-e29b-41d4-a716-446655440001",
 "date": "2025-01-25",
 "facilityId": "550e8400-e29b-41d4-a716-446655440002",
 "availableSlots": [
  {
   "startTime": "09:00",
   "endTime": "09:30",
   "available": true,
   "appointmentType": "consultation"
  },
  {
   "startTime": "09:30",
   "endTime": "10:00",
   "available": false,
   "appointmentType": "consultation"
  },
  {
   "startTime": "10:00",
   "endTime": "10:30",
   "available": true,
```

```
    "appointmentType": "consultation"
   }
 ],
 "workingHours": {
  "start": "09:00",
  "end": "17:00"
 },
 "lunchBreak": {
  "start": "13:00",
  "end": "14:00"
 }
}
```


POST /appointments

Book a new appointment.

Request Body:


```
{
 "patientId": "550e8400-e29b-41d4-a716-446655440000",
 "providerId": "550e8400-e29b-41d4-a716-446655440001",
 "facilityId": "550e8400-e29b-41d4-a716-446655440002",
 "appointmentType": "consultation",
 "date": "2025-01-25",
 "startTime": "09:00",
 "duration": 30,
 "reason": "Follow-up for hypertension",
 "notes": "Please bring previous test results"
}
```


Response (201 Created):


```
{
```

```
 "id": "appointment-uuid",
 "appointmentNumber": "APT202500125",
 "patientId": "550e8400-e29b-41d4-a716-446655440000",
 "providerId": "550e8400-e29b-41d4-a716-446655440001",
 "facilityId": "550e8400-e29b-41d4-a716-446655440002",
 "appointmentType": "consultation",
 "scheduledFor": "2025-01-25T09:00:00Z",
 "duration": 30,
 "status": "scheduled",
 "confirmationCode": "ABC123",
 "provider": {
  "name": "Dr. Sarah Mokoena",
  "specialty": "General Practice"
 },
 "facility": {
  "name": "Soweto Medical Centre",
  "address": "123 Medical Street, Soweto"
 },
 "createdAt": "2025-01-20T11:00:00Z"
}
```


GET /appointments

List appointments with filtering.

Query Parameters:

- patientId (UUID, optional): Filter by patient
- providerId (UUID, optional): Filter by provider
- facilityId (UUID, optional): Filter by facility
- status (string, optional): Filter by status (scheduled, completed, cancelled, no-show)

- from (date, optional): Start date
- to (date, optional): End date
- limit (integer, optional, default: 20)
- offset (integer, optional, default: 0)

Response (200 OK):


```
{
  "appointments": [
    {
      "id": "appointment-uuid",
      "appointmentNumber": "APT202500125",
      "patient": {
        "id": "patient-uuid",
        "name": "Thandi Mbatha"
      },
      "provider": {
        "id": "provider-uuid",
        "name": "Dr. Sarah Mokoena"
      },
      "facility": {
        "id": "facility-uuid",
        "name": "Soweto Medical Centre"
      },
      "scheduledFor": "2025-01-25T09:00:00Z",
      "duration": 30,
      "status": "scheduled",
      "appointmentType": "consultation"
    }
  ],
```

```
  "pagination": {
    "total": 15,
    "limit": 20,
    "offset": 0,
    "hasMore": false
  }
}
```


PATCH /appointments/{appointmentId}

Update appointment details.

Request Body:


```
{
  "status": "confirmed",
  "notes": "Patient confirmed attendance"
}
```


DELETE /appointments/{appointmentId}

Cancel an appointment.

Request Body:


```
{
  "reason": "Schedule conflict",
  "cancelledBy": "patient"
}
```


Response (200 OK):


```
{
  "id": "appointment-uuid",
  "status": "cancelled",
  "cancelledAt": "2025-01-20T12:00:00Z",
  "cancelledBy": "patient",
  "cancellationReason": "Schedule conflict"
```

}

PRESCRIPTION ENDPOINTS

POST /prescriptions

Create a new prescription.

Request Body:

```
{
  "patientId": "patient-uuid",
  "providerId": "provider-uuid",
  "encounterId": "encounter-uuid",
  "medications": [
    {
      "drugName": "Amlodipine",
      "drugCode": "NAPPI123456",
      "dosage": "5mg",
      "frequency": "once daily",
      "duration": 30,
      "durationUnit": "days",
      "quantity": 30,
      "instructions": "Take in the morning with food",
      "refills": 2,
      "isControlled": false
    },
    {
      "drugName": "Atorvastatin",
      "drugCode": "NAPPI789012",
      "dosage": "20mg",
      "frequency": "once daily at night",
```

```
    "duration": 30,
    "durationUnit": "days",
    "quantity": 30,
    "instructions": "Take at bedtime",
    "refills": 2,
    "isControlled": false
   }
 ],
 "diagnosis": "Hypertension and hyperlipidemia",
 "notes": "Monitor blood pressure and lipid levels",
 "preferredPharmacy": "pharmacy-uuid"
}
```


Response (201 Created):


```
{
 "id": "prescription-uuid",
 "prescriptionNumber": "RX202500089",
 "patientId": "patient-uuid",
 "providerId": "provider-uuid",
 "medications": [...],
 "status": "active",
 "issuedAt": "2025-01-20T14:00:00Z",
 "expiresAt": "2025-07-20T00:00:00Z",
 "eSignature": "digital-signature-hash",
 "qrCode": "...",
 "preferredPharmacy": {
  "id": "pharmacy-uuid",
  "name": "Clicks Soweto"
 }
```

}

GET /prescriptions/{prescriptionId}

Retrieve prescription details.

Response (200 OK):


```
{
 "id": "prescription-uuid",
 "prescriptionNumber": "RX202500089",
 "patient": {
  "id": "patient-uuid",
  "name": "Thandi Mbatha",
  "dateOfBirth": "1985-01-15"
 },
 "provider": {
  "id": "provider-uuid",
  "name": "Dr. Sarah Mokoena",
  "hpcsaNumber": "MP123456"
 },
 "medications": [
  {
   "drugName": "Amlodipine",
   "drugCode": "NAPPI123456",
   "dosage": "5mg",
   "frequency": "once daily",
   "duration": 30,
   "durationUnit": "days",
   "quantity": 30,
   "instructions": "Take in the morning with food",
   "refills": 2,
```

```
    "refillsRemaining": 2
   }
 ],
 "diagnosis": "Hypertension and hyperlipidemia",
 "status": "active",
 "issuedAt": "2025-01-20T14:00:00Z",
 "expiresAt": "2025-07-20T00:00:00Z",
 "dispensedCount": 0,
 "qrCode": "..."
}
```


POST /prescriptions/{prescriptionId}/dispense

Record prescription dispensing (Pharmacy use).

Request Body:


