Kiorex All-in-One Medical Platform

Technical Architecture Document

Version: 1.0
Date: October 2025
Document Owner: Technical Architecture Team
Status: Final

---

1. Executive Summary

This document describes the technical architecture for the Kiorex healthcare platform. The architecture follows microservices principles, ensuring scalability, maintainability, and high availability. The system is designed to handle 10M+ users, process millions of transactions monthly, and maintain 99.9% uptime while ensuring HIPAA and GDPR compliance.

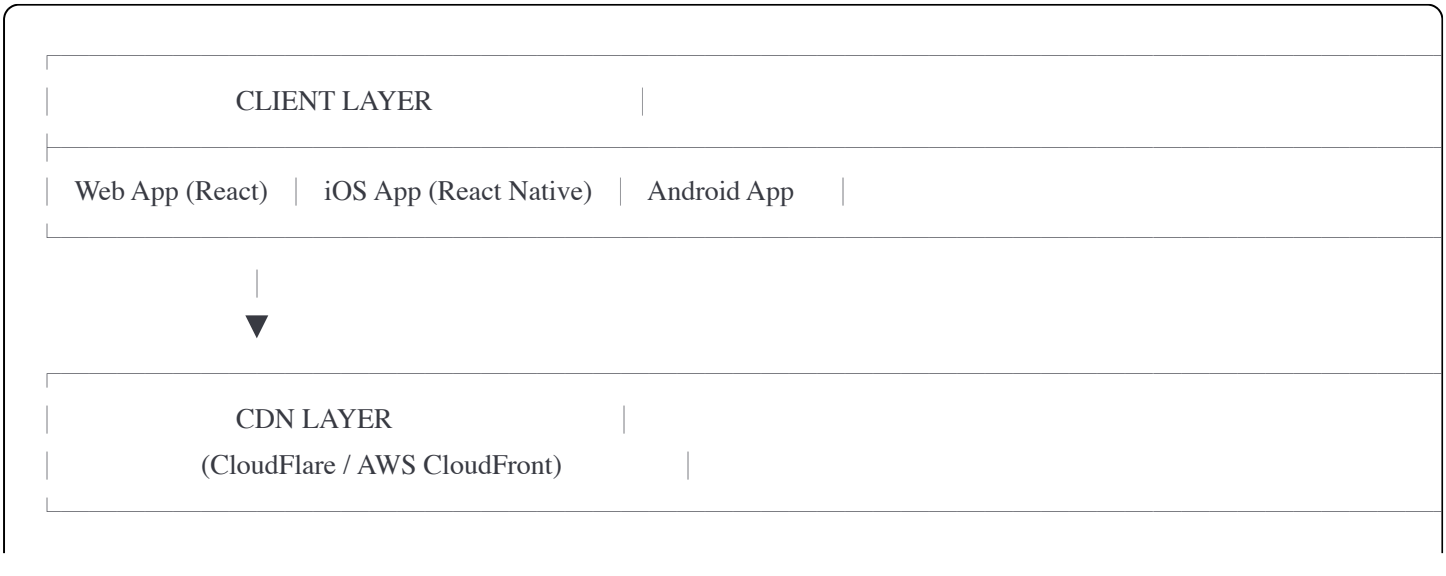
Key Architecture Decisions:

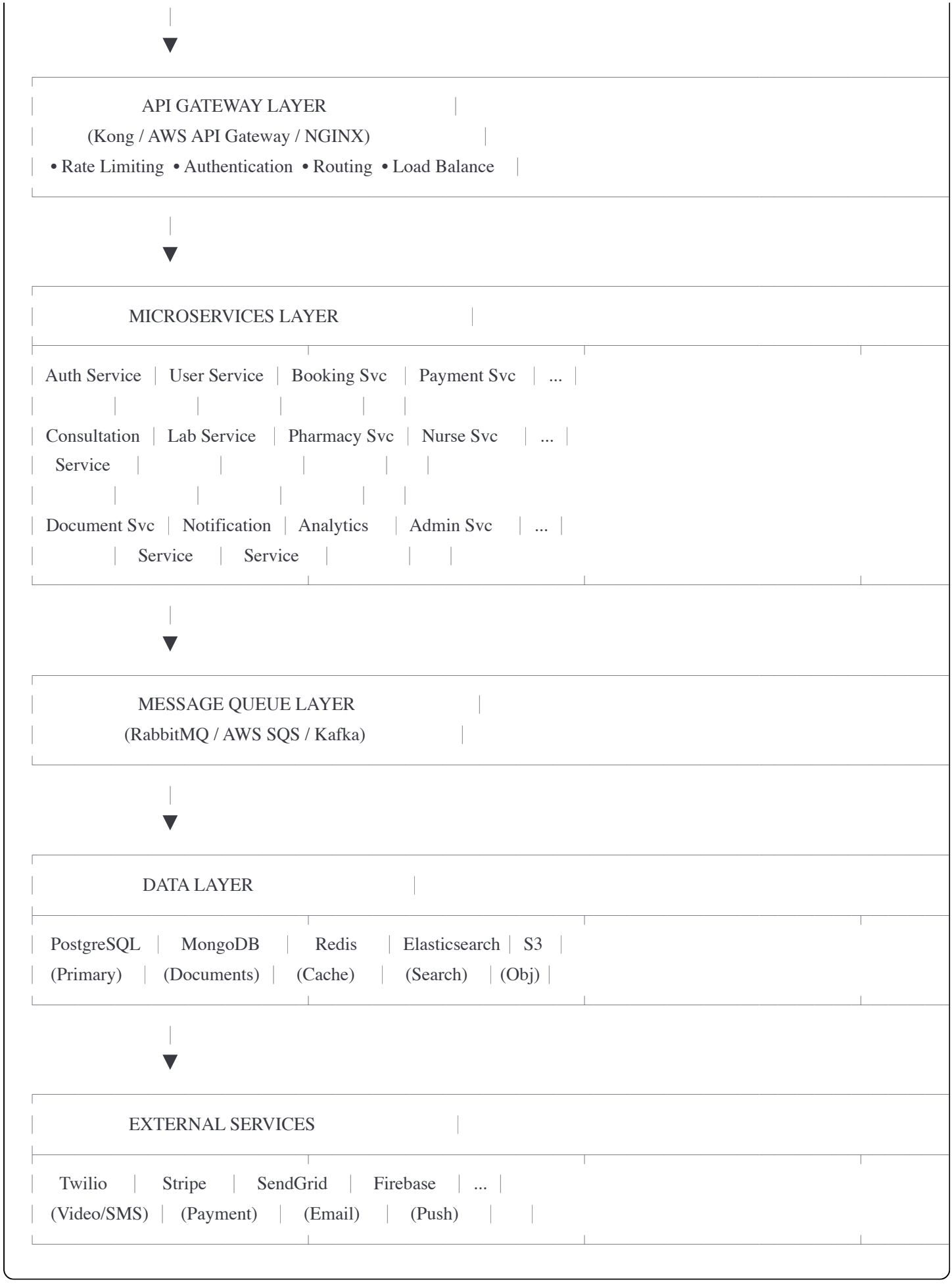
- **Microservices Architecture:** Independent, scalable services
- **Cloud-Native:** AWS/GCP for infrastructure
- **API-First Design:** RESTful APIs with GraphQL for complex queries
- **Event-Driven:** Message queues for asynchronous processing
- **Containerized:** Docker containers orchestrated by Kubernetes
- **Multi-Region:** Geographic distribution for low latency

---

2. Architecture Overview

2.1 High-Level Architecture Diagram





2.2 Architecture Principles

1. Scalability

- Horizontal scaling for all services
- Stateless services where possible
- Database sharding for large datasets
- Caching at multiple layers

2. Reliability

- Redundancy at all critical layers
- Automated failover
- Circuit breakers for external dependencies
- Graceful degradation

3. Security

- Zero-trust architecture
- End-to-end encryption
- Role-based access control
- Regular security audits

