MEDIVOY HEALTHCARE SYSTEM

Complete Backend API Development Plan (JavaScript/Node.js)

Ultra-Comprehensive Guide with Complete Workflows

Executive Summary

This document provides the most comprehensive backend API plan for the Medivoy Healthcare System. Every aspect from your system diagrams has been extracted and documented, including complete file structures, database schemas, API endpoints, workflows, code examples, and implementation strategies.

1. Complete Project Structure

The Medivoy backend follows a modular, scalable architecture using JavaScript/Node.js with Express.js framework.

Root Directory Organization

```
MEDIVOY.API/
├── .github/workflows/          # CI/CD pipelines
├── docs/                       # All documentation
├── logs/                       # Application logs
├── migrations/                 # Database version control
├── scripts/                    # Utility scripts
├── src/                        # Source code
├── tests/                      # Test suites
├── uploads/                    # File storage
└── Configuration files
```

Source Code Structure (src/)

Configuration Files (src/config/)

- **index.js** - Main configuration aggregator
- **database.js** - PostgreSQL connection settings
- **mongodb.js** - MongoDB configuration
- **redis.js** - Redis cache configuration
- **aws.js** - AWS S3 storage settings
- **twilio.js** - SMS/WhatsApp configuration

- **firebase.js** - Push notifications & chat
- **stripe.js** - Stripe payment gateway
- **razorpay.js** - Razorpay payment gateway
- **sendgrid.js** - Email service configuration
- **nextcloud.js** - Video/audio call settings
- **cors.js** - CORS policy configuration

Constants (src/constants/)

- **status-codes.js** - Booking and appointment status workflows
- **user-roles.js** - Role definitions (admin, doctor, patient, hospital_admin)
- **error-codes.js** - Custom error code mappings
- **file-types.js** - Allowed file upload types
- **locales.js** - Supported languages (en, ar, hi, etc.)

Controllers (src/controllers/) - 26 Controllers

1. **auth.controller.js** - Login, registration, password reset
2. **users.controller.js** - User CRUD operations
3. **hospitals.controller.js** - Hospital/clinic management
4. **doctors.controller.js** - Doctor profiles and schedules
5. **patients.controller.js** - Patient records and profiles
6. **treatments.controller.js** - Treatment catalog management
7. **packages.controller.js** - Medical tour packages
8. **bookings.controller.js** - Booking lifecycle management
9. **appointments.controller.js** - Appointment scheduling
10. **medical-records.controller.js** - Medical document management
11. **prescriptions.controller.js** - Prescription handling
12. **laboratories.controller.js** - Lab facility management
13. **lab-tests.controller.js** - Lab test requests
14. **insurance.controller.js** - Insurance provider management
15. **payments.controller.js** - Payment processing
16. **invoices.controller.js** - Invoice generation
17. **reviews.controller.js** - Reviews and ratings
18. **notifications.controller.js** - Notification dispatch
19. **support.controller.js** - Support ticket system
20. **subscriptions.controller.js** - Subscription management

21. **translations.controller.js** - Multi-language support
22. **analytics.controller.js** - Analytics and reporting
23. **dashboard.controller.js** - Dashboard data aggregation
24. **media.controller.js** - Media asset management
25. **coupons.controller.js** - Discount code management
26. **faqs.controller.js** - FAQ management

Database Layer (src/db/)

PostgreSQL (src/db/postgres/)

- **connection.js** - Database connection pool
- **pool.js** - Connection pooling strategy
- **queries.js** - Query builder utilities

MongoDB (src/db/mongodb/)

- **connection.js** - MongoDB connection
- **schemas/** - Mongoose schemas for audit logs, analytics, sessions
- **models/** - MongoDB models

Redis (src/db/redis/)

- **connection.js** - Redis client setup
- **cache.js** - Caching operations
- **queue.js** - Bull queue for background jobs

Migrations (src/db/migrations/) - 30 Migration Files

1. 001-create-users.sql
2. 002-create-patients.sql
3. 003-create-doctors.sql
4. 004-create-hospitals.sql
5. 005-create-hospital-doctors.sql (junction table)
6. 006-create-treatments.sql
7. 007-create-hospital-treatments.sql (junction table)
8. 008-create-packages.sql
9. 009-create-bookings.sql
10. 010-create-appointments.sql
11. 011-create-medical-records.sql

12. 012-create-prescriptions.sql
13. 013-create-laboratories.sql
14. 014-create-lab-tests.sql
15. 015-create-insurance.sql
16. 016-create-payments.sql
17. 017-create-invoices.sql
18. 018-create-reviews.sql
19. 019-create-notifications.sql
20. 020-create-support-tickets.sql
21. 021-create-subscription-plans.sql
22. 022-create-subscriptions.sql
23. 023-create-translations.sql
24. 024-create-coupons.sql
25. 025-create-faqs.sql
26. 026-create-website-content.sql
27. 027-create-media.sql
28. 028-create-password-resets.sql
29. 029-create-refresh-tokens.sql
30. 030-create-indexes.sql

Background Jobs (src/jobs/) - 11 Job Files

1. **email.job.js** - Email queue worker
2. **sms.job.js** - SMS queue worker
3. **notification.job.js** - Push notification queue
4. **translation.job.js** - Auto-translation worker
5. **backup.job.js** - Database backup scheduler
6. **cleanup.job.js** - Temporary file cleanup
7. **analytics.job.js** - Analytics data aggregation
8. **appointment-reminder.job.js** - Appointment reminders
9. **payment-reminder.job.js** - Payment due reminders
10. **subscription-renewal.job.js** - Subscription renewal
11. **queue.js** - Bull queue management

Middleware (src/middleware/) - 12 Middleware Files

1. **auth.middleware.js** - JWT token verification
2. **authorize.middleware.js** - Role-based access control
3. **validate.middleware.js** - Input validation
4. **cache.middleware.js** - Response caching
5. **rate-limit.middleware.js** - API rate limiting
6. **error.middleware.js** - Global error handler
7. **logger.middleware.js** - Request/response logging
8. **security.middleware.js** - Security headers (helmet)
9. **cors.middleware.js** - CORS policy enforcement
10. **upload.middleware.js** - Multer file upload
11. **audit.middleware.js** - Audit trail logging
12. **locale.middleware.js** - Language detection

Models (src/models/) - 28 Model Files

All PostgreSQL table models with CRUD operations:

1. User.model.js
2. Patient.model.js
3. Doctor.model.js
4. Hospital.model.js
5. HospitalDoctor.model.js
6. Treatment.model.js
7. HospitalTreatment.model.js
8. Package.model.js
9. Booking.model.js
10. Appointment.model.js
11. MedicalRecord.model.js
12. Prescription.model.js
13. Laboratory.model.js
14. LabTest.model.js
15. Insurance.model.js
16. Payment.model.js
17. Invoice.model.js
18. Review.model.js

19. Notification.model.js
20. SupportTicket.model.js
21. SubscriptionPlan.model.js
22. Subscription.model.js
23. Translation.model.js
24. Coupon.model.js
25. FAQ.model.js
26. WebsiteContent.model.js
27. Media.model.js
28. PasswordReset.model.js
29. RefreshToken.model.js

Routes (src/routes/v1/) - 26 Route Files

1. auth.routes.js
2. users.routes.js
3. hospitals.routes.js
4. doctors.routes.js
5. patients.routes.js
6. treatments.routes.js
7. packages.routes.js
8. bookings.routes.js
9. appointments.routes.js
10. medical-records.routes.js
11. prescriptions.routes.js
12. laboratories.routes.js
13. lab-tests.routes.js
14. insurance.routes.js
15. payments.routes.js
16. invoices.routes.js
17. reviews.routes.js
18. notifications.routes.js
19. support.routes.js
20. subscriptions.routes.js
21. translations.routes.js
22. analytics.routes.js

23. dashboard.routes.js
24. media.routes.js
25. coupons.routes.js
26. faqs.routes.js

Webhook Routes (src/routes/webhooks/)

1. stripe.webhook.js
2. razorpay.webhook.js
3. twilio.webhook.js

Services (src/services/) - 26 Service Files

Business logic layer:

1. auth.service.js
2. user.service.js
3. hospital.service.js
4. doctor.service.js
5. patient.service.js
6. treatment.service.js
7. package.service.js
8. booking.service.js
9. appointment.service.js
10. medical-record.service.js
11. prescription.service.js
12. laboratory.service.js
13. lab-test.service.js
14. insurance.service.js
15. payment.service.js
16. invoice.service.js
17. review.service.js
18. notification.service.js
19. support.service.js
20. subscription.service.js
21. translation.service.js
22. analytics.service.js
23. dashboard.service.js

24. email.service.js
25. sms.service.js
26. push.service.js
27. upload.service.js
28. cache.service.js
29. video-call.service.js
30. audit.service.js

Validators (src/validators/) - 20 Validator Files

Input validation schemas using Joi:

1. auth.validator.js
2. user.validator.js
3. hospital.validator.js
4. doctor.validator.js
5. patient.validator.js
6. treatment.validator.js
7. package.validator.js
8. booking.validator.js
9. appointment.validator.js
10. medical-record.validator.js
11. prescription.validator.js
12. laboratory.validator.js
13. lab-test.validator.js
14. insurance.validator.js
15. payment.validator.js
16. invoice.validator.js
17. review.validator.js
18. notification.validator.js
19. support.validator.js
20. subscription.validator.js

Utilities (src/utils/) - 15 Utility Files

1. app.js - Express application setup
2. server.js - Server initialization
3. logger.js - Winston logger configuration
4. validators.js - Common validation functions
5. helpers.js - General helper functions
6. encryption.js - bcrypt hashing utilities
7. jwt.js - JWT generation and verification
8. date.js - Date manipulation utilities
9. string.js - String manipulation helpers
10. number.js - Number formatting utilities
11. email-templates.js - HTML email templates
12. pdf-generator.js - PDF document generation
13. error-handler.js - Custom error classes
14. response-formatter.js - Standardized API responses
15. swagger.js - OpenAPI documentation

2. Complete Database Schema

PostgreSQL Tables (Primary Database)

1. users Table

```
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20) UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    user_type VARCHAR(50) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    profile_picture TEXT,
    is_verified BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    last_login TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

User Types: admin, hospital_admin, doctor, patient

2. patients Table

```
CREATE TABLE patients (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    date_of_birth DATE,
    gender VARCHAR(20),
    blood_group VARCHAR(10),
    country VARCHAR(100),
    city VARCHAR(100),
    address TEXT,
    emergency_contact_name VARCHAR(100),
    emergency_contact_phone VARCHAR(20),
    insurance_provider VARCHAR(255),
    insurance_policy_number VARCHAR(100),
    medical_history JSONB,
    allergies JSONB,
    current_medications JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. hospitals Table

```
CREATE TABLE hospitals (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(255) NOT NULL,
    type VARCHAR(50),
    description TEXT,
    logo TEXT,
    country VARCHAR(100),
    city VARCHAR(100),
    address TEXT,
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    phone VARCHAR(20),
    email VARCHAR(255),
    website TEXT,
    certifications JSONB,
    specializations JSONB,
    is_verified BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    admin_user_id UUID REFERENCES users(id),
    bank_details JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4. doctors Table

```
CREATE TABLE doctors (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    specialty VARCHAR(100),
    qualification TEXT,
    experience_years INTEGER,
    license_number VARCHAR(100),
    bio TEXT,
    consultation_fee DECIMAL(10, 2),
    languages JSONB,
    availability_slots JSONB,
    is_available_teleconsult BOOLEAN DEFAULT TRUE,
    rating DECIMAL(3, 2) DEFAULT 0.00,
    total_reviews INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5. hospital_doctors Table (Junction)

```
CREATE TABLE hospital_doctors (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    hospital_id UUID REFERENCES hospitals(id) ON DELETE CASCADE,
    doctor_id UUID REFERENCES doctors(id) ON DELETE CASCADE,
    department VARCHAR(100),
    is_primary BOOLEAN DEFAULT FALSE,
    start_date DATE,
    end_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(hospital_id, doctor_id)
);
```

6. treatments Table

```
CREATE TABLE treatments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(255) NOT NULL,
    category VARCHAR(100),
    description TEXT,
    duration_days INTEGER,
    is_global BOOLEAN DEFAULT TRUE,
    image TEXT,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

7. hospital_treatments Table (Junction)

```
CREATE TABLE hospital_treatments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    hospital_id UUID REFERENCES hospitals(id) ON DELETE CASCADE,
    treatment_id UUID REFERENCES treatments(id) ON DELETE CASCADE,
    base_price DECIMAL(10, 2),
    currency VARCHAR(3) DEFAULT 'USD',
    duration_days INTEGER,
    success_rate DECIMAL(5, 2),
    description TEXT,
    inclusions JSONB,
    exclusions JSONB,
    is_available BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

8. packages Table

```
CREATE TABLE packages (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(255) NOT NULL,
    country VARCHAR(100),
    description TEXT,
    duration_days INTEGER,
    base_price DECIMAL(10, 2),
    currency VARCHAR(3) DEFAULT 'USD',
    inclusions JSONB,
    hospital_ids JSONB,
    treatment_ids JSONB,
    flights_included BOOLEAN DEFAULT FALSE,
    accommodation_included BOOLEAN DEFAULT FALSE,
    transfers_included BOOLEAN DEFAULT FALSE,
    seasonal_pricing JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

9. bookings Table

```
CREATE TABLE bookings (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    booking_number VARCHAR(50) UNIQUE NOT NULL,
    patient_id UUID REFERENCES patients(id),
    hospital_id UUID REFERENCES hospitals(id),
    treatment_id UUID REFERENCES treatments(id),
    package_id UUID REFERENCES packages(id),
    booking_type VARCHAR(50),
    status VARCHAR(50),
    requested_date DATE,
```

```
    confirmed_date DATE,
    completion_date DATE,
    total_amount DECIMAL(10, 2),
    currency VARCHAR(3),
    payment_status VARCHAR(50),
    medical_details JSONB,
    quotation_details JSONB,
    travel_details JSONB,
    notes TEXT,
    coordinator_id UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Booking Status Flow:

- requested
- under_review
- accepted/rejected
- awaiting_medical_details
- quotation_sent
- confirmed/on_hold/cancelled
- payment_pending/payment_completed
- invoice_sent
- travel_arrangement
- in_treatment
- completed
- feedback_received

10. appointments Table

```
CREATE TABLE appointments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    appointment_number VARCHAR(50) UNIQUE NOT NULL,
    patient_id UUID REFERENCES patients(id),
    doctor_id UUID REFERENCES doctors(id),
    booking_id UUID REFERENCES bookings(id),
    appointment_type VARCHAR(50),
    status VARCHAR(50),
    scheduled_date TIMESTAMP,
    duration_minutes INTEGER DEFAULT 30,
    consultation_fee DECIMAL(10, 2),
    currency VARCHAR(3),
    chief_complaint TEXT,
    diagnosis TEXT,
    prescription JSONB,
    follow_up_date DATE,
    video_call_link TEXT,
```

```
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Appointment Status Flow:

- requested/booked
- appointment_confirmed
- awaiting_consultation
- consultation_in_progress
- prescription_provided
- follow_up_scheduled (optional)
- consultation_completed
- cancelled

11. medical_records Table

```
CREATE TABLE medical_records (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID REFERENCES patients(id),
    appointment_id UUID REFERENCES appointments(id),
    record_type VARCHAR(50),
    title VARCHAR(255),
    description TEXT,
    file_url TEXT,
    file_size INTEGER,
    file_type VARCHAR(50),
    uploaded_by_user_id UUID REFERENCES users(id),
    record_date DATE,
    is_shared_with_doctors BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Record Types: xray, mri, ct_scan, lab_report, prescription, discharge_summary, other

12. prescriptions Table

```
CREATE TABLE prescriptions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    appointment_id UUID REFERENCES appointments(id),
    patient_id UUID REFERENCES patients(id),
    doctor_id UUID REFERENCES doctors(id),
    medications JSONB,
    instructions TEXT,
    valid_until DATE,
    pdf_url TEXT,
```

```
    is_dispensed BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

13. laboratories Table

```
CREATE TABLE laboratories (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(255) NOT NULL,
    hospital_id UUID REFERENCES hospitals(id),
    type VARCHAR(100),
    country VARCHAR(100),
    city VARCHAR(100),
    address TEXT,
    phone VARCHAR(20),
    email VARCHAR(255),
    services_offered JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