```
{
 "pharmacyId": "pharmacy-uuid",
 "pharmacistId": "pharmacist-uuid",
 "medications": [
  {
   "medicationId": "medication-uuid-from-prescription",
   "quantityDispensed": 30,
   "batchNumber": "BATCH202501",
   "expiryDate": "2026-12-31",
   "manufacturer": "Pharma Co"
  }
 ],
 "notes": "Patient counseled on medication use"
}
```


Response (200 OK):

```
{
  "dispensingId": "dispensing-uuid",
  "prescriptionId": "prescription-uuid",
  "pharmacyId": "pharmacy-uuid",
  "pharmacistId": "pharmacist-uuid",
  "dispensedAt": "2025-01-21T10:00:00Z",
  "medications": [...],
  "prescriptionStatus": "partially-dispensed"
}
```

TELEMEDICINE ENDPOINTS

POST /telemedicine/sessions

Create a telemedicine session.

Request Body:


```
{
  "appointmentId": "appointment-uuid",
  "patientId": "patient-uuid",
  "providerId": "provider-uuid",
  "sessionType": "video",
  "recordSession": true
}
```


Response (201 Created):


```
{
  "sessionId": "session-uuid",
  "channelName": "consultation-202501-125",
  "sessionToken": "session-jwt-token",
  "agoraAppId": "agora-app-id",
  "agoraToken": "agora-rtc-token",
```

```
 "status": "waiting",
 "expiresAt": "2025-01-20T16:00:00Z",
 "createdAt": "2025-01-20T15:00:00Z"
}
```


POST /telemedicine/sessions/{sessionId}/join

Join a telemedicine session.

Request Body:


```
{
 "userId": "user-uuid",
 "role": "provider"
}
```


Response (200 OK):


```
{
 "sessionId": "session-uuid",
 "channelName": "consultation-202501-125",
 "sessionToken": "session-jwt-token",
 "rtcToken": "agora-rtc-token",
 "participantId": "participant-uuid",
 "status": "active"
}
```


POST /telemedicine/sessions/{sessionId}/end

End a telemedicine session.

Request Body:


```
{
 "duration": 1800,
 "notes": "Patient reports improvement in symptoms",
 "followUpRequired": false,
 "prescriptionIssued": true
```

}

Response (200 OK):

{
  "sessionId": "session-uuid",
  "status": "completed",
  "duration": 1800,
  "startedAt": "2025-01-20T15:05:00Z",
  "endedAt": "2025-01-20T15:35:00Z",
  "recordingUrl": "https://storage.ithalamed.com/recordings/session-uuid.mp4",
  "notes": "Patient reports improvement in symptoms"
}

COMMON SCHEMAS

AuthResponse Schema

```
components:
  schemas:
    AuthResponse:
      type: object
      properties:
        userId:
          type: string
          format: uuid
        email:
          type: string
          format: email
        userType:
          type: string
          enum: [patient, provider, facility_admin, pharmacist]
```

```
    accessToken:
      type: string
      description: JWT access token
    refreshToken:
      type: string
      description: JWT refresh token
    expiresIn:
      type: integer
      description: Access token expiration in seconds
    tokenType:
      type: string
      enum: [Bearer]
    mfaRequired:
      type: boolean
      description: Whether MFA is required
```

Error Schema

```
  Error:
    type: object
    required:
      - statusCode
      - message
      - timestamp
      - path
    properties:
      statusCode:
        type: integer
        example: 400
      message:
```

```
      type: string
      example: "Invalid request parameters"
    errors:
      type: array
      items:
        type: object
        properties:
          field:
            type: string
          message:
            type: string
    timestamp:
      type: string
      format: date-time
    path:
      type: string
      example: "/api/v1/patients"
    requestId:
      type: string
      format: uuid
```

Patient Schema

```
  Patient:
    type: object
    required:
      - id
      - patientNumber
      - firstName
      - lastName
```

```
    - dateOfBirth
    - gender
  properties:
    id:
      type: string
      format: uuid
    patientNumber:
      type: string
      pattern: ^PT\d{9}$
    firstName:
      type: string
    lastName:
      type: string
    dateOfBirth:
      type: string
      format: date
    gender:
      type: string
      enum: [male, female, other]
    bloodType:
      type: string
      enum: [A+, A-, B+, B-, AB+, AB-, O+, O-]
    phoneNumber:
      type: string
    email:
      type: string
      format: email
    address:
```

```
      $ref: '#/components/schemas/Address'
    medicalAid:
      $ref: '#/components/schemas/MedicalAid'
    allergies:
      type: array
      items:
        type: string
    chronicConditions:
      type: array
      items:
        type: string
    status:
      type: string
      enum: [active, inactive, deceased]
    createdAt:
      type: string
      format: date-time
    updatedAt:
      type: string
      format: date-time
```

SECURITY SCHEMES

```
components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

description: |

JWT Bearer token authentication

To authenticate:

1. Obtain token via /auth/login

2. Include in Authorization header: `Bearer {token}`

Token includes:

- userId

- userType

- permissions

- exp (expiration)

ApiKeyAuth:

type: apiKey

in: header

name: X-API-Key

description: |

API Key authentication for system-to-system integration

Contact api-support@ithalamed.com for API key

RESPONSE CODES

Success Codes

Code	Description	Usage
200	OK	Successful GET, PATCH, DELETE
201	Created	Successful POST (resource created)

Code	Description	Usage
204	No Content	Successful DELETE (no response body)

Client Error Codes

Code	Description	Usage
400	Bad Request	Invalid request data
401	Unauthorized	Missing or invalid authentication
403	Forbidden	Authenticated but insufficient permissions
404	Not Found	Resource does not exist
409	Conflict	Resource conflict (duplicate)
422	Unprocessable Entity	Validation errors
429	Too Many Requests	Rate limit exceeded

Server Error Codes

Code	Description	Usage
500	Internal Server Error	Unexpected server error
502	Bad Gateway	Upstream service error
503	Service Unavailable	Service temporarily down
504	Gateway Timeout	Upstream service timeout

PAGINATION

All list endpoints support cursor-based pagination:

Query Parameters:

- limit: Number of items (default: 20, max: 100)
- offset: Number of items to skip (default: 0)

Response Format:

```
{
  "data": [...],
```

```
  "pagination": {
    "total": 150,
    "limit": 20,
    "offset": 0,
    "hasMore": true,
    "nextOffset": 20
  }
}
```

FILTERING & SORTING

Filtering:

GET /appointments?status=scheduled&from=2025-01-01&to=2025-01-31

Sorting:

GET /patients?sort=createdAt&order=desc

Search:

GET /patients/search?q=Mbatha&limit=10

RATE LIMITING

Headers in Response:

X-RateLimit-Limit: 1000

X-RateLimit-Remaining: 995

X-RateLimit-Reset: 1642684800

When Limit Exceeded (429):


```
{
  "statusCode": 429,
  "message": "Rate limit exceeded",
  "retryAfter": 3600,
  "timestamp": "2025-01-20T15:00:00Z"
```

}

WEBHOOKS

IthalaMed supports webhooks for real-time event notifications.