4. Maintainability

- Clear service boundaries
- Comprehensive documentation
- Automated testing
- CI/CD pipelines

5. Performance

- Response time < 500ms (P95)
- Database query optimization
- Efficient caching strategies
- CDN for static assets

---

3. Microservices Architecture

3.1 Service Catalog

Core Services

1. Authentication Service

- **Responsibility:** User authentication and authorization
- **Technology:** Node.js (NestJS)
- **Database:** PostgreSQL (users, roles, permissions)
- **Cache:** Redis (sessions, tokens)
- **APIs:**
  - POST /auth/register
  - POST /auth/login
  - POST /auth/logout
  - POST /auth/refresh
  - POST /auth/forgot-password
  - POST /auth/verify-otp
  - POST /auth/social-login (Google, Facebook, Apple)
- **Key Features:**
  - JWT token generation
  - OAuth 2.0 implementation
  - Multi-factor authentication
  - Session management
  - Password hashing (bcrypt)
- **Dependencies:** Email Service, SMS Service
- **Scaling:** Stateless, horizontal scaling

---

2. User Service

- **Responsibility:** User profile and medical history management
- **Technology:** Node.js (Express)
- **Database:** PostgreSQL (profiles), MongoDB (medical records)
- **APIs:**
  - GET /users/:id
  - PUT /users/:id
  - POST /users/:id/medical-history
  - GET /users/:id/family-members
  - POST /users/:id/emergency-contacts
- **Key Features:**
  - Profile CRUD operations

- Family member management
- Medical history storage
- Emergency contact management

- **Dependencies:** Auth Service, Document Service
- **Scaling:** Horizontal scaling with read replicas

---

3. Consultation Service

- **Responsibility:** Doctor consultations and appointments
- **Technology:** Node.js (NestJS)
- **Database:** PostgreSQL (appointments, consultations)
- **Message Queue:** RabbitMQ (booking events)
- **APIs:**
  - GET /consultations/doctors (search)
  - GET /consultations/doctors/:id
  - POST /consultations/bookings
  - GET /consultations/bookings/:id
  - PUT /consultations/bookings/:id/cancel
  - POST /consultations/:id/start
  - POST /consultations/:id/end
  - POST /consultations/:id/prescription
- **Key Features:**
  - Doctor search and filtering
  - Availability management
  - Booking workflow
  - Video call room management
  - Prescription generation
- **Dependencies:** User Service, Payment Service, Video Service, Notification Service
- **Scaling:** Horizontal scaling, event-driven

---

4. Payment Service

- **Responsibility:** Payment processing and transaction management
- **Technology:** Node.js (Express) with PCI DSS compliance

- **Database:** PostgreSQL (transactions, encrypted sensitive data)
- **APIs:**
  - POST /payments/process
  - POST /payments/refund
  - GET /payments/transactions/:id
  - POST /payments/webhooks/stripe
  - GET /payments/wallet/:userId
  - POST /payments/wallet/topup
- **Key Features:**
  - Multiple payment gateway integration
  - Transaction recording
  - Refund processing
  - Wallet management
  - Invoice generation
  - Webhook handling
- **Dependencies:** Stripe, Razorpay, PayPal APIs
- **Scaling:** Horizontal scaling with transaction queue
- **Security:** PCI DSS Level 1 compliance, encryption at rest

---

5. Document Service

- **Responsibility:** Medical document storage and management
- **Technology:** Python (FastAPI)
- **Storage:** AWS S3 (documents), PostgreSQL (metadata)
- **Search:** Elasticsearch
- **APIs:**
  - POST /documents/upload
  - GET /documents/:id
  - GET /documents/search
  - DELETE /documents/:id
  - POST /documents/:id/share
  - GET /documents/:id/download
- **Key Features:**
  - Secure file upload (virus scanning)

- OCR for categorization
- Document search and indexing
- Access control
- Sharing with time-limited links
- Audit logging

- **Dependencies:** AWS S3, ClamAV (antivirus), Tesseract (OCR)
- **Scaling:** S3 auto-scales, service scales horizontally