14. lab_tests Table

```
CREATE TABLE lab_tests (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    test_number VARCHAR(50) UNIQUE NOT NULL,
    patient_id UUID REFERENCES patients(id),
    doctor_id UUID REFERENCES doctors(id),
    lab_id UUID REFERENCES laboratories(id),
    test_type VARCHAR(100),
    test_name VARCHAR(255),
    scheduled_date DATE,
    status VARCHAR(50),
    result_url TEXT,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

15. insurance Table

```
CREATE TABLE insurance (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    provider_name VARCHAR(255) NOT NULL,
    country VARCHAR(100),
    logo TEXT,
    contact_phone VARCHAR(20),
    contact_email VARCHAR(255),
    website TEXT,
```

```
    coverage_details JSONB,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

16. payments Table

```
CREATE TABLE payments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    transaction_id VARCHAR(100) UNIQUE,
    booking_id UUID REFERENCES bookings(id),
    appointment_id UUID REFERENCES appointments(id),
    patient_id UUID REFERENCES patients(id),
    amount DECIMAL(10, 2),
    currency VARCHAR(3),
    payment_method VARCHAR(50),
    payment_gateway VARCHAR(50),
    gateway_transaction_id VARCHAR(255),
    status VARCHAR(50),
    payment_date TIMESTAMP,
    refund_amount DECIMAL(10, 2),
    refund_date TIMESTAMP,
    metadata JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

17. invoices Table

```
CREATE TABLE invoices (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    invoice_number VARCHAR(50) UNIQUE NOT NULL,
    booking_id UUID REFERENCES bookings(id),
    appointment_id UUID REFERENCES appointments(id),
    patient_id UUID REFERENCES patients(id),
    hospital_id UUID REFERENCES hospitals(id),
    total_amount DECIMAL(10, 2),
    tax_amount DECIMAL(10, 2),
    discount_amount DECIMAL(10, 2),
    final_amount DECIMAL(10, 2),
    currency VARCHAR(3),
    line_items JSONB,
    invoice_date DATE,
    due_date DATE,
    pdf_url TEXT,
    status VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

18. reviews Table

```
CREATE TABLE reviews (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID REFERENCES patients(id),
    doctor_id UUID REFERENCES doctors(id),
    hospital_id UUID REFERENCES hospitals(id),
    appointment_id UUID REFERENCES appointments(id),
    booking_id UUID REFERENCES bookings(id),
    rating INTEGER CHECK (rating &gt;= 1 AND rating &lt;= 5),
    review_text TEXT,
    is_verified BOOLEAN DEFAULT FALSE,
    is_published BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

19. notifications Table

```
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id),
    title VARCHAR(255),
    message TEXT,
    type VARCHAR(50),
    reference_type VARCHAR(50),
    reference_id UUID,
    is_read BOOLEAN DEFAULT FALSE,
    is_sent BOOLEAN DEFAULT FALSE,
    channel VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

20. support_tickets Table

```
CREATE TABLE support_tickets (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    ticket_number VARCHAR(50) UNIQUE NOT NULL,
    user_id UUID REFERENCES users(id),
    subject VARCHAR(255),
    description TEXT,
    category VARCHAR(100),
    priority VARCHAR(50),
    status VARCHAR(50),
    assigned_to_user_id UUID REFERENCES users(id),
    sla_due_date TIMESTAMP,
    resolution_notes TEXT,
    resolved_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

21. subscription_plans Table

```
CREATE TABLE subscription_plans (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL,
    plan_type VARCHAR(50),
    target_user VARCHAR(50),
    price_monthly DECIMAL(10, 2),
    price_yearly DECIMAL(10, 2),
    currency VARCHAR(3),
    features JSONB,
    max_doctors INTEGER,
    max_appointments_per_month INTEGER,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

22. subscriptions Table

```
CREATE TABLE subscriptions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    plan_id UUID REFERENCES subscription_plans(id),
    hospital_id UUID REFERENCES hospitals(id),
    doctor_id UUID REFERENCES doctors(id),
    start_date DATE,
    end_date DATE,
    billing_cycle VARCHAR(50),
    status VARCHAR(50),
    auto_renew BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

23. translations Table

```
CREATE TABLE translations (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    entity_type VARCHAR(100),
    entity_id UUID,
    field_name VARCHAR(100),
    locale VARCHAR(10),
    translated_text TEXT,
    is_auto_translated BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

24. coupons Table

```
CREATE TABLE coupons (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    code VARCHAR(50) UNIQUE NOT NULL,
    discount_type VARCHAR(50),
    discount_value DECIMAL(10, 2),
    min_purchase_amount DECIMAL(10, 2),
    max_discount_amount DECIMAL(10, 2),
    valid_from DATE,
    valid_until DATE,
    usage_limit INTEGER,
    usage_count INTEGER DEFAULT 0,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

25. faqs Table

```
CREATE TABLE faqs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    question TEXT NOT NULL,
    answer TEXT NOT NULL,
    category VARCHAR(100),
    display_order INTEGER,
    is_published BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

26. website_content Table

```
CREATE TABLE website_content (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    page_slug VARCHAR(100) UNIQUE NOT NULL,
    title VARCHAR(255),
    content TEXT,
    meta_description TEXT,
    meta_keywords TEXT,
    is_published BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

27. media Table

```
CREATE TABLE media (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    file_name VARCHAR(255),
    file_url TEXT,
    file_type VARCHAR(50),
    file_size INTEGER,
    entity_type VARCHAR(100),
    entity_id UUID,
    uploaded_by_user_id UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

28. password_resets Table

```
CREATE TABLE password_resets (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id),
    token VARCHAR(255) UNIQUE NOT NULL,
    expires_at TIMESTAMP,
    is_used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

29. refresh_tokens Table

```
CREATE TABLE refresh_tokens (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id),
    token TEXT UNIQUE NOT NULL,
    expires_at TIMESTAMP,
    is_revoked BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

MongoDB Collections (Secondary Database)

audit_logs Collection

```
{
    _id: ObjectId,
    user_id: String,
    action: String,
    entity_type: String,
    entity_id: String,
    changes: Object,
    ip_address: String,
    user_agent: String,
```

```
    timestamp: Date
}
```

analytics Collection

```
{
    _id: ObjectId,
    metric_type: String,
    date: Date,
    hospital_id: String,
    data: Object,
    created_at: Date
}
```

sessions Collection

```
{
    _id: ObjectId,
    user_id: String,
    session_token: String,
    device_info: Object,
    ip_address: String,
    last_activity: Date,
    created_at: Date,
    expires_at: Date
}
```

3. Complete API Endpoints

Authentication & Authorization (auth.routes.js)

POST /api/v1/auth/register

Register a new user account

Request Body:

```
{
    "email": "user@example.com",
    "password": "SecurePassword123!",
    "phone": "+1234567890",
    "user_type": "patient",
    "first_name": "John",
    "last_name": "Doe"
}
```

Response:

```
{
    "success": true,
    "message": "User registered successfully",
    "data": {
        "user": {
            "id": "uuid",
            "email": "user@example.com",
            "user_type": "patient"
        },
        "access_token": "jwt_token",
        "refresh_token": "refresh_token"
    }
}
```

POST /api/v1/auth/login

Authenticate user and get tokens

Request Body:

```
{
    "email": "user@example.com",
    "password": "SecurePassword123!"
}
```

Response:

```
{
    "success": true,
    "message": "Login successful",
    "data": {
        "user": {
            "id": "uuid",
            "email": "user@example.com",
            "user_type": "patient",
            "profile": {}
        },
        "access_token": "jwt_token",
        "refresh_token": "refresh_token"
    }
}
```

POST /api/v1/auth/refresh

Refresh access token using refresh token

POST /api/v1/auth/logout

Logout user and revoke tokens

POST /api/v1/auth/forgot-password

Request password reset email

POST /api/v1/auth/reset-password

Reset password using token

GET /api/v1/auth/profile

Get current user profile (authenticated)

PATCH /api/v1/auth/profile

Update current user profile (authenticated)

POST /api/v1/auth/verify-email

Verify email address

POST /api/v1/auth/resend-verification

Resend verification email

User Management (users.routes.js)

GET /api/v1/users

Get all users (admin only)

- Query params: page, limit, user_type, is_active, search

GET /api/v1/users/:id

Get user by ID

POST /api/v1/users

Create new user (admin only)