Supported Events

- appointment.created
- appointment.updated
- appointment.cancelled
- prescription.created
- prescription.dispensed
- patient.registered
- claim.submitted
- claim.approved

Webhook Payload Format

```
{
  "eventId": "event-uuid",
  "eventType": "appointment.created",
  "timestamp": "2025-01-20T15:00:00Z",
  "data": {
    "appointmentId": "appointment-uuid",
    "patientId": "patient-uuid",
    "providerId": "provider-uuid",
    "scheduledFor": "2025-01-25T09:00:00Z"
  },
  "webhookId": "webhook-uuid",
  "attempt": 1
}
```

Webhook Security

All webhook requests include:

- X-IthalaMed-Signature: HMAC-SHA256 signature
- X-IthalaMed-Event: Event type
- X-IthalaMed-Delivery: Unique delivery ID

Signature Verification (Node.js):

```
const crypto = require('crypto');


function verifyWebhookSignature(payload, signature, secret) {
  const hmac = crypto.createHmac('sha256', secret);
  const digest = hmac.update(JSON.stringify(payload)).digest('hex');
  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(digest)
  );
}
```

API BEST PRACTICES

1. Always Use HTTPS

All API requests must use HTTPS. HTTP requests will be rejected.

2. Handle Rate Limits

Implement exponential backoff when rate limited:

```
async function callApiWithRetry(endpoint, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      const response = await fetch(endpoint);
      if (response.status === 429) {
        const retryAfter = response.headers.get('Retry-After');
        await sleep(retryAfter * 1000);
```

```
      continue;
    }
    return response;
  } catch (error) {
    if (i === maxRetries - 1) throw error;
    await sleep(Math.pow(2, i) * 1000);
  }
 }
}
```

3. Validate Input

Always validate input data before making API calls to avoid unnecessary requests.

4. Use Idempotency Keys

For critical operations (payments, prescriptions), use idempotency keys:

```
POST /prescriptions
X-Idempotency-Key: unique-key-123
```

5. Request Timeouts

Set appropriate timeouts (recommended: 30 seconds for most endpoints, 60 seconds for file uploads).

6. Error Handling

Always handle errors gracefully:

```
try {
 const response = await api.createAppointment(data);
 // Handle success
} catch (error) {
 if (error.statusCode === 409) {
  // Handle conflict (slot already booked)
 } else if (error.statusCode === 422) {
  // Handle validation errors
```

```
    console.error(error.errors);
  } else {
    // Handle other errors
  }
}
```

SDK & CLIENT LIBRARIES

JavaScript/TypeScript

```
npm install @ithalamed/api-client
import { IthalamedClient } from '@ithalamed/api-client';

const client = new IthalamedClient({
  apiKey: 'your-api-key',
  environment: 'production' // or 'staging', 'development'
});

// Create appointment
const appointment = await client.appointments.create({
  patientId: 'patient-uuid',
  providerId: 'provider-uuid',
  date: '2025-01-25',
  startTime: '09:00'
});
```

Python

```
pip install ithalamed-python
from ithalamed import IthalamedClient

client = IthalamedClient(api_key='your-api-key')
```

```
# Get patient
patient = client.patients.get('patient-uuid')
print(patient.firstName)
```

PHP

```
composer require ithalamed/php-sdk
use Ithalamed\Client;

$client = new Client(['api_key' => 'your-api-key']);

// List appointments
$appointments = $client->appointments->list([
  'patientId' => 'patient-uuid',
  'status' => 'scheduled'
]);
```

POSTMAN COLLECTION

Download our Postman collection for easy API testing:

- Production Collection
- Sandbox Collection

Quick Setup:

1. Import collection into Postman
2. Set environment variables:
    - base_url: https://api.ithalamed.com/v1
    - access_token: Your JWT token
3. Run authentication request
4. Start testing endpoints

SANDBOX ENVIRONMENT

Test API integration in our sandbox:

- **Base URL:** https://api-sandbox.ithalamed.com/v1
- **Test Credentials:** Available at https://docs.ithalamed.com/sandbox

Sandbox Features:

- No real data processing
- Test payment flows
- Simulated external services
- Reset data daily at 00:00 UTC

Test Cards (for payment testing):

- Success: 4242 4242 4242 4242
- Declined: 4000 0000 0000 0002
- Insufficient Funds: 4000 0000 0000 9995

CHANGELOG

Version 1.0.0 (2025-01-20)

- Initial API release
- Core patient, provider, appointment endpoints
- Prescription management
- Telemedicine sessions
- Authentication & authorization

Upcoming (v1.1.0 - Q2 2025)

- Lab results API
- Enhanced analytics endpoints
- Bulk operations support
- GraphQL API (beta)
- WebSocket real-time updates

IthalaMed Security Implementation Guide

Comprehensive Security Architecture & Implementation

SECURITY OVERVIEW

Security Principles

1. **Defense in Depth** - Multiple layers of security controls
2. **Zero Trust Architecture** - Never trust, always verify
3. **Least Privilege** - Minimum necessary access
4. **Privacy by Design** - Security built-in from start
5. **Compliance First** - POPIA, HIPAA-equivalent standards

Threat Model

Assets to Protect:

- Patient medical records (PHI/PII)
- Authentication credentials
- Prescription data
- Financial information
- System availability

Threat Actors:

- External attackers (hackers, ransomware)
- Insider threats (malicious employees)
- Accidental data exposure
- Third-party breaches
- Social engineering

Attack Vectors:

- API vulnerabilities
- SQL injection
- Cross-site scripting (XSS)
- Authentication bypass
- Data interception

- DDoS attacks

---

1. AUTHENTICATION & AUTHORIZATION

1.1 JWT-Based Authentication

Token Structure:

```
// Access Token (JWT)
{
  "header": {
    "alg": "RS256",
    "typ": "JWT",
    "kid": "key-id-123"
  },
  "payload": {
    "sub": "550e8400-e29b-41d4-a716-446655440000", // User ID
    "email": "thandi@example.com",
    "userType": "patient",
    "permissions": [
      "read:own_medical_records",
      "write:own_profile",
      "read:appointments"
    ],
    "iat": 1642684800, // Issued at
    "exp": 1642688400, // Expires (1 hour)
    "iss": "https://auth.ithalamed.com",
    "aud": "https://api.ithalamed.com"
  },
  "signature": "..."
}
```