---

6. Lab Service

- **Responsibility:** Lab test marketplace and bookings
- **Technology:** Node.js (NestJS)
- **Database:** PostgreSQL (tests, bookings), MongoDB (results)
- **APIs:**
  - GET /labs/tests (search)
  - GET /labs/:labId
  - POST /labs/bookings
  - GET /labs/bookings/:id
  - POST /labs/bookings/:id/results
  - GET /labs/results/:userId
- **Key Features:**
  - Test catalog management
  - Lab partner integration
  - Home collection scheduling
  - Result upload and parsing
  - Abnormal value flagging
- **Dependencies:** User Service, Payment Service, Notification Service
- **Scaling:** Horizontal scaling

---

7. Pharmacy Service

- **Responsibility:** Pharmacy marketplace and medicine orders
- **Technology:** Node.js (Express)
- **Database:** PostgreSQL (orders), MongoDB (medicine catalog)

- **Search:** Elasticsearch (medicine search)
- **APIs:**
  - GET /pharmacy/medicines/search
  - GET /pharmacy/medicines/:id
  - POST /pharmacy/orders
  - GET /pharmacy/orders/:id
  - PUT /pharmacy/orders/:id/status
  - POST /pharmacy/prescriptions/validate
- **Key Features:**
  - Medicine catalog (50K+ SKUs)
  - Prescription validation
  - Order processing
  - Delivery tracking
  - Substitute suggestions
  - Drug interaction checking
- **Dependencies:** User Service, Payment Service, Delivery Service
- **Scaling:** Horizontal scaling, heavy caching

---

8. Surgery Service

- **Responsibility:** Surgery marketplace and bookings
- **Technology:** Node.js (NestJS)
- **Database:** PostgreSQL (surgeries, bookings)
- **APIs:**
  - GET /surgeries/search
  - GET /surgeries/:id
  - POST /surgeries/inquiries
  - POST /surgeries/bookings
  - GET /surgeries/bookings/:id
- **Key Features:**
  - Surgery catalog
  - Pricing comparison
  - Multi-step booking workflow
  - Document collection

    - Payment plans
- **Dependencies:** User Service, Payment Service, Document Service
- **Scaling:** Horizontal scaling

---

9. Nurse Service

- **Responsibility:** Nurse booking and service management
- **Technology:** Node.js (Express)
- **Database:** PostgreSQL (nurses, bookings)
- **APIs:**
    - GET /nurses/search
    - GET /nurses/:id
    - POST /nurses/bookings
    - POST /nurses/bookings/:id/checkin
    - POST /nurses/bookings/:id/checkout
    - POST /nurses/bookings/:id/notes
- **Key Features:**
    - Nurse profiles and verification
    - Availability management
    - GPS-based check-in/out
    - Service notes
    - Rating and reviews
- **Dependencies:** User Service, Payment Service, Location Service
- **Scaling:** Horizontal scaling

---

10. Notification Service

- **Responsibility:** Multi-channel notification delivery
- **Technology:** Node.js (Express)
- **Database:** MongoDB (notification logs)
- **Message Queue:** RabbitMQ (notification queue)
- **APIs:**
    - POST /notifications/send
    - GET /notifications/:userId

- PUT /notifications/preferences

- **Key Features:**
  - Push notifications (Firebase)
  - Email (SendGrid)
  - SMS (Twilio)
  - Template management
  - Delivery tracking
  - User preferences
  - Scheduling
- **Dependencies:** Firebase, Twilio, SendGrid
- **Scaling:** Queue-based processing, horizontal workers

---

11. Analytics Service

- **Responsibility:** Data analytics and reporting
- **Technology:** Python (FastAPI)
- **Database:** PostgreSQL (data warehouse), InfluxDB (time-series)
- **APIs:**
  - GET /analytics/dashboard
  - GET /analytics/reports/:type
  - POST /analytics/custom-query
- **Key Features:**
  - Real-time analytics
  - Custom reporting
  - Data aggregation
  - KPI tracking
  - Predictive analytics (ML models)
- **Dependencies:** All services (data collection)
- **Scaling:** Data warehouse with scheduled ETL jobs