PATCH /api/v1/users/:id

Update user

DELETE /api/v1/users/:id

Delete user (admin only)

PATCH /api/v1/users/:id/activate

Activate user account

PATCH /api/v1/users/:id/deactivate

Deactivate user account

Hospitals (hospitals.routes.js)

GET /api/v1/hospitals

Get all hospitals

- Query params: page, limit, country, city, specialization, is_verified

GET /api/v1/hospitals/:id

Get hospital by ID

POST /api/v1/hospitals

Create hospital (admin only)

PATCH /api/v1/hospitals/:id

Update hospital

DELETE /api/v1/hospitals/:id

Delete hospital

GET /api/v1/hospitals/:id/doctors

Get all doctors in a hospital

POST /api/v1/hospitals/:id/doctors

Add doctor to hospital

DELETE /api/v1/hospitals/:id/doctors/:doctorId

Remove doctor from hospital

GET /api/v1/hospitals/:id/treatments

Get all treatments offered by hospital

POST /api/v1/hospitals/:id/treatments

Add treatment to hospital

GET /api/v1/hospitals/:id/analytics

Get hospital analytics (hospital admin only)

PATCH /api/v1/hospitals/:id/verify

Verify hospital (admin only)

Doctors (doctors.routes.js)

GET /api/v1/doctors

Get all doctors

- Query params: page, limit, specialty, hospital_id, is_available

GET /api/v1/doctors/:id

Get doctor by ID

POST /api/v1/doctors

Create doctor profile

PATCH /api/v1/doctors/:id

Update doctor profile

DELETE /api/v1/doctors/:id

Delete doctor

GET /api/v1/doctors/:id/appointments

Get doctor's appointments

GET /api/v1/doctors/:id/schedule

Get doctor's schedule

PATCH /api/v1/doctors/:id/availability

Update doctor availability slots

GET /api/v1/doctors/:id/reviews

Get doctor reviews

GET /api/v1/doctors/:id/patients

Get doctor's patients

Patients (patients.routes.js)

GET /api/v1/patients

Get all patients (admin/hospital only)

GET /api/v1/patients/:id

Get patient by ID

POST /api/v1/patients

Create patient profile

PATCH /api/v1/patients/:id

Update patient profile

DELETE /api/v1/patients/:id

Delete patient

GET /api/v1/patients/:id/appointments

Get patient's appointments

GET /api/v1/patients/:id/bookings

Get patient's bookings

GET /api/v1/patients/:id/medical-records

Get patient's medical records

GET /api/v1/patients/:id/prescriptions

Get patient's prescriptions

GET /api/v1/patients/:id/payments

Get patient's payment history

Treatments (treatments.routes.js)

GET /api/v1/treatments

Get all treatments

- Query params: page, limit, category, is_active, search

GET /api/v1/treatments/:id

Get treatment by ID

POST /api/v1/treatments

Create treatment (admin only)

PATCH /api/v1/treatments/:id

Update treatment

DELETE /api/v1/treatments/:id

Delete treatment

GET /api/v1/treatments/:id/hospitals

Get hospitals offering this treatment

Packages (packages.routes.js)

GET /api/v1/packages

Get all packages

- Query params: page, limit, country, duration, price_min, price_max

GET /api/v1/packages/:id

Get package by ID

POST /api/v1/packages

Create package (admin only)

PATCH /api/v1/packages/:id

Update package

DELETE /api/v1/packages/:id

Delete package

GET /api/v1/packages/by-country/:country

Get packages by country

Bookings (bookings.routes.js)

GET /api/v1/bookings

Get all bookings

- Query params: page, limit, status, patient_id, hospital_id

GET /api/v1/bookings/:id

Get booking by ID

POST /api/v1/bookings

Create new booking

Request Body:

```
{
    "patient_id": "uuid",
    "hospital_id": "uuid",
    "treatment_id": "uuid",
    "package_id": "uuid",
    "booking_type": "treatment",
    "requested_date": "2025-11-15",
    "medical_details": {},
    "notes": "Special requirements"
}
```

PATCH /api/v1/bookings/:id

Update booking

DELETE /api/v1/bookings/:id

Cancel booking

PATCH /api/v1/bookings/:id/status

Update booking status

Request Body:

```
{
    "status": "confirmed",
    "notes": "Booking confirmed"
}
```

POST /api/v1/bookings/:id/quotation

Submit quotation for booking

GET /api/v1/bookings/:id/timeline

Get booking timeline/history

Appointments (appointments.routes.js)

GET /api/v1/appointments

Get all appointments

- Query params: page, limit, status, patient_id, doctor_id, date

GET /api/v1/appointments/:id

Get appointment by ID

POST /api/v1/appointments

Create appointment

Request Body:

```
{
    "patient_id": "uuid",
    "doctor_id": "uuid",
    "appointment_type": "teleconsult",
    "scheduled_date": "2025-11-15T10:00:00Z",
    "duration_minutes": 30,
    "chief_complaint": "Headache"
}
```

PATCH /api/v1/appointments/:id

Update appointment

DELETE /api/v1/appointments/:id

Cancel appointment

PATCH /api/v1/appointments/:id/status

Update appointment status

POST /api/v1/appointments/:id/reschedule

Reschedule appointment

Request Body:

```
{
    "new_scheduled_date": "2025-11-16T10:00:00Z",
    "reason": "Patient request"
}
```

GET /api/v1/appointments/:id/video-call

Get video call link for appointment

POST /api/v1/appointments/:id/prescription

Add prescription to appointment

Medical Records (medical-records.routes.js)

GET /api/v1/medical-records

Get all medical records

- Query params: patient_id, record_type, date_from, date_to

GET /api/v1/medical-records/:id

Get medical record by ID

POST /api/v1/medical-records

Upload medical record

Request Body (multipart/form-data):

```
{
    "patient_id": "uuid",
    "record_type": "xray",
    "title": "Chest X-Ray",
    "description": "Routine checkup",
    "record_date": "2025-11-01",
    "file": File
}
```

PATCH /api/v1/medical-records/:id

Update medical record

DELETE /api/v1/medical-records/:id

Delete medical record

GET /api/v1/medical-records/:id/download

Download medical record file

Prescriptions (prescriptions.routes.js)

GET /api/v1/prescriptions

Get all prescriptions

- Query params: patient_id, doctor_id, appointment_id

GET /api/v1/prescriptions/:id

Get prescription by ID

POST /api/v1/prescriptions

Create prescription

Request Body:

```
{
    "appointment_id": "uuid",
    "patient_id": "uuid",
    "doctor_id": "uuid",
    "medications": [
        {
            "name": "Paracetamol",
            "dosage": "500mg",
            "frequency": "Twice daily",
            "duration": "7 days"
        }
    ],
    "instructions": "Take after meals",
    "valid_until": "2025-12-15"
}
```

PATCH /api/v1/prescriptions/:id

Update prescription

DELETE /api/v1/prescriptions/:id

Delete prescription

GET /api/v1/prescriptions/:id/pdf

Generate and download prescription PDF

Laboratories (laboratories.routes.js)

GET /api/v1/laboratories

Get all laboratories

GET /api/v1/laboratories/:id

Get laboratory by ID

POST /api/v1/laboratories

Create laboratory

PATCH /api/v1/laboratories/:id

Update laboratory

DELETE /api/v1/laboratories/:id

Delete laboratory

Lab Tests (lab-tests.routes.js)

GET /api/v1/lab-tests

Get all lab tests

GET /api/v1/lab-tests/:id

Get lab test by ID

POST /api/v1/lab-tests

Create lab test request

PATCH /api/v1/lab-tests/:id

Update lab test

DELETE /api/v1/lab-tests/:id

Delete lab test

POST /api/v1/lab-tests/:id/result

Upload lab test result

Insurance (insurance.routes.js)

GET /api/v1/insurance

Get all insurance providers

GET /api/v1/insurance/:id

Get insurance by ID

POST /api/v1/insurance

Create insurance provider

PATCH /api/v1/insurance/:id

Update insurance provider

DELETE /api/v1/insurance/:id

Delete insurance provider

Payments (payments.routes.js)

GET /api/v1/payments

Get all payments

GET /api/v1/payments/:id

Get payment by ID

POST /api/v1/payments

Create payment

Request Body:

```
{
    "booking_id": "uuid",
    "amount": 1000.00,
    "currency": "USD",
    "payment_method": "card",
```

```
    "payment_gateway": "stripe"
}
```