Implementation (Node.js):

```
import * as jwt from 'jsonwebtoken';
import { readFileSync } from 'fs';

export class JWTService {
 private privateKey: Buffer;
 private publicKey: Buffer;

 constructor() {
  // RSA key pair (2048-bit minimum)
  this.privateKey = readFileSync('/secure/keys/private.pem');
  this.publicKey = readFileSync('/secure/keys/public.pem');
 }

 generateAccessToken(userId: string, userType: string, permissions: string[]): string {
  return jwt.sign(
   {
    sub: userId,
    userType,
    permissions,
    tokenType: 'access'
   },
   this.privateKey,
   {
    algorithm: 'RS256',
    expiresIn: '1h',
    issuer: 'https://auth.ithalamed.com',
    audience: 'https://api.ithalamed.com',
```

```
      keyid: 'key-id-123'
    }
  );
}

generateRefreshToken(userId: string): string {
  return jwt.sign(
    {
      sub: userId,
      tokenType: 'refresh'
    },
    this.privateKey,
    {
      algorithm: 'RS256',
      expiresIn: '7d',
      issuer: 'https://auth.ithalamed.com'
    }
  );
}

verifyToken(token: string): jwt.JwtPayload {
  try {
    return jwt.verify(token, this.publicKey, {
      algorithms: ['RS256'],
      issuer: 'https://auth.ithalamed.com',
      audience: 'https://api.ithalamed.com'
    }) as jwt.JwtPayload;
  } catch (error) {
```

```
    throw new UnauthorizedException('Invalid token');
   }
  }
}
```

1.2 Multi-Factor Authentication (MFA)

Implementation using TOTP (Time-based One-Time Password):

```
import * as speakeasy from 'speakeasy';
import * as QRCode from 'qrcode';

export class MFAService {
 async generateSecret(userId: string, email: string) {
  const secret = speakeasy.generateSecret({
   name: `IthalaMed (${email})`,
   issuer: 'IthalaMed',
   length: 32
  });

  // Generate QR code for user to scan
  const qrCodeUrl = await QRCode.toDataURL(secret.otpauth_url);

  // Store secret.base32 encrypted in database
  await this.storeUserMFASecret(userId, secret.base32);

  return {
   secret: secret.base32,
   qrCode: qrCodeUrl
  };
 }
```

```
 verifyToken(userId: string, token: string): boolean {
  const secret = await this.getUserMFASecret(userId);

  return speakeasy.totp.verify({
   secret: secret,
   encoding: 'base32',
   token: token,
   window: 1 // Allow 1 step before/after (30 seconds)
  });
 }
}
```

MFA Enforcement:

```
@Injectable()
export class MFAGuard implements CanActivate {
 canActivate(context: ExecutionContext): boolean {
  const request = context.switchToHttp().getRequest();
  const user = request.user;

  // Check if MFA is required for this user
  if (user.mfaEnabled && !request.headers['x-mfa-verified']) {
   throw new UnauthorizedException({
    message: 'MFA verification required',
    mfaRequired: true
   });
  }

  return true;
```

}

}

1.3 Role-Based Access Control (RBAC)

Permission Matrix:

Resource	Patient	Provider	Pharmacist	Admin	System
Own Medical Records	Read/Write	-	-	Read	Read/Write
Other Medical Records	-	Read/Write	Read	Read	Read/Write
Appointments (Own)	Read/Write	Read/Write	-	Read/Write	Read/Write
Prescriptions (Create)	-	Write	-	-	-
Prescriptions (Dispense)	-	-	Write	-	-
Prescriptions (Read Own)	Read	-	-	-	-
User Management	-	-	-	Read/Write	Read/Write
System Config	-	-	-	Read/Write	Read/Write

Implementation:

```
// Define permissions
export enum Permission {
  READ_OWN_MEDICAL_RECORDS = 'read:own_medical_records',
  WRITE_OWN_MEDICAL_RECORDS = 'write:own_medical_records',
  READ_ALL_MEDICAL_RECORDS = 'read:all_medical_records',
  WRITE_MEDICAL_RECORDS = 'write:medical_records',
  CREATE_PRESCRIPTION = 'create:prescription',
  DISPENSE_PRESCRIPTION = 'dispense:prescription',
  MANAGE_USERS = 'manage:users',
  SYSTEM_ADMIN = 'system:admin'
}


// Define roles
```

```
export const ROLES = {
  PATIENT: [
    Permission.READ_OWN_MEDICAL_RECORDS,
    Permission.WRITE_OWN_MEDICAL_RECORDS,
    'read:own_appointments',
    'write:own_appointments',
    'read:own_prescriptions'
  ],
  PROVIDER: [
    Permission.READ_ALL_MEDICAL_RECORDS,
    Permission.WRITE_MEDICAL_RECORDS,
    Permission.CREATE_PRESCRIPTION,
    'read:appointments',
    'write:appointments'
  ],
  PHARMACIST: [
    'read:prescriptions',
    Permission.DISPENSE_PRESCRIPTION,
    'read:patient_medication_history'
  ],
  ADMIN: [
    Permission.MANAGE_USERS,
    Permission.SYSTEM_ADMIN,
    'read:all',
    'write:all'
  ]
};
```

```
// Permission guard
@Injectable()
export class PermissionsGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const requiredPermissions = this.reflector.get<Permission[]>(
      'permissions',
      context.getHandler()
    );

    if (!requiredPermissions) {
      return true;
    }

    const request = context.switchToHttp().getRequest();
    const user = request.user;

    return requiredPermissions.every(permission =>
      user.permissions?.includes(permission)
    );
  }
}

// Usage in controller
@Controller('medical-records')
export class MedicalRecordsController {
  @Get(':id')
```

```
 @UseGuards(JwtAuthGuard, PermissionsGuard)
 @Permissions(Permission.READ_ALL_MEDICAL_RECORDS)
 async getMedicalRecord(@Param('id') id: string) {
  // Only users with READ_ALL_MEDICAL_RECORDS permission can access
  return this.medicalRecordsService.findById(id);
 }
}
```

1.4 Attribute-Based Access Control (ABAC)

For fine-grained access:

```
export class ABACService {
 canAccess(
  user: User,
  resource: Resource,
  action: string,
  context?: any
 ): boolean {
  // Rule 1: Patients can only access their own records
  if (user.type === 'patient' && resource.type === 'medical_record') {
   return resource.patientId === user.id;
  }

  // Rule 2: Providers can access records of their patients only
  if (user.type === 'provider' && resource.type === 'medical_record') {
   return this.isProviderForPatient(user.id, resource.patientId);
  }

  // Rule 3: Emergency access (break-glass)
  if (context?.emergency === true && user.type === 'ems') {
```

```
    this.logEmergencyAccess(user, resource);
    return true;
   }

   // Rule 4: Data retention - can't access archived records older than 7 years
   if (resource.archivedAt && this.isOlderThan(resource.archivedAt, 7, 'years')) {
    return user.permissions.includes('access:archived');
   }

   return false;
  }
}
```