---

12. Admin Service

- **Responsibility:** Administrative functions and dashboards
- **Technology:** Node.js (NestJS)

- **Database:** PostgreSQL
- **APIs:**
  - GET /admin/users
  - PUT /admin/users/:id/status
  - GET /admin/providers
  - POST /admin/providers/:id/verify
  - GET /admin/transactions
  - GET /admin/support/tickets
- **Key Features:**
  - User management
  - Provider verification
  - Transaction monitoring
  - Support ticketing
  - System configuration
- **Dependencies:** All services
- **Scaling:** Horizontal scaling

3.2 Service Communication

Synchronous Communication (REST)

- **Use Case:** Real-time operations requiring immediate response
- **Examples:** User authentication, payment processing, data retrieval
- **Protocol:** HTTP/HTTPS with JSON
- **Implementation:** RESTful APIs
- **Timeout:** 30 seconds
- **Retry:** 3 attempts with exponential backoff

Asynchronous Communication (Events)

- **Use Case:** Operations that don't require immediate response
- **Examples:** Notifications, analytics updates, background processing
- **Protocol:** Message queues (RabbitMQ/SQS)
- **Implementation:** Event-driven architecture
- **Guaranteed Delivery:** At-least-once delivery
- **Dead Letter Queue:** For failed messages

Service Mesh

- **Technology:** Istio or Linkerd
- **Features:**
  - Service discovery
  - Load balancing
  - Circuit breaking
  - Retry logic
  - Distributed tracing
  - mTLS between services

---

4. Database Architecture

4.1 Database Strategy

Primary Database: PostgreSQL

- **Use Cases:** Transactional data, relational data
- **Tables:**
  - users, user_profiles, user_roles
  - doctors, doctor_specializations, doctor_availability
  - appointments, consultations, prescriptions
  - transactions, payments, refunds
  - bookings (lab, pharmacy, surgery, nurse)
  - providers (labs, pharmacies, hospitals)

Schema Design Principles:

- Normalized to 3NF for transactional tables
- Denormalized for read-heavy tables
- Appropriate indexing for search queries
- Foreign key constraints for data integrity
- Soft deletes for audit trail

Scaling Strategy:

- Master-slave replication for read scaling
- Connection pooling

- Partitioning for large tables (by date/region)
- Read replicas in each region

Document Database: MongoDB

- **Use Cases:** Flexible schemas, nested documents
- **Collections:**
  - medical_records (complex nested structure)
  - medicine_catalog (varying attributes)
  - lab_test_results
  - consultation_notes
  - notification_logs

Scaling Strategy:

- Sharding by user_id for even distribution
- Replica sets for high availability
- Indexes on frequently queried fields

Cache: Redis

- **Use Cases:** Session storage, frequently accessed data, rate limiting
- **Data Structures:**
  - Strings: Session tokens, user preferences
  - Hashes: User profile cache
  - Sets: Active users, online doctors
  - Sorted Sets: Leaderboards, trending searches
  - Lists: Recent items, activity feeds

Caching Strategy:

- Cache-aside pattern
- TTL-based expiration (15min-24hr depending on data)
- Cache warming for popular data
- Redis Cluster for high availability

Search Engine: Elasticsearch

- **Use Cases:** Full-text search, complex queries
- **Indices:**
  - doctors (searchable by name, specialty, location)
  - medicines (searchable by name, composition)
  - lab_tests
  - medical_documents

Indexing Strategy:

- Real-time indexing via change data capture
- Custom analyzers for medical terms
- Geospatial queries for location-based search
- Aggregations for faceted search

Object Storage: AWS S3

- **Use Cases:** File storage (images, documents, videos)
- **Buckets:**
  - user-profile-images
  - medical-documents
  - consultation-recordings
  - prescription-pdfs
  - lab-reports

Organization:

- Folder structure: /{user_id}/{category}/{file_id}.{ext}
- Encryption at rest (AES-256)
- Versioning enabled
- Lifecycle policies (move to Glacier after 2 years)
- CDN (CloudFront) for fast access

Time-Series Database: InfluxDB

- **Use Cases:** Metrics, logs, time-series data
- **Measurements:**
  - api_response_times

- user_activity
- system_metrics
- business_kpis

Retention Policies:

- Real-time data: 7 days (raw)
- Aggregated hourly: 30 days
- Aggregated daily: 2 years

4.2 Data Backup & Recovery

Backup Strategy:

- PostgreSQL: Continuous archiving + daily full backups
- MongoDB: Hourly snapshots
- Redis: Daily RDB snapshots
- S3: Versioning enabled, cross-region replication

Recovery Strategy:

- RPO (Recovery Point Objective): 1 hour
- RTO (Recovery Time Objective): 4 hours
- Monthly disaster recovery drills
- Backup restoration testing

---

5. Security Architecture

5.1 Security Layers

Network Security

- **VPC (Virtual Private Cloud):** Isolated network
- **Subnets:** Public (API Gateway, Load Balancer), Private (Services, Databases)
- **Security Groups:** Firewall rules per service
- **WAF (Web Application Firewall):** Protection against common attacks
- **DDoS Protection:** CloudFlare or AWS Shield

Application Security

- **Authentication:** JWT tokens with 24-hour expiry
- **Authorization:** Role-based access control (RBAC)
- **API Security:** Rate limiting (100 req/min per user)
- **Input Validation:** All inputs sanitized
- **Output Encoding:** Prevent XSS attacks
- **CSRF Protection:** CSRF tokens for state-changing operations

Data Security

- **Encryption at Rest:** AES-256 for all databases and storage
- **Encryption in Transit:** TLS 1.3 for all communications
- **Data Masking:** PII masked in logs and non-production environments
- **Key Management:** AWS KMS or HashiCorp Vault
- **Secrets Management:** Environment-specific secret rotation

Compliance

- **HIPAA Compliance:**
  - Business Associate Agreements (BAAs)
  - Audit logging of all PHI access
  - Encryption of all PHI
  - Regular risk assessments
  - Staff training
- **GDPR Compliance:**
  - Data minimization
  - Right to access
  - Right to erasure
  - Data portability
  - Consent management

Security Monitoring

- **SIEM (Security Information and Event Management):** Centralized log analysis
- **Intrusion Detection:** Real-time threat detection
- **Vulnerability Scanning:** Weekly automated scans
- **Penetration Testing:** Quarterly by third-party

- **Security Audits:** Annual comprehensive audits

---

6. Infrastructure Architecture

6.1 Cloud Infrastructure (AWS)

Compute

- **EKS (Elastic Kubernetes Service):** Container orchestration
- **EC2 Instances:** t3.medium to c5.2xlarge based on service
- **Auto Scaling Groups:** Scale 2-20 instances per service
- **Lambda:** Serverless for scheduled jobs

Networking

- **Route 53:** DNS management
- **CloudFront:** CDN for static assets
- **Application Load Balancer:** Traffic distribution
- **VPC Peering:** Inter-region connectivity

Storage

- **S3:** Object storage for files
- **EBS:** Block storage for databases
- **EFS:** Shared file system

Database

- **RDS (PostgreSQL):** Managed relational database
- **DocumentDB:** MongoDB-compatible
- **ElastiCache (Redis):** Managed cache

Monitoring & Logging

- **CloudWatch:** Metrics and logs
- **X-Ray:** Distributed tracing
- **Prometheus + Grafana:** Custom metrics
- **ELK Stack:** Log aggregation and analysis

6.2 Kubernetes Architecture

Cluster Setup

- **Multi-zone deployment:** 3 availability zones
- **Node pools:**
  - System pool: 3 nodes (control plane, monitoring)
  - Application pool: 6-20 nodes (auto-scaling)
  - Database pool: 3 nodes (stateful workloads)