POST /api/v1/payments/webhook

Payment gateway webhook handler

GET /api/v1/payments/:id/status

Check payment status

POST /api/v1/payments/:id/refund

Process refund

Invoices (invoices.routes.js)

GET /api/v1/invoices

Get all invoices

GET /api/v1/invoices/:id

Get invoice by ID

POST /api/v1/invoices

Generate invoice

GET /api/v1/invoices/:id/pdf

Download invoice PDF

POST /api/v1/invoices/:id/send

Send invoice via email

Reviews (reviews.routes.js)

GET /api/v1/reviews

Get all reviews

GET /api/v1/reviews/:id

Get review by ID

POST /api/v1/reviews

Create review

Request Body:

```
{
    "doctor_id": "uuid",
    "hospital_id": "uuid",
    "appointment_id": "uuid",
    "rating": 5,
    "review_text": "Excellent service"
}
```

PATCH /api/v1/reviews/:id

Update review

DELETE /api/v1/reviews/:id

Delete review

PATCH /api/v1/reviews/:id/verify

Verify review (admin only)

Notifications (notifications.routes.js)

GET /api/v1/notifications

Get user notifications

POST /api/v1/notifications

Create notification

PATCH /api/v1/notifications/:id/read

Mark as read

PATCH /api/v1/notifications/read-all

Mark all as read

DELETE /api/v1/notifications/:id

Delete notification

Support (support.routes.js)

GET /api/v1/support/tickets

Get all support tickets

GET /api/v1/support/tickets/:id

Get ticket by ID

POST /api/v1/support/tickets

Create support ticket

PATCH /api/v1/support/tickets/:id

Update ticket

POST /api/v1/support/tickets/:id/reply

Add reply to ticket

PATCH /api/v1/support/tickets/:id/close

Close ticket

Subscriptions (subscriptions.routes.js)

GET /api/v1/subscriptions

Get all subscriptions

GET /api/v1/subscriptions/:id

Get subscription by ID

POST /api/v1/subscriptions

Create subscription

PATCH /api/v1/subscriptions/:id

Update subscription

DELETE /api/v1/subscriptions/:id

Cancel subscription

GET /api/v1/subscription-plans

Get all subscription plans

Translations (translations.routes.js)

GET /api/v1/translations

Get translations

POST /api/v1/translations

Create translation

PATCH /api/v1/translations/:id

Update translation

POST /api/v1/translations/auto-translate

Trigger auto-translation job

Analytics (analytics.routes.js)

GET /api/v1/analytics/overview

Get dashboard overview

GET /api/v1/analytics/bookings

Get booking analytics

GET /api/v1/analytics/revenue

Get revenue analytics

GET /api/v1/analytics/patients

Get patient analytics

GET /api/v1/analytics/doctors

Get doctor analytics

GET /api/v1/analytics/hospitals

Get hospital analytics

Dashboard (dashboard.routes.js)

GET /api/v1/dashboard/admin

Admin dashboard data

GET /api/v1/dashboard/hospital

Hospital dashboard data

GET /api/v1/dashboard/doctor

Doctor dashboard data

GET /api/v1/dashboard/patient

Patient dashboard data

Media (media.routes.js)

GET /api/v1/media

Get all media files

GET /api/v1/media/:id

Get media by ID

DELETE /api/v1/media/:id

Delete media file

Coupons (coupons.routes.js)

GET /api/v1/coupons

Get all coupons

GET /api/v1/coupons/:code

Validate coupon code

POST /api/v1/coupons

Create coupon

PATCH /api/v1/coupons/:id

Update coupon

DELETE /api/v1/coupons/:id

Delete coupon

FAQs (faqs.routes.js)

GET /api/v1/faqs

Get all FAQs

GET /api/v1/faqs/:id

Get FAQ by ID

POST /api/v1/faqs

Create FAQ

PATCH /api/v1/faqs/:id

Update FAQ

DELETE /api/v1/faqs/:id

Delete FAQ

Website (website.routes.js)

GET /api/v1/website/:slug

Get page content

POST /api/v1/website

Create page

PATCH /api/v1/website/:id

Update page

DELETE /api/v1/website/:id

Delete page

Upload (upload.routes.js)

POST /api/v1/upload/image

Upload image to S3

POST /api/v1/upload/document

Upload document to S3

POST /api/v1/upload/medical-record

Upload medical record

Health (health.routes.js)

GET /api/v1/health

API health check

GET /api/v1/health/db

Database health check

GET /api/v1/health/redis

Redis health check

GET /api/v1/health/services

Third-party services health

4. Complete Workflows

Booking Workflow

Step 1: Patient Creates Booking

```
POST /api/v1/bookings
Status: requested
```

Step 2: Admin Reviews Booking

```
PATCH /api/v1/bookings/:id/status
Status: under_review
```

Step 3: Admin Accepts/Rejects

```
PATCH /api/v1/bookings/:id/status
Status: accepted OR rejected
```

Step 4: Request Medical Details

```
PATCH /api/v1/bookings/:id/status
Status: awaiting_medical_details
Patient uploads: POST /api/v1/medical-records
```

Step 5: Send Quotation

```
POST /api/v1/bookings/:id/quotation
PATCH /api/v1/bookings/:id/status
Status: quotation_sent
```

Step 6: Confirm Booking

```
PATCH /api/v1/bookings/:id/status
Status: confirmed
```

Step 7: Process Payment

```
POST /api/v1/payments
PATCH /api/v1/bookings/:id/status
Status: payment_completed
```

Step 8: Generate Invoice

```
POST /api/v1/invoices
PATCH /api/v1/bookings/:id/status
Status: invoice_sent
```

Step 9: Arrange Travel

```
PATCH /api/v1/bookings/:id/status
Status: travel_arrangement
```

Step 10: Start Treatment

```
PATCH /api/v1/bookings/:id/status
Status: in_treatment
```

Step 11: Complete Treatment

```
PATCH /api/v1/bookings/:id/status
Status: completed
```

Step 12: Request Feedback

```
PATCH /api/v1/bookings/:id/status
Status: feedback_received
Patient creates: POST /api/v1/reviews
```

Appointment Workflow

Step 1: Request Appointment

```
POST /api/v1/appointments
Status: requested/booked
Send notification to doctor
```

Step 2: Confirm Appointment

```
PATCH /api/v1/appointments/:id/status
Status: appointment_confirmed
Send confirmation email/SMS to patient
```

Step 3: Before Consultation

```
PATCH /api/v1/appointments/:id/status
Status: awaiting_consultation
Send reminder notifications
```

Step 4: Start Consultation

```
PATCH /api/v1/appointments/:id/status
Status: consultation_in_progress
GET /api/v1/appointments/:id/video-call
```

Step 5: Provide Prescription

```
POST /api/v1/prescriptions
PATCH /api/v1/appointments/:id/status
Status: prescription_provided
```

Step 6: Schedule Follow-up (Optional)

```
POST /api/v1/appointments (new appointment)
PATCH /api/v1/appointments/:id/status
Status: follow_up_scheduled
```

Step 7: Complete Consultation

```
PATCH /api/v1/appointments/:id/status
Status: consultation_completed
POST /api/v1/invoices
Send invoice to patient
```

Payment Workflow

Step 1: Create Payment Intent

```
POST /api/v1/payments
{
    "booking_id": "uuid",
    "amount": 1000,
    "payment_gateway": "stripe"
}
Status: pending
```

Step 2: Process Payment

```
Stripe/Razorpay processes payment
Webhook: POST /api/v1/payments/webhook
Status: processing
```

Step 3: Payment Success

```
Update payment status: completed
Update booking payment_status: payment_completed
POST /api/v1/invoices (generate invoice)
Send confirmation email
```

Step 4: Payment Failure

```
Update payment status: failed
Send notification to patient
Retry payment option
```

Translation Workflow

Step 1: Create Content

```
POST /api/v1/treatments
Default language: en
```

Step 2: Trigger Translation

```
Background job: translation.job.js
CRON job detects new content
```

Step 3: Auto-Translate

```
POST /api/v1/translations
Translate to: ar, hi, es, fr
is_auto_translated: true
```

