**Mingus Production Database Migration Plan**

**🚨 Phase 1: Critical Security & Performance Fixes (Week 1)**

**Immediate Actions Required**

**1. Security Hardening**

# Remove hard-coded secrets from config/development.py

# Move these to environment variables:

export SECRET\_KEY="$(openssl rand -base64 32)"

export SUPABASE\_URL="https://your-project.supabase.co"

export SUPABASE\_KEY="your-anon-key"

export SUPABASE\_SERVICE\_ROLE\_KEY="your-service-role-key"

export SUPABASE\_JWT\_SECRET="your-jwt-secret"

export DATABASE\_URL="postgresql://user:pass@localhost:5432/mingus\_prod"

**2. Database Migration Script**

-- Create production database

CREATE DATABASE mingus\_production;

-- Create necessary extensions

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE EXTENSION IF NOT EXISTS "pg\_stat\_statements";

-- Run the schema migration

\i migrations/001\_create\_base\_schema.sql

\i migrations/002\_add\_subscription\_tables.sql

\i migrations/003\_add\_performance\_indexes.sql

**3. Performance Indexes (Add Immediately)**

-- Critical missing indexes

CREATE INDEX CONCURRENTLY idx\_user\_profiles\_user\_id ON user\_profiles(user\_id);

CREATE INDEX CONCURRENTLY idx\_onboarding\_progress\_user\_id ON onboarding\_progress(user\_id);

CREATE INDEX CONCURRENTLY idx\_health\_checkins\_user\_date ON user\_health\_checkins(user\_id, checkin\_date);

CREATE INDEX CONCURRENTLY idx\_financial\_submissions\_user\_date ON financial\_questionnaire\_submissions(user\_id, submitted\_at);

CREATE INDEX CONCURRENTLY idx\_reminder\_schedules\_user\_due ON reminder\_schedules(user\_id, scheduled\_date);

**📊 Phase 2: Business Model Implementation (Week 2)**

**1. Subscription Management Tables**

-- Core subscription table

CREATE TABLE subscriptions (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

plan\_name VARCHAR(50) NOT NULL CHECK (plan\_name IN ('budget', 'mid-tier', 'professional')),

monthly\_price DECIMAL(10,2) NOT NULL,

status VARCHAR(20) NOT NULL DEFAULT 'active',

current\_period\_start TIMESTAMPTZ NOT NULL,

current\_period\_end TIMESTAMPTZ NOT NULL,

created\_at TIMESTAMPTZ DEFAULT NOW(),

updated\_at TIMESTAMPTZ DEFAULT NOW()

);

-- Feature access control

CREATE TABLE feature\_access\_log (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID NOT NULL REFERENCES users(id),

feature\_name VARCHAR(100) NOT NULL,

access\_count INTEGER DEFAULT 1,

access\_date DATE DEFAULT CURRENT\_DATE,

subscription\_plan VARCHAR(50) NOT NULL,

CONSTRAINT unique\_user\_feature\_date UNIQUE (user\_id, feature\_name, access\_date)

);

**2. Enhanced User Demographics**

-- Add target market specific fields

ALTER TABLE user\_profiles ADD COLUMN IF NOT EXISTS metro\_area VARCHAR(100);

ALTER TABLE user\_profiles ADD COLUMN IF NOT EXISTS years\_at\_current\_job INTEGER;

ALTER TABLE user\_profiles ADD COLUMN IF NOT EXISTS job\_security\_rating INTEGER;

ALTER TABLE user\_profiles ADD COLUMN IF NOT EXISTS student\_loan\_balance DECIMAL(12,2) DEFAULT 0;

ALTER TABLE user\_profiles ADD COLUMN IF NOT EXISTS dependents INTEGER DEFAULT 0;

-- Add constraints for data integrity

ALTER TABLE user\_profiles ADD CONSTRAINT valid\_job\_security CHECK (job\_security\_rating BETWEEN 1 AND 10);