---

2. DATA ENCRYPTION

2.1 Encryption at Rest

Database Encryption:

```
-- PostgreSQL Transparent Data Encryption (TDE)
-- Enable at database level
ALTER DATABASE ithalamed_db SET encryption = 'on';

-- Column-level encryption for sensitive fields
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Encrypt ID number
CREATE TABLE patients (
   id UUID PRIMARY KEY,
   id_number BYTEA NOT NULL, -- Encrypted
   -- Other fields...
```

```
);

-- Encryption functions
CREATE OR REPLACE FUNCTION encrypt_id_number(plain_text TEXT)
RETURNS BYTEA AS $$
BEGIN
  RETURN pgp_sym_encrypt(plain_text, current_setting('app.encryption_key'));
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION decrypt_id_number(encrypted BYTEA)
RETURNS TEXT AS $$
BEGIN
  RETURN pgp_sym_decrypt(encrypted, current_setting('app.encryption_key'));
END;
$$ LANGUAGE plpgsql;
```

Application-Level Encryption:

```
import * as crypto from 'crypto';

export class EncryptionService {
 private readonly algorithm = 'aes-256-gcm';
 private readonly keyLength = 32;
 private readonly ivLength = 16;

 constructor(
  @Inject('ENCRYPTION_KEY')
  private readonly encryptionKey: string
 ) {}
```

```
 encrypt(plaintext: string): string {
  const key = crypto.scryptSync(this.encryptionKey, 'salt', this.keyLength);
  const iv = crypto.randomBytes(this.ivLength);

  const cipher = crypto.createCipheriv(this.algorithm, key, iv);

  let encrypted = cipher.update(plaintext, 'utf8', 'hex');
  encrypted += cipher.final('hex');

  const authTag = cipher.getAuthTag();

  // Return IV + AuthTag + Encrypted data
  return iv.toString('hex') + authTag.toString('hex') + encrypted;
 }

 decrypt(ciphertext: string): string {
  const key = crypto.scryptSync(this.encryptionKey, 'salt', this.keyLength);

  // Extract IV, AuthTag, and encrypted data
  const iv = Buffer.from(ciphertext.slice(0, this.ivLength * 2), 'hex');
  const authTag = Buffer.from(ciphertext.slice(this.ivLength * 2, (this.ivLength + 16) * 2),
'hex');
  const encrypted = ciphertext.slice((this.ivLength + 16) * 2);

  const decipher = crypto.createDecipheriv(this.algorithm, key, iv);
  decipher.setAuthTag(authTag);
```

```
    let decrypted = decipher.update(encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }
}
```

Key Management with AWS KMS:

```
import { KMS } from 'aws-sdk';

export class KMSService {
  private kms: KMS;

  constructor() {
    this.kms = new KMS({
      region: process.env.AWS_REGION
    });
  }

  async encrypt(plaintext: string, keyId: string): Promise<string> {
    const result = await this.kms.encrypt({
      KeyId: keyId,
      Plaintext: Buffer.from(plaintext)
    }).promise();

    return result.CiphertextBlob.toString('base64');
  }

  async decrypt(ciphertext: string): Promise<string> {
```

```
    const result = await this.kms.decrypt({
      CiphertextBlob: Buffer.from(ciphertext, 'base64')
    }).promise();

    return result.Plaintext.toString('utf8');
  }

  async generateDataKey(keyId: string): Promise<{ plaintext: Buffer; encrypted: string }> {
    const result = await this.kms.generateDataKey({
      KeyId: keyId,
      KeySpec: 'AES_256'
    }).promise();

    return {
      plaintext: result.Plaintext,
      encrypted: result.CiphertextBlob.toString('base64')
    };
  }
}
```

2.2 Encryption in Transit

TLS Configuration (Nginx):

```
server {
    listen 443 ssl http2;
    server_name api.ithalamed.com;

    # TLS 1.3 only (most secure)
    ssl_protocols TLSv1.3;
```

```
    # Strong cipher suites
    ssl_ciphers 'TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256';
    ssl_prefer_server_ciphers on;

    # SSL certificates
    ssl_certificate /etc/ssl/certs/ithalamed.crt;
    ssl_certificate_key /etc/ssl/private/ithalamed.key;

    # OCSP Stapling
    ssl_stapling on;
    ssl_stapling_verify on;

    # HSTS (force HTTPS for 1 year)
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains;
preload" always;

    # Additional security headers
    add_header X-Frame-Options "DENY" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Referrer-Policy "strict-origin-when-cross-origin" always;

    # Content Security Policy
    add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline';
style-src 'self' 'unsafe-inline'" always;

    location / {
        proxy_pass http://backend:3000;
        proxy_set_header Host $host;
```

```
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Certificate Pinning (Mobile):

```
// iOS Implementation
import Alamofire

class CertificatePinningManager {
    static let shared = CertificatePinningManager()

    private let session: Session

    init() {
        let evaluators: [String: ServerTrustEvaluating] = [
            "api.ithalamed.com": PublicKeysTrustEvaluator()
        ]

        let manager = ServerTrustManager(evaluators: evaluators)

        self.session = Session(
            serverTrustManager: manager
        )
    }

    func request(_ url: String) -> DataRequest {
        return session.request(url)
```

```
  }
}
```

---

3. INPUT VALIDATION & SANITIZATION

3.1 API Input Validation

```
import { IsString, IsEmail, IsUUID, Length, Matches, IsDateString } from 'class-validator';

export class CreatePatientDto {
 @IsString()
 @Length(2, 100)
 @Matches(/^[a-zA-Z\s'-]+$/, {
  message: 'First name can only contain letters, spaces, hyphens and apostrophes'
 })
 firstName: string;

 @IsString()
 @Length(2, 100)
 @Matches(/^[a-zA-Z\s'-]+$/)
 lastName: string;

 @IsString()
 @Matches(/^\d{13}$/, {
  message: 'ID number must be exactly 13 digits'
 })
 @Transform(({ value }) => value.trim())
 idNumber: string;

 @IsDateString()
```

```
  dateOfBirth: string;

  @IsEmail()
  @Transform(({ value }) => value.toLowerCase().trim())
  email: string;

  @IsString()
  @Matches(/^\+27[0-9]{9}$/, {
    message: 'Phone number must be in format +27XXXXXXXXX'
  })
  phoneNumber: string;
}
```

3.2 SQL Injection Prevention

Always use parameterized queries:

```
// BAD - Vulnerable to SQL injection
async findByEmail(email: string) {
  const query = `SELECT * FROM users WHERE email = '${email}'`;
  return this.database.query(query);
}

// GOOD - Safe parameterized query
async findByEmail(email: string) {
  const query = 'SELECT * FROM users WHERE email = $1';
  return this.database.query(query, [email]);
}

// BETTER - Using ORM (TypeORM)
async findByEmail(email: string) {
```

```
  return this.userRepository.findOne({
    where: { email }
  });
}
```

3.3 XSS Prevention

```
import * as DOMPurify from 'isomorphic-dompurify';

export class SanitizationService {
  sanitizeHtml(dirty: string): string {
    return DOMPurify.sanitize(dirty, {
      ALLOWED_TAGS: ['b', 'i', 'em', 'strong', 'p', 'br'],
      ALLOWED_ATTR: []
    });
  }

  sanitizeInput(input: string): string {
    // Remove potential XSS vectors
    return input
      .replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, '')
      .replace(/javascript:/gi, '')
      .replace(/on\w+\s*=/gi, '');
  }
}
```

---

4. API SECURITY

4.1 Rate Limiting