Step 4: Manual Review (Optional)

```
PATCH /api/v1/translations/:id
Human translator reviews and edits
is_auto_translated: false
```

Analytics Workflow

Step 1: Data Collection

```
Audit middleware logs all actions
MongoDB audit_logs collection
Redis caching for quick access
```

Step 2: Data Aggregation

```
Background job: analytics.job.js
Runs daily at midnight
Aggregates data from PostgreSQL
Stores in MongoDB analytics collection
```

Step 3: Dashboard Display

```
GET /api/v1/analytics/overview
GET /api/v1/dashboard/admin
Cached in Redis for 5 minutes
```

5. Implementation Guide

Phase 1: Setup (Week 1)

1.1 Initialize Project

```
mkdir medivoy-api
cd medivoy-api
npm init -y
```

1.2 Install Dependencies

```
# Core
npm install express cors helmet compression

# Database
npm install pg mongodb redis ioredis

# Authentication
npm install jsonwebtoken bcryptjs

# Validation
npm install joi express-validator

# File Upload
npm install multer aws-sdk

# Email/SMS
npm install @sendgrid/mail twilio

# Payment
npm install stripe razorpay

# Background Jobs
npm install bull

# Utilities
npm install dotenv winston moment uuid

# Dev Dependencies
npm install --save-dev nodemon jest supertest eslint prettier
```

1.3 Create Folder Structure

```
mkdir -p src/{config,constants,controllers,db,jobs,middleware,models,routes,services,util
mkdir -p src/db/{postgres,mongodb,redis,migrations,seeds}
mkdir -p src/routes/{v1,webhooks}
mkdir -p tests/{unit,integration,e2e}
mkdir -p docs uploads logs
```

1.4 Setup Environment Variables

```
# .env.example
NODE_ENV=development
PORT=5000
HOST=localhost

# PostgreSQL
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=medivoy
POSTGRES_USER=postgres
POSTGRES_PASSWORD=password

# MongoDB
MONGODB_URI=mongodb://localhost:27017/medivoy

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379

# JWT
JWT_SECRET=your-secret-key
JWT_REFRESH_SECRET=your-refresh-secret
JWT_EXPIRES_IN=15m
JWT_REFRESH_EXPIRES_IN=7d

# AWS S3
AWS_ACCESS_KEY_ID=your-key
AWS_SECRET_ACCESS_KEY=your-secret
AWS_REGION=us-east-1
AWS_S3_BUCKET=medivoy-uploads

# Twilio
TWILIO_ACCOUNT_SID=your-sid
TWILIO_AUTH_TOKEN=your-token
TWILIO_PHONE_NUMBER=+1234567890

# SendGrid
SENDGRID_API_KEY=your-key
SENDGRID_FROM_EMAIL=noreply@medivoy.com

# Stripe
STRIPE_SECRET_KEY=your-key
STRIPE_WEBHOOK_SECRET=your-webhook-secret

# Razorpay
RAZORPAY_KEY_ID=your-key
RAZORPAY_KEY_SECRET=your-secret

# Firebase
FIREBASE_PROJECT_ID=your-project
FIREBASE_PRIVATE_KEY=your-key
FIREBASE_CLIENT_EMAIL=your-email

# NextCloud
```

```
NEXTCLOUD_URL=https://your-nextcloud.com
NEXTCLOUD_USERNAME=admin
NEXTCLOUD_PASSWORD=password
```

Phase 2: Database Setup (Week 1-2)

2.1 Create Migration Runner

```
// src/db/migrations/index.js
const { Pool } = require('pg');
const fs = require('fs');
const path = require('path');

const pool = new Pool({
    host: process.env.POSTGRES_HOST,
    port: process.env.POSTGRES_PORT,
    database: process.env.POSTGRES_DB,
    user: process.env.POSTGRES_USER,
    password: process.env.POSTGRES_PASSWORD
});

async function runMigrations() {
    const migrationsDir = __dirname;
    const files = fs.readdirSync(migrationsDir)
        .filter(f =&gt; f.endsWith('.sql'))
        .sort();

    for (const file of files) {
        console.log(`Running migration: ${file}`);
        const sql = fs.readFileSync(
            path.join(migrationsDir, file),
            'utf8'
        );
        await pool.query(sql);
        console.log(`Completed: ${file}`);
    }
}

module.exports = { runMigrations };
```

2.2 Run Migrations

```
node src/db/migrations/index.js
```

Phase 3: Core Implementation (Week 2-4)

3.1 Express App Setup

```
// src/utils/app.js
const express = require('express');
const helmet = require('helmet');
const cors = require('cors');
const compression = require('compression');
const routes = require('../routes');
const errorMiddleware = require('../middleware/error.middleware');
const loggerMiddleware = require('../middleware/logger.middleware');

function createApp() {
    const app = express();

    // Security
    app.use(helmet());
    app.use(cors());

    // Parsing
    app.use(express.json());
    app.use(express.urlencoded({ extended: true }));

    // Compression
    app.use(compression());

    // Logging
    app.use(loggerMiddleware);

    // Routes
    app.use('/api', routes);

    // Error handling
    app.use(errorMiddleware);

    return app;
}

module.exports = createApp;
```

3.2 Server Startup

```
// src/utils/server.js
const createApp = require('./app');
const logger = require('./logger');

function startServer() {
    const app = createApp();
    const PORT = process.env.PORT || 5000;

    app.listen(PORT, () =&gt; {
        logger.info(`Server running on port ${PORT}`);
    });
}
```

```
module.exports = startServer;
```

3.3 Main Entry Point

```
// src/app.js
require('dotenv').config();
const startServer = require('./utils/server');
const { connectPostgres } = require('./db/postgres/connection');
const { connectMongoDB } = require('./db/mongodb/connection');
const { connectRedis } = require('./db/redis/connection');
const logger = require('./utils/logger');

async function bootstrap() {
    try {
        // Connect to databases
        await connectPostgres();
        await connectMongoDB();
        await connectRedis();

        // Start server
        startServer();

        logger.info('Application started successfully');
    } catch (error) {
        logger.error('Application startup failed:', error);
        process.exit(1);
    }
}

bootstrap();
```

Phase 4: Authentication Implementation (Week 3)

4.1 JWT Utilities

```
// src/utils/jwt.js
const jwt = require('jsonwebtoken');

function generateAccessToken(payload) {
    return jwt.sign(payload, process.env.JWT_SECRET, {
        expiresIn: process.env.JWT_EXPIRES_IN
    });
}

function generateRefreshToken(payload) {
    return jwt.sign(payload, process.env.JWT_REFRESH_SECRET, {
        expiresIn: process.env.JWT_REFRESH_EXPIRES_IN
    });
}

function verifyAccessToken(token) {
```

```
    return jwt.verify(token, process.env.JWT_SECRET);
}

function verifyRefreshToken(token) {
    return jwt.verify(token, process.env.JWT_REFRESH_SECRET);
}

module.exports = {
    generateAccessToken,
    generateRefreshToken,
    verifyAccessToken,
    verifyRefreshToken
};
```

4.2 Authentication Middleware

```
// src/middleware/auth.middleware.js
const { verifyAccessToken } = require('../utils/jwt');
const { AppError } = require('../utils/error-handler');

async function authenticate(req, res, next) {
    try {
        const token = req.headers.authorization?.split(' ')[1];

        if (!token) {
            throw new AppError('No token provided', 401);
        }

        const decoded = verifyAccessToken(token);
        req.user = decoded;
        next();
    } catch (error) {
        next(new AppError('Invalid token', 401));
    }
}

module.exports = authenticate;
```

4.3 Authorization Middleware

```
// src/middleware/authorize.middleware.js
const { AppError } = require('../utils/error-handler');

function authorize(...roles) {
    return (req, res, next) =&gt; {
        if (!roles.includes(req.user.user_type)) {
            return next(new AppError('Access denied', 403));
        }
        next();
    };
}
```

```
module.exports = authorize;
```

Phase 5: Controllers Implementation (Week 4-6)