ALTER TABLE user\_profiles ADD CONSTRAINT valid\_years\_job CHECK (years\_at\_current\_job >= 0);

**🔧 Phase 3: Performance Optimization (Week 3)**

**1. Query Optimization**

# Fix N+1 query problems

def get\_user\_dashboard\_data(user\_id):

return db.session.query(User).options(

joinedload(User.profile),

joinedload(User.onboarding\_progress),

joinedload(User.subscription)

).filter(User.id == user\_id).first()

# Optimize health check-in queries

def get\_recent\_health\_checkins(user\_id, limit=10):

return db.session.query(UserHealthCheckin)\

.filter(UserHealthCheckin.user\_id == user\_id)\

.order\_by(UserHealthCheckin.checkin\_date.desc())\

.limit(limit).all()

**2. Caching Strategy**

# Cache expensive calculations

@cache.memoize(timeout=1800) # 30 minutes

def get\_cash\_flow\_forecast(user\_id):

# Pre-calculate and cache 365-day forecast

return CashFlowService.calculate\_daily\_forecast(user\_id)

@cache.memoize(timeout=3600) # 1 hour

def get\_user\_financial\_summary(user\_id):

# Cache comprehensive financial summary

return FinancialSummaryService.generate\_summary(user\_id)

**3. Background Task Processing**

# Move heavy calculations to background

@celery.task

def calculate\_health\_spending\_correlations(user\_id):

# Correlation analysis in background

correlations = HealthAnalysisService.analyze\_correlations(user\_id)

cache.set(f"correlations\_{user\_id}", correlations, timeout=3600)

return correlations

@celery.task

def generate\_financial\_forecast(user\_id):

# 365-day forecast calculation

forecast = CashFlowService.calculate\_daily\_forecast(user\_id)

cache.set(f"forecast\_{user\_id}", forecast, timeout=1800)

return forecast

**🎯 Phase 4: Feature Access Control (Week 4)**

**1. Subscription Service**

class SubscriptionService:

def \_\_init\_\_(self, db\_session):

self.db = db\_session

def check\_feature\_access(self, user\_id, feature\_name):

"""Check if user can access a feature based on their subscription"""

subscription = self.db.query(Subscription).filter(

Subscription.user\_id == user\_id,

Subscription.status == 'active'

).first()

if not subscription:

return {'access': False, 'reason': 'no\_subscription'}

# Check feature limits

limit = self.db.query(FeatureLimit).filter(

FeatureLimit.plan\_name == subscription.plan\_name,

FeatureLimit.feature\_name == feature\_name

).first()

if not limit:

return {'access': False, 'reason': 'feature\_not\_available'}

# Check usage count

current\_usage = self.get\_monthly\_usage(user\_id, feature\_name)

if limit.limit\_value != -1 and current\_usage >= limit.limit\_value:

return {'access': False, 'reason': 'limit\_exceeded'}

return {'access': True, 'remaining': limit.limit\_value - current\_usage}

**2. Feature Decorators**

def requires\_subscription(feature\_name):

def decorator(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

user\_id = get\_current\_user\_id()

access = subscription\_service.check\_feature\_access(user\_id, feature\_name)

if not access['access']:

return jsonify({

'error': 'Feature access denied',

'reason': access['reason'],

'upgrade\_url': '/pricing'

}), 403

return f(\*args, \*\*kwargs)

return decorated\_function

return decorator

# Usage example

@app.route('/api/health/correlation-analysis')

@requires\_subscription('correlation\_analysis\_access')

def get\_correlation\_analysis():

# This endpoint requires mid-tier or professional subscription

return jsonify(correlation\_service.analyze\_user\_data(user\_id))

**💾 Phase 5: Data Migration Strategy**

**1. SQLite to PostgreSQL Migration**

def migrate\_sqlite\_to\_postgres():