```
import { RateLimiterMemory } from 'rate-limiter-flexible';
```

```
@Injectable()
export class RateLimitGuard implements CanActivate {
 private rateLimiter: RateLimiterMemory;

 constructor() {
  this.rateLimiter = new RateLimiterMemory({
   points: 100, // Number of requests
   duration: 60, // Per 60 seconds
   blockDuration: 60 * 15, // Block for 15 minutes
  });
 }

 async canActivate(context: ExecutionContext): Promise<boolean> {
  const request = context.switchToHttp().getRequest();
  const key = request.user?.id || request.ip;

  try {
   const rateLimiterRes = await this.rateLimiter.consume(key);

   // Add rate limit headers
   request.res.setHeader('X-RateLimit-Limit', 100);
   request.res.setHeader('X-RateLimit-Remaining', rateLimiterRes.remainingPoints);
   request.res.setHeader('X-RateLimit-Reset', new Date(Date.now() +
rateLimiterRes.msBeforeNext).toISOString());

   return true;
  } catch (error) {
   throw new TooManyRequestsException({
```

```
      message: 'Rate limit exceeded',
      retryAfter: Math.round(error.msBeforeNext / 1000)
    });
  }
 }
}
```

4.2 API Key Management

```
export class ApiKeyService {
 async generateApiKey(userId: string, name: string): Promise<string> {
  // Generate secure random key
  const key = `itm_${crypto.randomBytes(32).toString('hex')}`;

  // Hash the key before storing
  const hashedKey = await bcrypt.hash(key, 12);

  await this.apiKeyRepository.create({
   userId,
   name,
   keyHash: hashedKey,
   keyPrefix: key.substring(0, 12), // Store prefix for identification
   createdAt: new Date(),
   lastUsedAt: null,
   expiresAt: new Date(Date.now() + 365 * 24 * 60 * 60 * 1000) // 1 year
  });

  // Return unhashed key only once
  return key;
 }
```

```
async validateApiKey(key: string): Promise<User> {
 const prefix = key.substring(0, 12);

 const apiKey = await this.apiKeyRepository.findOne({
  where: { keyPrefix: prefix }
 });

 if (!apiKey) {
  throw new UnauthorizedException('Invalid API key');
 }

 if (apiKey.expiresAt < new Date()) {
  throw new UnauthorizedException('API key expired');
 }

 const isValid = await bcrypt.compare(key, apiKey.keyHash);

 if (!isValid) {
  throw new UnauthorizedException('Invalid API key');
 }

 // Update last used timestamp
 await this.apiKeyRepository.update(apiKey.id, {
  lastUsedAt: new Date()
 });

 return this.userService.findById(apiKey.userId);
```

```
 }
}
```

4.3 CORS Configuration

```
import { CorsOptions } from '@nestjs/common/interfaces/external/cors-options.interface';

export const corsConfig: CorsOptions = {
 origin: (origin, callback) => {
  const allowedOrigins = [
   'https://ithalamed.com',
   'https://app.ithalamed.com',
   'https://admin.ithalamed.com'
  ];

  // Allow requests with no origin (mobile apps, Postman)
  if (!origin) {
   return callback(null, true);
  }

  if (allowedOrigins.includes(origin)) {
   callback(null, true);
  } else {
   callback(new Error('Not allowed by CORS'));
  }
 },
 methods: 'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS',
 credentials: true,
 maxAge: 86400, // 24 hours
```

```
  allowedHeaders: [
    'Content-Type',
    'Authorization',
    'X-Requested-With',
    'X-API-Key',
    'X-Idempotency-Key'
  ],
  exposedHeaders: [
    'X-RateLimit-Limit',
    'X-RateLimit-Remaining',
    'X-RateLimit-Reset'
  ]
};
```

---

5. SECURITY MONITORING & LOGGING

5.1 Audit Logging

```
@Injectable()
export class AuditService {
  constructor(
    @InjectRepository(AuditLog)
    private auditRepository: Repository<AuditLog>
  ) {}

  async log(event: AuditEvent) {
    const auditLog = this.auditRepository.create({
      userId: event.userId,
      action: event.action,
      resource: event.resource,
```

```
    resourceId: event.resourceId,
    ipAddress: event.ipAddress,
    userAgent: event.userAgent,
    timestamp: new Date(),
    success: event.success,
    errorMessage: event.errorMessage,
    metadata: event.metadata
   });

   await this.auditRepository.save(auditLog);

   // Send to centralized logging (e.g., ELK stack)
   this.logger.info('Audit event', auditLog);
  }
}

// Usage in controller
@Post()
async createPrescription(@Body() dto: CreatePrescriptionDto, @Req() req: Request) {
 try {
   const prescription = await this.prescriptionService.create(dto);

   await this.auditService.log({
    userId: req.user.id,
    action: 'CREATE_PRESCRIPTION',
    resource: 'prescription',
    resourceId: prescription.id,
    ipAddress: req.ip,
```

```
      userAgent: req.headers['user-agent'],
      success: true,
      metadata: {
        patientId: dto.patientId,
        medicationCount: dto.medications.length
      }
    });

    return prescription;
  } catch (error) {
    await this.auditService.log({
      userId: req.user.id,
      action: 'CREATE_PRESCRIPTION',
      resource: 'prescription',
      resourceId: null,
      ipAddress: req.ip,
      userAgent: req.headers['user-agent'],
      success: false,
      errorMessage: error.message
    });

    throw error;
  }
}
```

5.2 Security Event Monitoring

```
export class SecurityMonitor {
  async detectAnomalies(userId: string) {
    // Check for suspicious activity
```

```
const recentLogs = await this.getRecentAuditLogs(userId, '1 hour');

// Multiple failed login attempts
const failedLogins = recentLogs.filter(log =>
  log.action === 'LOGIN' && !log.success
);

if (failedLogins.length >= 5) {
  await this.triggerAlert('MULTIPLE_FAILED_LOGINS', userId, {
    count: failedLogins.length
  });

  // Lock account
  await this.userService.lockAccount(userId);
}

// Access from new location
const currentIp = recentLogs[0]?.ipAddress;
const knownIps = await this.getUserKnownIps(userId);

if (currentIp && !knownIps.includes(currentIp)) {
  await this.triggerAlert('NEW_LOCATION_ACCESS', userId, {
    ipAddress: currentIp
  });
}

// Unusual data access patterns
const dataAccessCount = recentLogs.filter(log =>
```

```
    log.action.startsWith('READ_') && log.success
  ).length;

  if (dataAccessCount > 100) {
    await this.triggerAlert('EXCESSIVE_DATA_ACCESS', userId, {
      count: dataAccessCount
    });
  }
 }

 private async triggerAlert(type: string, userId: string, metadata: any) {
  // Send to security team
  await this.notificationService.send({
    type: 'SECURITY_ALERT',
    recipients: ['security@ithalamed.com'],
    channels: ['email', 'slack'],
    data: {
      alertType: type,
      userId,
      timestamp: new Date(),
      metadata
    }
  });