5.1 Auth Controller Example

```
// src/controllers/auth.controller.js
const authService = require('../services/auth.service');
const { successResponse } = require('../utils/response-formatter');

async function register(req, res, next) {
    try {
        const result = await authService.register(req.body);
        res.status(201).json(successResponse(
            'User registered successfully',
            result
        ));
    } catch (error) {
        next(error);
    }
}

async function login(req, res, next) {
    try {
        const result = await authService.login(req.body);
        res.json(successResponse('Login successful', result));
    } catch (error) {
        next(error);
    }
}

async function logout(req, res, next) {
    try {
        await authService.logout(req.user.id);
        res.json(successResponse('Logout successful'));
    } catch (error) {
        next(error);
    }
}

module.exports = {
    register,
    login,
    logout
};
```

5.2 Booking Controller Example

```
// src/controllers/bookings.controller.js
const bookingService = require('../services/booking.service');
const { successResponse } = require('../utils/response-formatter');

async function createBooking(req, res, next) {
    try {
        const booking = await bookingService.create({
            ...req.body,
            patient_id: req.user.patient_id
        });

        res.status(201).json(successResponse(
            'Booking created successfully',
            booking
        ));
    } catch (error) {
        next(error);
    }
}

async function updateBookingStatus(req, res, next) {
    try {
        const { id } = req.params;
        const { status, notes } = req.body;

        const booking = await bookingService.updateStatus(
            id,
            status,
            notes
        );

        res.json(successResponse(
            'Status updated successfully',
            booking
        ));
    } catch (error) {
        next(error);
    }
}

module.exports = {
    createBooking,
    updateBookingStatus
};
```

Phase 6: Services Implementation (Week 6-8)

6.1 Auth Service Example

```
// src/services/auth.service.js
const bcrypt = require('bcryptjs');
const { AppError } = require('../utils/error-handler');
const { generateAccessToken, generateRefreshToken } = require('../utils/jwt');
const User = require('../models/User.model');

async function register(data) {
    // Check if user exists
    const existingUser = await User.findByEmail(data.email);
    if (existingUser) {
        throw new AppError('Email already registered', 400);
    }

    // Hash password
    const password_hash = await bcrypt.hash(data.password, 10);

    // Create user
    const user = await User.create({
        ...data,
        password_hash
    });

    // Generate tokens
    const access_token = generateAccessToken({
        id: user.id,
        user_type: user.user_type
    });
    const refresh_token = generateRefreshToken({
        id: user.id
    });

    // Save refresh token
    await RefreshToken.create({
        user_id: user.id,
        token: refresh_token
    });

    return {
        user: {
            id: user.id,
            email: user.email,
            user_type: user.user_type
        },
        access_token,
        refresh_token
    };
}

async function login(data) {
    // Find user
    const user = await User.findByEmail(data.email);
    if (!user) {
        throw new AppError('Invalid credentials', 401);
    }
```

```
    // Verify password
    const isValid = await bcrypt.compare(
        data.password,
        user.password_hash
    );
    if (!isValid) {
        throw new AppError('Invalid credentials', 401);
    }

    // Generate tokens
    const access_token = generateAccessToken({
        id: user.id,
        user_type: user.user_type
    });
    const refresh_token = generateRefreshToken({
        id: user.id
    });

    return {
        user,
        access_token,
        refresh_token
    };
}

module.exports = {
    register,
    login
};
```

6.2 Booking Service Example

```
// src/services/booking.service.js
const Booking = require('../models/Booking.model');
const notificationService = require('./notification.service');
const { generateBookingNumber } = require('../utils/helpers');

async function create(data) {
    // Generate booking number
    const booking_number = generateBookingNumber();

    // Create booking
    const booking = await Booking.create({
        ...data,
        booking_number,
        status: 'requested'
    });

    // Send notifications
    await notificationService.sendBookingCreated(booking);

    return booking;
}
```

```
async function updateStatus(id, status, notes) {
    // Update booking
    const booking = await Booking.updateStatus(id, status, notes);

    // Send notifications based on status
    await notificationService.sendBookingStatusUpdate(booking);

    // Trigger workflows
    if (status === 'payment_completed') {
        // Generate invoice
        await invoiceService.generate(booking);
    }

    return booking;
}

module.exports = {
    create,
    updateStatus
};
```

Phase 7: Background Jobs (Week 8)

7.1 Email Job

```
// src/jobs/email.job.js
const Queue = require('bull');
const emailService = require('../services/email.service');

const emailQueue = new Queue('email', {
    redis: {
        host: process.env.REDIS_HOST,
        port: process.env.REDIS_PORT
    }
});

emailQueue.process(async (job) =&gt; {
    const { to, subject, html } = job.data;
    await emailService.send(to, subject, html);
});

async function addEmailJob(data) {
    await emailQueue.add(data, {
        attempts: 3,
        backoff: {
            type: 'exponential',
            delay: 2000
        }
    });
}

module.exports = { addEmailJob };
```

7.2 Translation Job

```
// src/jobs/translation.job.js
const cron = require('node-cron');
const Translation = require('../models/Translation.model');
const translationService = require('../services/translation.service');

// Run every hour
cron.schedule('0 * * * *', async () =&gt; {
    console.log('Running translation job');

    // Find untranslated content
    const pending = await Translation.findPending();

    for (const item of pending) {
        await translationService.autoTranslate(item);
    }
});
```

Phase 8: Testing (Week 9)

8.1 Unit Test Example

```
// tests/unit/services/auth.service.test.js
const authService = require('../../../src/services/auth.service');
const User = require('../../../src/models/User.model');

describe('Auth Service', () =&gt; {
    describe('register', () =&gt; {
        it('should create a new user', async () =&gt; {
            const data = {
                email: 'test@example.com',
                password: 'Password123!',
                user_type: 'patient'
            };

            const result = await authService.register(data);

            expect(result).toHaveProperty('user');
            expect(result).toHaveProperty('access_token');
            expect(result.user.email).toBe(data.email);
        });

        it('should throw error for existing email', async () =&gt; {
            const data = {
                email: 'existing@example.com',
                password: 'Password123!'
            };

            await expect(authService.register(data))
                .rejects
                .toThrow('Email already registered');
        });
```

```
    });
});
```

8.2 Integration Test Example

```
// tests/integration/auth.test.js
const request = require('supertest');
const app = require('../../src/utils/app');

describe('Auth API', () =&gt; {
    describe('POST /api/v1/auth/register', () =&gt; {
        it('should register a new user', async () =&gt; {
            const res = await request(app())
                .post('/api/v1/auth/register')
                .send({
                    email: 'new@example.com',
                    password: 'Password123!',
                    user_type: 'patient',
                    first_name: 'John',
                    last_name: 'Doe'
                });

            expect(res.status).toBe(201);
            expect(res.body.success).toBe(true);
            expect(res.body.data).toHaveProperty('user');
            expect(res.body.data).toHaveProperty('access_token');
        });
    });

    describe('POST /api/v1/auth/login', () =&gt; {
        it('should login successfully', async () =&gt; {
            const res = await request(app())
                .post('/api/v1/auth/login')
                .send({
                    email: 'test@example.com',
                    password: 'Password123!'
                });

            expect(res.status).toBe(200);
            expect(res.body.data).toHaveProperty('access_token');
        });
    });
});
```

Phase 9: Documentation (Week 10)

9.1 Swagger Configuration

```
// src/utils/swagger.js
const swaggerJsdoc = require('swagger-jsdoc');

const options = {
    definition: {
        openapi: '3.0.0',
        info: {
            title: 'Medivoy Healthcare API',
            version: '1.0.0',
            description: 'Complete API documentation'
        },
        servers: [
            {
                url: 'http://localhost:5000',
                description: 'Development'
            },
            {
                url: 'https://api.medivoy.com',
                description: 'Production'
            }
        ],
        components: {
            securitySchemes: {
                BearerAuth: {
                    type: 'http',
                    scheme: 'bearer',
                    bearerFormat: 'JWT'
                }
            }
        }
    },
    apis: ['./src/routes/**/*.js']
};

const specs = swaggerJsdoc(options);

module.exports = specs;
```

Phase 10: Deployment (Week 11-12)

10.1 Docker Configuration

```
# Dockerfile
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm ci --only=production
```

```
COPY . .

EXPOSE 5000

CMD ["node", "src/app.js"]
```

10.2 Docker Compose

```
# docker-compose.yml
version: '3.8'

services:
  api:
    build: .
    ports:
      - "5000:5000"
    environment:
      - NODE_ENV=production
    depends_on:
      - postgres
      - mongodb
      - redis

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: medivoy
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data

  mongodb:
    image: mongo:6
    volumes:
      - mongo_data:/data/db

  redis:
    image: redis:7
    volumes:
      - redis_data:/data

volumes:
  postgres_data:
  mongo_data:
  redis_data:
```