# Connect to both databases

sqlite\_conn = sqlite3.connect('instance/mingus.db')

pg\_conn = psycopg2.connect(DATABASE\_URL)

# Migration order (respecting foreign keys)

migration\_order = [

'users',

'user\_profiles',

'user\_preferences',

'onboarding\_progress',

'user\_health\_checkins',

'financial\_questionnaire\_submissions',

'health\_spending\_correlations',

'reminder\_schedules',

'phone\_verification',

'verification\_analytics',

'verification\_audit\_log'

]

for table in migration\_order:

migrate\_table(sqlite\_conn, pg\_conn, table)

print(f"Migrated {table}")

# Add subscription data for existing users

add\_default\_subscriptions()

**2. Default Subscription Assignment**

def add\_default\_subscriptions():

# Give existing users a trial subscription

users = db.session.query(User).all()

for user in users:

subscription = Subscription(

user\_id=user.id,

plan\_name='budget', # Start with budget tier

monthly\_price=10.00,

status='trial',

current\_period\_start=datetime.utcnow(),

current\_period\_end=datetime.utcnow() + timedelta(days=30),

trial\_start=datetime.utcnow(),

trial\_end=datetime.utcnow() + timedelta(days=30),

is\_trial=True

)

db.session.add(subscription)

db.session.commit()

**🏃‍♂️ Quick Start Commands**

**1. Environment Setup**

# Create PostgreSQL database

createdb mingus\_production

# Set environment variables

export DATABASE\_URL="postgresql://user:password@localhost:5432/mingus\_production"

export SECRET\_KEY="your-secret-key"

export REDIS\_URL="redis://localhost:6379/1"

# Install dependencies

pip install psycopg2-binary redis celery

**2. Database Schema Creation**

# Run migrations

python scripts/create\_production\_schema.py

# Add indexes

python scripts/add\_performance\_indexes.py

# Verify schema

python scripts/verify\_schema.py

**3. Application Configuration**

# config/production.py

class ProductionConfig(Config):

DATABASE\_URL = os.environ.get('DATABASE\_URL')

REDIS\_URL = os.environ.get('REDIS\_URL')

SECRET\_KEY = os.environ.get('SECRET\_KEY')

# Security settings

SESSION\_COOKIE\_SECURE = True

WTF\_CSRF\_ENABLED = True

BYPASS\_AUTH = False

# Performance settings

SQLALCHEMY\_ENGINE\_OPTIONS = {

'pool\_size': 20,

'max\_overflow': 30,

'pool\_pre\_ping': True,

'pool\_recycle': 1800

}

**📊 Success Metrics**

**Technical Metrics**

* **Database Performance**: <100ms for 95% of queries
* **API Response Time**: <200ms for 95% of requests
* **Cache Hit Rate**: >80% for frequently accessed data
* **Database Connections**: <50% of pool utilization
* **Error Rate**: <0.1% of requests

**Business Metrics**

* **Free to Paid Conversion**: 15% target
* **Monthly Retention**: 80% target
* **Feature Usage**: 70% of users using core features
* **Revenue per User**: $18 average (mixed tier adoption)

**User Experience Metrics**

* **Onboarding Completion**: 85% target
* **Weekly Active Users**: 60% of subscribers
* **Feature Satisfaction**: 4.5/5 average rating
* **Support Ticket Volume**: <5% of users per month

**🎯 Next Steps**

1. **Immediate**: Fix security vulnerabilities and add missing indexes
2. **Week 1**: Complete PostgreSQL migration and basic subscription setup
3. **Week 2**: Implement feature access control and usage tracking
4. **Week 3**: Deploy to production with monitoring
5. **Week 4**: Launch with marketing to target audience

**Estimated Timeline**: 4-6 weeks to full production deployment **Estimated Cost**: $500-1000/month for infrastructure (1000 users) **Risk Level**: Medium (with proper testing and staged rollout)