Deployment Strategy

- **Blue-Green Deployment:** Zero-downtime releases
- **Canary Releases:** Gradual rollout (10%, 50%, 100%)
- **Rolling Updates:** For minor changes

Resource Management

- **Resource Requests:** Guaranteed resources
- **Resource Limits:** Maximum allowed resources
- **Horizontal Pod Autoscaling:** Based on CPU/memory
- **Cluster Autoscaling:** Based on pending pods

Service Discovery

- **Kubernetes DNS:** Internal service discovery
- **Ingress Controller:** External traffic routing
- **Service Mesh (Istio):** Advanced routing and observability

---

7. Integration Architecture

7.1 Third-Party Integrations

Video Conferencing

- **Primary:** Twilio Video
- **Fallback:** Agora
- **Integration:** REST API
- **Features:** Room creation, recording, quality monitoring

Payment Gateways

- **Primary:** Stripe
- **Secondary:** Razorpay (India/MENA)
- **Integration:** REST API + Webhooks
- **Features:** Card payments, UPI, wallets, refunds

Communication

- **SMS:** Twilio
- **Email:** SendGrid
- **Push Notifications:** Firebase Cloud Messaging
- **Integration:** REST APIs

Maps & Location

- **Primary:** Google Maps API
- **Features:** Geocoding, distance calculation, routing

Analytics

- **Product Analytics:** Mixpanel
- **Error Tracking:** Sentry
- **APM:** Datadog

7.2 API Design

REST API Standards

- **Versioning:** URL-based (/v1/, /v2/)
- **Resource Naming:** Plural nouns (GET /users, POST /consultations)
- **HTTP Methods:** GET (read), POST (create), PUT (update), DELETE (delete)
- **Status Codes:**
  - 200 (OK), 201 (Created), 204 (No Content)
  - 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found)
  - 500 (Internal Server Error)
- **Response Format:** JSON
- **Pagination:** Limit/offset or cursor-based
- **Filtering:** Query parameters (?specialty=cardiology&rating>4)

- **Sorting:** Query parameter (?sort=rating:desc)

GraphQL (for complex queries)

- **Use Cases:** Mobile apps, complex nested data
- **Endpoint:** /graphql
- **Features:** Single endpoint, client-specified queries, reduced over-fetching

API Documentation

- **Format:** OpenAPI 3.0 (Swagger)
- **Tools:** Swagger UI for interactive exploration
- **Versioning:** Separate docs for each version

---

8. Performance Optimization

8.1 Caching Strategy

Multi-Layer Caching

1. **CDN Layer:** Static assets (images, CSS, JS) - 30 day TTL
2. **API Gateway Cache:** API responses - 5 minute TTL
3. **Application Cache (Redis):** Database query results - 15 minute TTL
4. **Database Cache:** Query cache - enabled

Cache Invalidation

- **Time-based:** TTL expiration
- **Event-based:** Cache invalidation on data updates
- **Manual:** Admin-triggered cache clear

8.2 Database Optimization

Query Optimization

- **Indexing:** All foreign keys and frequently queried columns
- **Query Analysis:** EXPLAIN plans for slow queries
- **N+1 Query Prevention:** Eager loading, batch queries

Connection Pooling

- **Pool Size:** 20-50 connections per service
- **Connection Reuse:** Persistent connections

- **Timeout:** 30 seconds

8.3 API Optimization

Response Optimization

- **Compression:** Gzip compression for responses
- **Pagination:** Limit to 100 items per page
- **Field Selection:** Allow clients to specify required fields
- **Lazy Loading:** Load expensive fields on demand

Rate Limiting

- **User Level:** 100 requests/minute
- **IP Level:** 1000 requests/minute
- **API Key Level:** Custom limits

---

9. Monitoring & Observability

9.1 Monitoring Stack

Application Monitoring

- **APM:** Datadog or New Relic
- **Metrics:**
  - Response times (P50, P95, P99)
  - Error rates
  - Throughput (requests/second)
  - Availability

