Psyter Platform - Cloud Migration & Modernization Strategy

Document Date: November 13, 2025
Project: Psyter Platform Cloud Migration
Strategy: Frontend-First, Then Backend Module-by-Module
Status: Strategic Plan (Pre-Implementation)

The migration strategy outlined here is based on audit findings from 8 repositories. Final implementation timelines and resource requirements will be determined after completing:

1. **Feature Analysis** - Deep dive into each feature (migrate/redesign/add/sunset decisions)
2. **System Design** - Detailed microservices architecture, API contracts, data models
3. **Architecture Documentation** - Service boundaries, communication patterns, infrastructure topology
4. **Resource Planning** - Team composition, timeline estimation, budget allocation

---

Executive Summary

Based on the comprehensive audit revealing **641 critical issues** across 8 repositories, this document outlines a **pragmatic, risk-minimized migration strategy** to AWS/GCP cloud infrastructure with modern technology stack.

Strategic Approach

Phase 1: Frontend First

- Migrate Web Application → Next.js 14
- Migrate Mobile Apps → React Native
- **Benefits:** Immediate UX improvements, maintained backend compatibility

Phase 2: Backend Module-by-Module

- Strangler Fig Pattern: New microservices alongside legacy APIs
- Gradual migration without big-bang rewrite
- **Benefits:** Continuous delivery, reduced risk, parallel operation

---

Why Frontend-First Strategy?

Advantages

1. ☑ **Lower Risk**
    - Frontend changes don't affect backend data
    - Can rollback independently
    - Existing APIs continue to work

2. ☑ **Faster Time-to-Value**
    - Modern UI/UX improvements visible immediately
    - Better mobile performance (React Native)
    - SEO improvements (Next.js SSR)
3. ☑ **Team Momentum**
    - Quick wins build confidence
    - Modern stack attracts developers
    - Easier recruitment for React/Next.js vs legacy .NET
4. ☑ **Business Continuity**
    - Backend keeps running unchanged
    - Zero data migration risk initially
    - Gradual user migration possible
5. ☑ **Technical Validation**
    - Prove cloud infrastructure works
    - Test CI/CD pipelines
    - Validate monitoring/alerting setup

Technology Stack Decisions

Frontend Stack

- **Web Application:** Next.js 14 (React 18) - Modern framework with SSR/SSG for better SEO and performance
- **Mobile Apps:** React Native 0.73 - Single codebase for iOS & Android with 90% code sharing

Backend Stack (Microservices)

- **Auth Service:** Node.js (NestJS) - Modern async framework for authentication
- **User & Appointment Services:** Django 5.0 - Built-in security, admin panel, complex business logic
- **Payment & Notification Services:** Node.js - Event-driven architecture for real-time operations
- **Media & Reporting Services:** Node.js & Django - Optimized for their specific use cases

Cloud Infrastructure

- **Recommendation:** AWS (better ecosystem maturity, team familiarity)
- **Compute:** ECS Fargate / EKS for containerized applications
- **Database:** RDS PostgreSQL (HIPAA-compliant)
- **Storage & CDN:** S3 + CloudFront for media and static assets
- **Monitoring:** CloudWatch + X-Ray for observability

Step-by-Step Migration Plan

Phase 1: Frontend Migration

Stage 1: Infrastructure Setup & Web Application Foundation

Key Activities:

1. **Cloud Infrastructure Setup**
    - AWS account organization (production, staging, development)
    - Networking and security configuration
    - CI/CD pipeline setup with GitHub Actions
    - Monitoring and observability tools
2. **Next.js Web Application**
    - Modern web application with Next.js 14
    - Authentication integration with existing backend
    - Responsive design with modern UI components
    - API integration layer for seamless backend communication

Deliverables:

- ☑ AWS infrastructure operational
- ☑ Next.js application deployed
- ☑ CI/CD pipeline functional

Stage 2: Web Application Feature Migration

Core User Features:

- User dashboard with appointments and notifications
- Provider search and booking (simplified from 7 steps to 3)
- Appointment management (view, reschedule, cancel)
- Profile management and settings
- Payment history and methods
- Messaging and notifications

Key Improvements:

- Modern React components replacing legacy views
- Optimistic UI updates for better user experience
- Bilingual support (English & Arabic with RTL)
- Mobile-responsive design
- Improved accessibility and performance

Deliverables:

- ☑ Feature parity with legacy web application
- ☑ Bilingual support (EN/AR)
- ☑ Enhanced user experience

Stage 3: Mobile Application Migration (React Native)

React Native Development:

- Single codebase for iOS and Android with 60-70% code sharing
- Modern mobile architecture with TypeScript
- Shared business logic with web application
- Platform-specific optimizations where needed

Client App Features:

- Authentication with biometric support (Face ID, Touch ID, Fingerprint)
- Provider search and booking
- Video consultations
- Real-time messaging
- Appointment management
- Payment processing
- Offline support for viewing past appointments

Provider App Features:

- Schedule and availability management
- Appointment handling
- Video consultations
- Patient records access
- Prescription management
- Earnings and reports

Deliverables:

- ☑ React Native apps for iOS & Android
- ☑ Feature parity with legacy native apps
- ☑ Published to app stores (internal testing)

Phase 2: Backend Migration

Strategy: Strangler Fig Pattern

Gradual replacement approach instead of big-bang rewrite:

- New microservices deployed alongside legacy APIs
- API Gateway intelligently routes traffic to new or legacy services
- Dual-write ensures data consistency during transition
- Feature flags enable gradual user migration
- Continuous monitoring validates parity before full cutover

Key Principles:

1. **Dual-Write:** Write to both old and new systems during transition
2. **Gradual Migration:** Switch traffic incrementally with validation

3. **Data Consistency:** Continuous verification before cutover
4. **Risk Mitigation:** Easy rollback if issues detected
5. **Business Continuity:** Zero downtime, seamless user experience

---

Stage 4: Core Infrastructure & Auth Service

Microservices Infrastructure:

- Kubernetes cluster (EKS) with auto-scaling
- API Gateway for intelligent traffic routing
- PostgreSQL database (Multi-AZ, HIPAA-compliant)
- Message queue and event bus for async operations

Authentication Service (Node.js):

- Modern authentication with JWT tokens
- OAuth 2.0 and multi-factor authentication (2FA)
- Session management and password reset
- Biometric authentication support for mobile apps

Migration Approach:

- Deploy new auth service alongside legacy system
- Dual-write to ensure data consistency
- Gradual traffic migration with monitoring
- Rollback capability if issues arise

Deliverables:

- ☑ Microservices infrastructure operational
- ☑ Auth service deployed and validated
- ☑ Frontend integrated with new authentication

Stage 5: User Management & Appointment Services

User Management Service (Django):

- Complete user profile and account management
- Role-based access control
- Built-in admin panel for support operations
- GDPR compliance (data export/deletion)
- Audit trail for all user actions

Why Django: Built-in security features, admin panel, and ORM prevent common vulnerabilities like SQL injection

Appointment Service (Django):

- Appointment booking and management
- Provider availability and scheduling

- Complex business logic (timezones, conflicts, policies)
- Recurring appointments support
- Integration with notification service

Migration Approach:

- Shadow traffic to validate parity
- Gradual traffic migration with monitoring
- Data consistency verification throughout
- Fallback to legacy if discrepancies detected

Deliverables:

- ☑ User and Appointment services deployed
- ☑ Data consistency validated
- ☑ Gradual traffic migration successful

Stage 6: Payment & Notification Services

Payment Service (Node.js):

- Payment gateway integration (Stripe/PayFort)
- Secure payment processing with PCI-DSS compliance
- Webhook handling for payment events
- Refund processing and invoice generation
- Event-driven architecture for reliability

Why Node.js: Excellent for async webhooks and event-driven payment processing

Security Measures:

- Tokenization (never store card details)
- PCI-DSS Level 1 compliance
- Encryption at rest and in transit
- Webhook signature verification

Notification Service (Node.js):

- Multi-channel notifications (push, email, SMS, in-app)
- Appointment reminders and booking confirmations
- Payment receipts and system announcements
- WebSockets for real-time updates
- User preference management

Deliverables:

- ☑ Payment service handling all transactions securely
- ☑ Notification service operational across all channels
- ☑ Zero payment failures during migration

Stage 7: Media & Video Services

Media Service (Node.js):

- Secure file storage on S3 with CDN distribution
- Image processing and optimization
- Video transcoding for multiple formats
- Malware scanning on all uploads
- Access control with signed URLs
- Support for profile pictures, medical documents, prescriptions, session recordings

Security Measures:

- Antivirus scanning on upload
- File type and size validation
- Encryption at rest (S3 SSE)
- Access control via presigned URLs

Video Consultation Service:

- Enhanced video session reliability
- Pre-call network diagnostics
- Automatic reconnection handling
- Session recording and quality monitoring
- Improved error handling for better user experience

Recommendation: Continue with Agora.io, enhanced with better monitoring and reliability features

Deliverables:

- ☑ Media service with secure file handling
- ☑ Video service with improved reliability
- ☑ Enhanced user experience for consultations

Stage 8: Reporting & Analytics Services

Reporting Service (Django):

- Provider earnings and appointment statistics
- Revenue analytics and user engagement metrics
- Custom report builder for business insights
- Scheduled reports with email delivery
- Data warehouse integration for complex analytics

Why Django: Excellent ORM for complex queries, built-in admin for custom reports, Celery for background jobs

Admin Dashboard Service:

- Centralized user and provider management
- Content management and system settings

- Feature flags for controlled rollouts
- Audit logs for compliance
- Support tools for operations team

Deliverables:

- ☑ Reporting service with comprehensive analytics
- ☑ Admin dashboard for operations
- ☑ Support team enablement

Stage 9: Legacy Decommission & Optimization

Final Migration & Validation:

- Complete traffic cutover to new platform
- Legacy service decommissioning
- Data migration verification and archival
- Comprehensive performance testing
- Security and compliance audits (HIPAA, GDPR, PCI-DSS)

Platform Optimization:

- Performance tuning for optimal response times
- Cost optimization across cloud resources
- Documentation for operations and development teams
- Monitoring and alerting setup for production
- Knowledge transfer and team training

Deliverables:

- ☑ Legacy platform fully decommissioned
- ☑ 100% traffic on modern cloud platform
- ☑ Security and compliance validated
- ☑ Comprehensive operational documentation
- ☑ Team fully trained on new platform

Module-by-Module Migration Priority

Priority 1: Critical Services (Early Phase)

1. **Auth Service** - Security foundation
2. **User Service** - Core data
3. **Appointment Service** - Primary business logic
4. **Payment Service** - Revenue critical
5. **Notification Service** - User engagement

Priority 2: Supporting Services (Mid Phase)

6. **Media Service** - File handling

7. **Video Service** - Core feature improvement

Priority 3: Analytics & Admin (Later Phase)

8. **Reporting Service** - Business intelligence
9. **Admin Dashboard** - Operations support

Priority 4: Decommission & Polish (Final Phase)

10. **Legacy Cleanup** - Remove old code
11. **Optimization** - Performance tuning
12. **Documentation** - Knowledge transfer

Data Migration Strategy

Dual-Write with Gradual Read Migration

Phase 1: Dual Write

- Write operations go to both legacy and new systems
- Ensures data consistency during transition
- Allows validation before full cutover

Phase 2: Read from New

- Gradually shift read traffic to new services
- Legacy continues to receive writes
- Monitor for discrepancies

Phase 3: Legacy Readonly

- New services handle all operations
- Legacy kept as backup during transition
- Archive after verification period

Data Consistency:

- Continuous automated verification between systems
- Alerting for any discrepancies detected
- Sample-based deep comparison of records
- Operations team alerted for manual review when needed

Our Own Feature Analysis

Features to Migrate (Keep)

Core Features:

- ☑ User registration & authentication
- ☑ Provider search with filters

- ☑ Appointment booking
- ☑ Video consultations
- ☑ Payment processing
- ☑ Messaging/Chat
- ☑ Push notifications
- ☑ Prescriptions
- ☑ Medical records
- ☑ Multi-language support (EN/AR)

Provider Features:

- ☑ Schedule management
- ☑ Availability settings
- ☑ Patient management
- ☑ Earnings reports
- ☑ Session notes

Features to Redesign (Improve)

User Experience:

- 🔄 **Registration Flow:** 7 steps → 3 steps (60% drop-off → <20%)
- 🔄 **Provider Search:** Add smart filters, AI recommendations
- 🔄 **Booking Flow:** Simplify from 8 clicks to 3 clicks
- 🔄 **Video Pre-call:** Add network test, device check
- 🔄 **Questionnaire:** 50 questions → Adaptive questionnaire (10-15 questions)

Accessibility:

- 🔄 **WCAG Compliance:** 35% → 100% (screen reader support, keyboard navigation)
- 🔄 **Mobile Touch Targets:** 32dp → 48dp
- 🔄 **Color Contrast:** 2.8:1 → 4.5:1 (AA standard)

Performance:

- 🔄 **Provider List:** 5-10s load → <1s (pagination, caching)
- 🔄 **Image Loading:** Lazy loading, WebP format, CDN
- 🔄 **App Startup:** 2-3s → <1s

Features to Add (New)

User-Requested:

- ➕ **Biometric Authentication:** Face ID, Touch ID, Fingerprint
- ➕ **Dark Mode:** System theme support
- ➕ **Offline Mode:** View past appointments, cached data
- ➕ **Saved Payment Methods:** Secure tokenization
- ➕ **Appointment Reminders:** 24h, 1h before (customizable)
- ➕ **Provider Reviews/Ratings:** User feedback system
- ➕ **In-App File Sharing:** Share documents during session

- ➕ **Smart Scheduling:** AI-powered time recommendations
- ➕ **Telemedicine History:** Aggregated health timeline
- ➕ **Family Accounts:** Manage dependents

Business Value:

- ➕ **Referral Program:** User acquisition
- ➕ **Subscription Plans:** Recurring revenue
- ➕ **Insurance Integration:** Claims processing
- ➕ **Corporate Wellness:** B2B packages
- ➕ **Analytics Dashboard:** User insights

Features to Sunset (Remove)

Low Usage:

- ❌ **Collaboration Mode (Legacy):** <2% usage, replace with better video
- ❌ **Old Calendar View:** Migrate to modern calendar component
- ❌ **jQuery Dependencies:** Replace with React
- ❌ **Multiple Authentication Systems:** Consolidate to single auth

---

Risk Management

Risk	Impact	Mitigation Strategy
Data Loss During Migration	Critical	Dual-write strategy, continuous verification, legacy backup retention
Performance Degradation	High	Load testing before cutover, gradual traffic shift, rollback capability
Team Skill Gaps	Medium	Training programs, pair programming, external consultants when needed
Timeline Delays	Medium	Phased approach with MVP mindset, prioritize critical features
Budget Overruns	High	Phased implementation, continuous cost monitoring, regular reviews
Security Vulnerabilities	Critical	Regular security audits, penetration testing, bug bounty program
User Resistance	Medium	Gradual rollout, user training, feedback collection and response

Rollback Capability

Frontend:

- Legacy web application maintained during transition
- Quick DNS switchback capability (< 5 minutes)
- Mobile app version management for emergency updates

Backend:

- API Gateway enables instant routing to legacy services
- Feature flags for rapid service disable
- Data synchronization to legacy database if needed

Conclusion

This migration strategy balances risk, speed, and business continuity through:

1. **Frontend-First Approach:** Immediate UX improvements, lower risk, faster time-to-value
2. **Module-by-Module Backend:** Gradual migration without big-bang disruption
3. **Strangler Fig Pattern:** Safe coexistence of new and legacy systems
4. **Cloud-Native Infrastructure:** Scalable, secure, HIPAA/GDPR compliant
5. **Modern Technology Stack:** Addresses inherent security and maintainability issues

Expected Business Outcomes:

- ☑ **Security & Compliance:** HIPAA, GDPR, PCI-DSS Level 1 compliant platform
- ☑ **Scalability:** Infrastructure capable of 10x growth without redesign
- ☑ **Development Velocity:** 3-5x faster feature delivery with modern stack
- ☑ **User Experience:** Improved performance, accessibility, and engagement
- ☑ **Cost Efficiency:** Reduced maintenance overhead, optimized cloud resources
- ☑ **Competitive Advantage:** Modern features attract and retain users

Next Steps:

1. **Stakeholder Approval:** Present strategy to leadership and key stakeholders
2. **Feature Analysis:** Deep dive into all features for migrate/redesign/add/sunset decisions
3. **System Design:** Detailed architecture, API contracts, and data models
4. **Resource Planning:** Team assembly, timeline estimation, budget allocation
5. **Implementation:** Begin Phase 1 infrastructure and frontend development

---

Document Prepared By: Technical Leadership Team
Date: November 13, 2025
Status: STRATEGIC PLAN - PENDING DETAILED ANALYSIS
Next Phase: Feature Analysis, System Design & Architecture Documentation

---

END OF CLOUD MIGRATION STRATEGY