  // Log to SIEM
  this.logger.warn(`Security alert: ${type}`, { userId, metadata });
 }
}
```

---

6. COMPLIANCE & DATA PROTECTION

6.1 POPIA Compliance Implementation

```
@Injectable()
export class ConsentService {
 async grantConsent(userId: string, consentType: ConsentType) {
  const consent = await this.consentRepository.create({
   userId,
   consentType,
   granted: true,
   grantedAt: new Date(),
   version: '1.0',
   ipAddress: request.ip
  });

  await this.consentRepository.save(consent);

  // Audit log
  await this.auditService.log({
   userId,
   action: 'GRANT_CONSENT',
   resource: 'consent',
   resourceId: consent.id
  });
 }

 async checkConsent(userId: string, consentType: ConsentType): Promise<boolean> {
  const consent = await this.consentRepository.findOne({
```

```
    where: { userId, consentType, granted: true }
   });

   return !!consent;
  }

  async revokeConsent(userId: string, consentType: ConsentType) {
   await this.consentRepository.update(
    { userId, consentType },
    { granted: false, revokedAt: new Date() }
   );

   // Trigger data deletion if necessary
   if (consentType === ConsentType.DATA_PROCESSING) {
    await this.dataDeleteService.scheduleUserDataDeletion(userId);
   }
  }
}
```

6.2 Data Subject Rights

```
// Right to Access (Data Export)
async exportUserData(userId: string): Promise<Buffer> {
  const patient = await this.patientService.findById(userId);
  const appointments = await this.appointmentService.findByPatient(userId);
  const prescriptions = await this.prescriptionService.findByPatient(userId);
  const medicalRecords = await this.medicalRecordsService.findByPatient(userId);

  const exportData = {
   patient,
```

```
    appointments,
    prescriptions,
    medicalRecords,
    exportedAt: new Date(),
    format: 'JSON'
  };

  // Generate PDF or JSON export
  return this.generateExportFile(exportData);
}

// Right to Erasure (Right to be Forgotten)
async deleteUserData(userId: string) {
  // Verify user consent for deletion
  const canDelete = await this.canDeleteUserData(userId);

  if (!canDelete) {
    throw new BadRequestException('Cannot delete data due to legal or medical retention requirements');
  }

  // Soft delete with anonymization
  await this.patientRepository.update(userId, {
    firstName: 'DELETED',
    lastName: 'USER',
    email: `deleted_${userId}@anonymized.com`,
    phoneNumber: '+27000000000',
    idNumber: await this.encryptionService.encrypt('DELETED'),
```

```
    deletedAt: new Date()
  });

  // Anonymize medical records (retain for legal reasons)
  await this.anonymizeMedicalRecords(userId);

  // Audit log
  await this.auditService.log({
    userId,
    action: 'DATA_DELETION',
    resource: 'user',
    resourceId: userId,
    success: true
  });
}

// Data Retention Policy
async applyRetentionPolicy() {
  const retentionPeriod = 7; // years
  const cutoffDate = new Date();
  cutoffDate.setFullYear(cutoffDate.getFullYear() - retentionPeriod);

  // Archive old records
  await this.archiveOldRecords(cutoffDate);

  // Delete data beyond retention period (if legally allowed)
  await this.deleteExpiredData(cutoffDate);
}
```

---

7. INCIDENT RESPONSE

7.1 Data Breach Detection & Response

```
@Injectable()
export class IncidentResponseService {
 async detectBreach(event: SecurityEvent) {
  // Automated breach detection
  const isBreach = await this.analyzeSecurityEvent(event);

  if (isBreach) {
   await this.initiateBreachProtocol(event);
  }
 }

 private async initiateBreachProtocol(event: SecurityEvent) {
  // 1. Contain the breach
  await this.containBreach(event);

  // 2. Alert security team immediately
  await this.alertSecurityTeam(event);

  // 3. Document incident
  const incident = await this.createIncidentRecord(event);

  // 4. Assess impact
  const impact = await this.assessImpact(incident);

  // 5. If PHI/PII compromised, notify authorities (POPIA requirement)
```

```
  if (impact.phiCompromised) {
    await this.notifyAuthorities(incident);
    await this.notifyAffectedUsers(incident);
  }

  // 6. Implement fixes
  await this.implementSecurityFixes(incident);

  // 7. Post-incident review
  await this.schedulePostIncidentReview(incident);
}

private async containBreach(event: SecurityEvent) {
  // Revoke compromised credentials
  if (event.type === 'CREDENTIAL_COMPROMISE') {
    await this.revokeUserTokens(event.userId);
    await this.forcePasswordReset(event.userId);
  }

  // Block malicious IPs
  if (event.ipAddress) {
    await this.firewallService.blockIp(event.ipAddress);
  }

  // Isolate affected systems
  if (event.affectedSystems) {
    await this.isolateSystems(event.affectedSystems);
  }
```

```
}

private async notifyAuthorities(incident: Incident) {
  // POPIA requires notification within 72 hours
  const notification = {
    incidentId: incident.id,
    incidentType: incident.type,
    dataTypes: incident.compromisedDataTypes,
    affectedCount: incident.affectedUsers.length,
    mitigationSteps: incident.mitigationSteps,
    timestamp: new Date()
  };

  // Send to Information Regulator (South Africa)
  await this.sendToRegulator(notification);
}

private async notifyAffectedUsers(incident: Incident) {
  for (const userId of incident.affectedUsers) {
    await this.notificationService.send({
      type: 'SECURITY_BREACH',
      userId,
      channels: ['email', 'sms', 'push'],
      data: {
        incidentType: incident.type,
        dataTypes: incident.compromisedDataTypes,
        actionRequired: 'Please reset your password immediately',
        supportContact: 'security@ithalamed.com'
```

```
      }
    });
  }
 }
}
```

7.2 Security Metrics & KPIs

```
export class SecurityMetrics {
 async generateSecurityReport(period: 'daily' | 'weekly' | 'monthly') {
  const metrics = {
   // Authentication metrics
   failedLoginAttempts: await this.countFailedLogins(period),
   accountLockouts: await this.countAccountLockouts(period),
   mfaAdoptionRate: await this.calculateMfaAdoptionRate(),

   // Access control metrics
   unauthorizedAccessAttempts: await this.countUnauthorizedAccess(period),
   privilegedAccessEvents: await this.countPrivilegedAccess(period),

   // Data protection metrics
   encryptedDataPercentage: await this.calculateEncryptedDataPercentage(),
   dataExfiltrationAttempts: await this.countDataExfiltration(period),

   // Vulnerability metrics
   criticalVulnerabilities: await this.countCriticalVulnerabilities(),
   patchComplianceRate: await this.calculatePatchCompliance(),