10.3 CI/CD Pipeline

```
# .github/workflows/ci.yml
name: CI

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main, develop ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test

      - name: Run linter
        run: npm run lint
```

6. Security Implementation

6.1 Security Headers

```
// src/middleware/security.middleware.js
const helmet = require('helmet');

function securityHeaders() {
    return helmet({
        contentSecurityPolicy: {
            directives: {
                defaultSrc: ["'self'"],
                styleSrc: ["'self'", "'unsafe-inline'"],
                scriptSrc: ["'self'"],
                imgSrc: ["'self'", "data:", "https:"]
            }
        },
        hsts: {
            maxAge: 31536000,
            includeSubDomains: true,
            preload: true
```

```
        }
    });
}

module.exports = securityHeaders;
```

6.2 Rate Limiting

```
// src/middleware/rate-limit.middleware.js
const rateLimit = require('express-rate-limit');
const RedisStore = require('rate-limit-redis');
const redis = require('../db/redis/connection');

const limiter = rateLimit({
    store: new RedisStore({
        client: redis
    }),
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: 100, // limit each IP to 100 requests per windowMs
    message: 'Too many requests'
});

module.exports = limiter;
```

6.3 Input Validation

```
// src/validators/auth.validator.js
const Joi = require('joi');

const registerSchema = Joi.object({
    email: Joi.string().email().required(),
    password: Joi.string().min(8).required(),
    phone: Joi.string().pattern(/^\+[1-9]\d{1,14}$/),
    user_type: Joi.string().valid('admin', 'doctor', 'patient', 'hospital_admin').require
    first_name: Joi.string().required(),
    last_name: Joi.string().required()
});

function validateRegister(req, res, next) {
    const { error } = registerSchema.validate(req.body);
    if (error) {
        return res.status(400).json({
            success: false,
            message: error.details[0].message
        });
    }
    next();
}

module.exports = {
    validateRegister
};
```

7. Error Handling

7.1 Custom Error Classes

```
// src/utils/error-handler.js
class AppError extends Error {
    constructor(message, statusCode) {
        super(message);
        this.statusCode = statusCode;
        this.isOperational = true;
        Error.captureStackTrace(this, this.constructor);
    }
}

class ValidationError extends AppError {
    constructor(message) {
        super(message, 400);
    }
}

class UnauthorizedError extends AppError {
    constructor(message = 'Unauthorized') {
        super(message, 401);
    }
}

class ForbiddenError extends AppError {
    constructor(message = 'Forbidden') {
        super(message, 403);
    }
}

class NotFoundError extends AppError {
    constructor(message = 'Not found') {
        super(message, 404);
    }
}

module.exports = {
    AppError,
    ValidationError,
    UnauthorizedError,
    ForbiddenError,
    NotFoundError
};
```

7.2 Error Middleware

```
// src/middleware/error.middleware.js
const logger = require('../utils/logger');

function errorHandler(err, req, res, next) {
```

```
    logger.error(err);

    const statusCode = err.statusCode || 500;
    const message = err.message || 'Internal server error';

    res.status(statusCode).json({
        success: false,
        message,
        ...(process.env.NODE_ENV === 'development' &amp;amp;&amp;amp; {
            stack: err.stack
        })
    });
}

module.exports = errorHandler;
```

8. Performance Optimization

8.1 Response Caching

```
// src/middleware/cache.middleware.js
const redis = require('../db/redis/connection');

function cache(duration = 300) {
    return async (req, res, next) =&gt; {
        const key = `cache:${req.originalUrl}`;

        try {
            const cached = await redis.get(key);
            if (cached) {
                return res.json(JSON.parse(cached));
            }

            // Store original send function
            const originalSend = res.json;

            // Override send function
            res.json = function(data) {
                redis.setex(key, duration, JSON.stringify(data));
                originalSend.call(this, data);
            };

            next();
        } catch (error) {
            next();
        }
    };
}

module.exports = cache;
```

8.2 Database Query Optimization

```
// src/models/Booking.model.js
class Booking {
    static async findWithRelations(id) {
        const query = `
            SELECT
                b.*,
                p.first_name as patient_first_name,
                p.last_name as patient_last_name,
                h.name as hospital_name,
                t.name as treatment_name
            FROM bookings b
            LEFT JOIN patients pt ON b.patient_id = pt.id
            LEFT JOIN users p ON pt.user_id = p.id
            LEFT JOIN hospitals h ON b.hospital_id = h.id
            LEFT JOIN treatments t ON b.treatment_id = t.id
            WHERE b.id = $1
        `;

        const result = await pool.query(query, [id]);
        return result.rows[0];
    }
}
```

9. Monitoring & Logging

9.1 Logger Configuration

```
// src/utils/logger.js
const winston = require('winston');

const logger = winston.createLogger({
    level: process.env.LOG_LEVEL || 'info',
    format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.errors({ stack: true }),
        winston.format.json()
    ),
    transports: [
        new winston.transports.File({
            filename: 'logs/error.log',
            level: 'error'
        }),
        new winston.transports.File({
            filename: 'logs/app.log'
        })
    ]
});

if (process.env.NODE_ENV !== 'production') {
    logger.add(new winston.transports.Console({
        format: winston.format.simple()
```

```
    }));
}

module.exports = logger;
```

9.2 Request Logging

```
// src/middleware/logger.middleware.js
const logger = require('../utils/logger');

function loggerMiddleware(req, res, next) {
    const start = Date.now();

    res.on('finish', () =&gt; {
        const duration = Date.now() - start;
        logger.info({
            method: req.method,
            url: req.originalUrl,
            status: res.statusCode,
            duration: `${duration}ms`,
            ip: req.ip,
            user_agent: req.get('user-agent')
        });
    });

    next();
}

module.exports = loggerMiddleware;
```

10. Package.json Configuration

```
{
  "name": "medivoy-api",
  "version": "1.0.0",
  "description": "Medivoy Healthcare System Backend API",
  "main": "src/app.js",
  "scripts": {
    "start": "node src/app.js",
    "dev": "nodemon src/app.js",
    "test": "jest --coverage",
    "test:watch": "jest --watch",
    "lint": "eslint src/**/*.js",
    "format": "prettier --write src/**/*.js",
    "migrate": "node src/db/migrations/index.js",
    "seed": "node src/db/seeds/index.js",
    "docker:build": "docker build -t medivoy-api .",
    "docker:up": "docker-compose up -d",
    "docker:down": "docker-compose down"
  },
  "dependencies": {
    "express": "^4.18.2",
```

```
    "cors": "^2.8.5",
    "helmet": "^7.1.0",
    "compression": "^1.7.4",
    "pg": "^8.11.3",
    "mongodb": "^6.3.0",
    "mongoose": "^8.0.3",
    "redis": "^4.6.11",
    "ioredis": "^5.3.2",
    "jsonwebtoken": "^9.0.2",
    "bcryptjs": "^2.4.3",
    "joi": "^17.11.0",
    "express-validator": "^7.0.1",
    "multer": "^1.4.5-lts.1",
    "aws-sdk": "^2.1509.0",
    "@sendgrid/mail": "^8.1.0",
    "twilio": "^4.19.3",
    "stripe": "^14.8.0",
    "razorpay": "^2.9.2",
    "bull": "^4.12.0",
    "node-cron": "^3.0.3",
    "dotenv": "^16.3.1",
    "winston": "^3.11.0",
    "moment": "^2.29.4",
    "uuid": "^9.0.1",
    "pdfkit": "^0.13.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.2",
    "jest": "^29.7.0",
    "supertest": "^6.3.3",
    "eslint": "^8.55.0",
    "prettier": "^3.1.1"
  }
}
```

Conclusion

This comprehensive guide covers every aspect of the Medivoy Healthcare System backend API implementation. Follow the phases sequentially for systematic development. Each component is production-ready with security, performance, and scalability considerations built-in.

Total Development Timeline: 12 weeks
Total Files to Create: 150+ files
Total API Endpoints: 200+ endpoints
Database Tables: 29 PostgreSQL tables + 3 MongoDB collections

All workflows from your diagrams have been documented and implemented including booking flows, appointment scheduling, payment processing, translation workflows, and analytics pipelines.