Infrastructure Monitoring

- **Tool:** Prometheus + Grafana
- **Metrics:**
  - CPU usage
  - Memory usage
  - Disk I/O
  - Network traffic

Log Aggregation

- **Stack:** ELK (Elasticsearch, Logstash, Kibana)
- **Log Levels:** ERROR, WARN, INFO, DEBUG
- **Structured Logging:** JSON format
- **Retention:** 90 days (7 years for critical logs)

Distributed Tracing

- **Tool:** Jaeger or AWS X-Ray
- **Features:** End-to-end request tracking, latency analysis

Alerts

- **Channels:** PagerDuty, Slack, Email
- **Triggers:**
  - Error rate > 1%
  - Response time > 2 seconds
  - Service downtime
  - Database connection failures
  - Payment failures > 5%

---

10. Disaster Recovery & Business Continuity

10.1 High Availability

Multi-Region Deployment

- **Primary Region:** us-east-1 (N. Virginia)
- **Secondary Region:** eu-west-1 (Ireland)
- **Failover:** Automated DNS failover

Database Replication

- **PostgreSQL:** Master-slave replication across regions
- **MongoDB:** Replica sets across regions
- **Redis:** Redis Sentinel for automatic failover

Load Balancing

- **Global:** Route 53 with health checks

- **Regional:** Application Load Balancer
- **Service Level:** Kubernetes service load balancing

10.2 Backup Strategy

Automated Backups:

- Databases: Daily full + hourly incremental
- File storage: Continuous replication
- Configuration: Version-controlled (Git)

Backup Testing:

- Monthly restoration drills
- Quarterly full disaster recovery simulation

---

11. Development & Deployment

11.1 Development Workflow

Git Workflow

- **Branching Strategy:** GitFlow
- **Branches:** main, develop, feature/*, *hotfix/*
- **Code Review:** Mandatory for all PRs
- **Merge Requirements:** 2 approvals, CI passing

CI/CD Pipeline

Continuous Integration:

1. Code commit to feature branch
2. Automated tests run (unit, integration)
3. Code quality checks (SonarQube)
4. Security scanning (Snyk)
5. Build Docker image
6. Push to container registry

Continuous Deployment:

1. Merge to develop → Deploy to staging
2. Merge to main → Deploy to production

3. Blue-green deployment
4. Automated smoke tests
5. Rollback on failure

Environment Strategy

- **Local:** Developer machines
- **Development:** Shared dev environment
- **Staging:** Production mirror
- **Production:** Live environment

---

12. Appendices

A. Technology Stack Summary

Layer	Technology	Purpose
Frontend	React, React Native	Web and mobile apps
API Gateway	Kong / NGINX	API management
Backend	Node.js, Python	Microservices
Databases	PostgreSQL, MongoDB, Redis	Data storage
Search	Elasticsearch	Full-text search
Storage	AWS S3	Object storage
Queue	RabbitMQ	Message queue
Container	Docker, Kubernetes	Orchestration
Cloud	AWS	Infrastructure
Monitoring	Datadog, Prometheus	Observability
Logging	ELK Stack	Log management

B. Capacity Planning

Year 1 Capacity

- **Users:** 500K
- **Concurrent Users:** 10K
- **Requests/Second:** 500
- **Database Size:** 200GB
- **Storage:** 5TB

Year 3 Capacity

- **Users:** 5M
- **Concurrent Users:** 100K
- **Requests/Second:** 5,000
- **Database Size:** 2TB
- **Storage:** 50TB

C. Cost Estimates

Infrastructure Costs (Monthly)

Component	Year 1	Year 2	Year 3
Compute (EKS)	$8K	$25K	$60K
Database (RDS)	$5K	$15K	$35K
Storage (S3)	$2K	$8K	$20K
CDN	$3K	$10K	$25K
Monitoring	$2K	$5K	$12K
Total	**$20K**	**$63K**	**$152K**

Document Approval:

Role	Name	Signature	Date
CTO	_____	_____	_____
Lead Architect	_____	_____	_____
Security Lead	_____	_____	_____

End of Technical Architecture Document