   // Incident metrics
   securityIncidents: await this.countSecurityIncidents(period),
```

```
    meanTimeToDetect: await this.calculateMTTD(period),
    meanTimeToRespond: await this.calculateMTTR(period),

    // Compliance metrics
    complianceScore: await this.calculateComplianceScore(),
    auditLogRetention: await this.checkAuditLogRetention()
   };

   return metrics;
  }
}
```

---

8. PENETRATION TESTING & VULNERABILITY MANAGEMENT

8.1 Automated Security Scanning

```
# GitHub Actions - Security Scanning
name: Security Scan

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  schedule:
    - cron: '0 2 * * *' # Daily at 2 AM

jobs:
  dependency-scan:
    runs-on: ubuntu-latest
```

```
  steps:
    - uses: actions/checkout@v2

    - name: Run npm audit
      run: npm audit --audit-level=high

    - name: Snyk security scan
      uses: snyk/actions/node@master
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
      with:
        args: --severity-threshold=high

  code-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Run SonarQube scan
        uses: sonarsource/sonarcloud-github-action@master
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}

      - name: Run Semgrep
        uses: returntocorp/semgrep-action@v1
        with:
          config: >-
```

```
        p/security-audit
        p/owasp-top-ten

  container-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Build Docker image
        run: docker build -t ithalamed/api:latest .

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: ithalamed/api:latest
          severity: CRITICAL,HIGH
```

8.2 Manual Penetration Testing Checklist

Quarterly Penetration Testing:

- [ ] Authentication bypass attempts
- [ ] SQL injection testing (all endpoints)
- [ ] XSS vulnerability testing
- [ ] CSRF protection verification
- [ ] API rate limiting testing
- [ ] Authorization bypass attempts
- [ ] Session management testing
- [ ] Encryption validation
- [ ] Input validation testing
- [ ] File upload security

- [ ] API key security
- [ ] Third-party integration security
- [ ] Mobile app security (if applicable)
- [ ] Infrastructure security (AWS, database)

---

9. SECURITY TRAINING & AWARENESS

9.1 Developer Security Training

Mandatory Training Topics:

1. **OWASP Top 10** (8 hours)
    - Injection flaws
    - Broken authentication
    - Sensitive data exposure
    - XML external entities
    - Broken access control
    - Security misconfiguration
    - XSS
    - Insecure deserialization
    - Using components with known vulnerabilities
    - Insufficient logging & monitoring
2. **Secure Coding Practices** (4 hours)
    - Input validation
    - Output encoding
    - Authentication & session management
    - Access control
    - Cryptographic practices
    - Error handling
    - Data protection
3. **Healthcare Security & Compliance** (4 hours)

- POPIA requirements
- HIPAA-equivalent standards
- Patient data protection
- Consent management
- Breach notification requirements

9.2 Security Code Review Checklist

```
// Security code review checklist
export const SECURITY_REVIEW_CHECKLIST = {
 authentication: [
  'Are passwords hashed with bcrypt (cost >= 12)?',
  'Is MFA implemented for sensitive operations?',
  'Are JWT tokens signed with RS256?',
  'Is token expiration appropriate (1 hour for access)?',
  'Are refresh tokens properly secured?'
 ],
 authorization: [
  'Is RBAC/ABAC properly implemented?',
  'Are all endpoints protected with guards?',
  'Is the principle of least privilege followed?',
  'Are resource ownership checks in place?'
 ],
 dataProtection: [
  'Is sensitive data encrypted at rest?',
  'Is TLS 1.3 enforced for all connections?',
  'Are database credentials stored securely?',
  'Is PII/PHI properly redacted in logs?'
 ],
 inputValidation: [
```

```
    'Are all inputs validated with class-validator?',
    'Are parameterized queries used (no string concatenation)?',
    'Is output properly encoded/sanitized?',
    'Are file uploads validated (type, size, content)?'
  ],
  errorHandling: [
    'Are error messages generic (no stack traces)?',
    'Are errors logged with proper context?',
    'Are sensitive details excluded from error responses?'
  ],
  logging: [
    'Are all security events logged?',
    'Do logs include user ID, timestamp, action?',
    'Are logs tamper-proof?',
    'Is PII excluded from logs?'
  ]
};
```

---

10. SECURITY CONFIGURATION CHECKLIST

Production Security Checklist

Infrastructure:

- [ ] TLS 1.3 enabled on all endpoints
- [ ] WAF (Web Application Firewall) configured
- [ ] DDoS protection enabled
- [ ] VPC/private subnets configured
- [ ] Security groups properly configured
- [ ] Intrusion detection system (IDS) deployed
- [ ] Database encryption enabled

- [ ] Backup encryption enabled
- [ ] Key rotation policy implemented

Application:

- [ ] Environment variables secured (not in code)
- [ ] Debug mode disabled in production
- [ ] Error stack traces disabled
- [ ] CORS properly configured
- [ ] Security headers configured
- [ ] Rate limiting enabled
- [ ] Input validation on all endpoints
- [ ] SQL injection protection verified
- [ ] XSS protection enabled
- [ ] CSRF protection enabled

Authentication & Authorization:

- [ ] Strong password policy enforced
- [ ] MFA available for all users
- [ ] Session timeout configured (1 hour)
- [ ] JWT secret stored securely
- [ ] Refresh token rotation enabled
- [ ] Account lockout after failed attempts
- [ ] Password reset flow secured

Monitoring & Logging:

- [ ] Centralized logging configured (ELK/Splunk)
- [ ] Security event monitoring enabled
- [ ] Audit logs tamper-proof
- [ ] Log retention policy (7 years)
- [ ] Alerting configured for security events
- [ ] SIEM integration completed

Compliance:

- [ ] POPIA compliance verified
- [ ] Data retention policy implemented
- [ ] Consent management system operational
- [ ] Privacy policy published
- [ ] Data breach response plan documented
- [ ] Regular security audits scheduled
- [ ] Penetration testing conducted (quarterly)

Third-Party:

- [ ] Third-party services security reviewed
- [ ] API keys rotated regularly
- [ ] Vendor security assessments completed
- [ ] Data processing agreements signed
- [ ] Subprocessor list maintained

SECURITY CONTACTS

Security Team:

- Security Email: security@ithalamed.com
- Bug Bounty: bugbounty@ithalamed.com
- Incident Response: incident@ithalamed.com (24/7)

Responsible Disclosure: If you discover a security vulnerability, please email security@ithalamed.com with:

1. Description of the vulnerability
2. Steps to reproduce
3. Potential impact
4. Your contact information

We commit to:

- Acknowledge receipt within 24 hours

- Provide initial assessment within 72 hours
- Keep you informed of progress
- Credit you in our security hall of fame (if desired)

Do NOT:

- Access or modify data without permission
- Perform DoS/DDoS attacks
- Use social engineering
- Disclose vulnerability publicly before we've addressed it