IVF EMR Technical Documentation

Developer Guide and System Architecture

Technical Team

2024

Table of Contents

# System Architecture Overview

## Technology Stack

### Frontend Architecture

* **Framework**: React 18 with TypeScript
* **State Management**: React Hooks and Context API
* **UI Components**: Custom design system with Tailwind CSS
* **Build Tool**: Vite for fast development and optimized builds
* **Testing**: Jest and React Testing Library

### Backend Services

* **Runtime**: Node.js with Express.js framework
* **Database**: PostgreSQL with Prisma ORM
* **Authentication**: JWT tokens with refresh mechanism
* **File Storage**: AWS S3 for document and image storage
* **Caching**: Redis for session and data caching

### Infrastructure

* **Deployment**: Docker containers with Kubernetes orchestration
* **Cloud Provider**: AWS with multi-region deployment
* **Monitoring**: CloudWatch, Prometheus, and Grafana
* **CI/CD**: GitHub Actions with automated testing and deployment
* **Security**: WAF, VPC, and encrypted data transmission

## Data Model Architecture

### Core Entities

#### Patient Entity

interface Patient {  
 id: string; // Primary key: P-YYYY-NNNN  
 name: string; // Full legal name  
 age: number; // Calculated from DOB  
 dateOfBirth: string; // ISO 8601 format  
 contactInfo: ContactInfo; // Nested contact details  
 medicalHistory: MedicalHistory; // Medical background  
 insuranceInfo: InsuranceInfo; // Coverage information  
 currentCycle: TreatmentCycle; // Active treatment  
 treatmentHistory: TreatmentCycle[]; // Previous cycles  
 notes: string; // Clinical notes  
 createdAt: Date; // Record creation  
 updatedAt: Date; // Last modification  
}

**Field Validation Rules:** - id: Auto-generated, format P-YYYY-NNNN, unique constraint - name: Required, 2-100 characters, alphabetic with spaces - age: Calculated field, range 18-65, updated on DOB change - dateOfBirth: Required, valid date, not future, age calculation source - contactInfo.email: Valid email format, unique constraint - contactInfo.phone: US phone format (XXX) XXX-XXXX

**Business Logic:** - Age automatically calculated from dateOfBirth - Insurance eligibility checked on cycle initiation - Medical history influences protocol recommendations - Contact preferences determine notification methods

#### Treatment Cycle Entity

interface TreatmentCycle {  
 id: string; // Unique cycle identifier  
 patientId: string; // Foreign key to Patient  
 cycleNumber: number; // Sequential cycle count  
 startDate: string; // Cycle initiation date  
 endDate?: string; // Cycle completion date  
 protocol: ProtocolType; // Treatment protocol  
 status: CycleStatus; // Current cycle state  
 medications: Medication[]; // Prescribed drugs  
 monitoring: MonitoringEvent[]; // Scheduled appointments  
 procedures: Procedure[]; // Clinical procedures  
 labResults: LabResult[]; // Test results  
 outcomes: CycleOutcome; // Final results  
 notes: string; // Cycle-specific notes  
}

**Status Workflow:**

planned → active → monitoring → procedures → completed  
 ↓ ↓ ↓ ↓  
cancelled ← cancelled ← cancelled ← cancelled

**Derived Calculations:** - Cycle day: currentDate - startDate - Expected procedures: Based on protocol and cycle day - Medication adjustments: Based on monitoring results - Success probability: Algorithm using age, AMH, previous cycles

#### Laboratory Results Entity

interface LabResult {  
 id: string; // Unique result identifier  
 patientId: string; // Foreign key to Patient  
 cycleId: string; // Foreign key to TreatmentCycle  
 testType: LabTestType; // Type of test performed  
 value: number; // Numeric result value  
 unit: string; // Unit of measurement  
 referenceRange: string; // Normal range for test  
 cycleDay: number; // Day of cycle when drawn  
 collectionDate: Date; // Sample collection time  
 resultDate: Date; // Result availability time  
 interpretation: ResultInterpretation; // Clinical significance  
 technician: string; // Lab technician ID  
 verified: boolean; // Quality control flag  
 critical: boolean; // Critical value flag  
}

**Reference Ranges by Test Type:**

const REFERENCE\_RANGES = {  
 'E2': { unit: 'pg/mL', ranges: {  
 'baseline': { min: 20, max: 80 },  
 'stimulation': { min: 100, max: 4000 },  
 'trigger': { min: 1500, max: 6000 }  
 }},  
 'FSH': { unit: 'mIU/mL', ranges: {  
 'baseline': { min: 3, max: 20 },  
 'menopause': { min: 25, max: 134 }  
 }},  
 'LH': { unit: 'mIU/mL', ranges: {  
 'baseline': { min: 2, max: 15 },  
 'surge': { min: 20, max: 100 }  
 }},  
 'hCG': { unit: 'mIU/mL', ranges: {  
 'negative': { min: 0, max: 5 },  
 'positive': { min: 25, max: 50000 }  
 }}  
};

## Field Semantics and Correlations

### Data Relationships

#### Patient-Cycle Correlation

// One-to-Many relationship  
Patient.id → TreatmentCycle.patientId  
  
// Derived values from correlation:  
- Total cycles attempted  
- Time between cycles  
- Protocol progression  
- Cumulative success rate

#### Cycle-Monitoring Correlation

// Cycle day determines appropriate tests  
const getRequiredTests = (cycleDay: number, protocol: string) => {  
 const testSchedule = {  
 'Long Agonist': {  
 baseline: [1, 2, 3], // E2, FSH, LH  
 stimulation: [5, 7, 9], // E2, LH  
 trigger: [11, 12] // E2, LH, P4  
 }  
 };  
 return testSchedule[protocol][getCyclePhase(cycleDay)];  
};

#### Lab Results-Dosing Correlation

// Medication adjustments based on response  
const calculateDosage = (  
 baselineE2: number,  
 currentE2: number,  
 currentDose: number,  
 cycleDay: number  
): number => {  
 const expectedE2 = baselineE2 \* Math.pow(2, cycleDay - 3);  
 const responseRatio = currentE2 / expectedE2;  
   
 if (responseRatio < 0.5) {  
 return Math.min(currentDose \* 1.5, 450); // Increase dose  
 } else if (responseRatio > 2.0) {  
 return Math.max(currentDose \* 0.75, 75); // Decrease dose  
 }  
 return currentDose; // Maintain dose  
};

### Derived Value Calculations

#### BMI and Protocol Selection

const calculateBMI = (weight: number, height: number): number => {  
 return weight / Math.pow(height / 100, 2);  
};  
  
const selectProtocol = (age: number, bmi: number, amh: number): string => {  
 if (age > 40 || amh < 1.0) {  
 return 'Antagonist Protocol'; // Gentle stimulation  
 } else if (bmi > 30) {  
 return 'Long Agonist Protocol'; // Better control  
 } else {  
 return 'Short Agonist Protocol'; // Standard approach  
 }  
};

#### Success Probability Algorithm

const calculateSuccessProbability = (  
 age: number,  
 amh: number,  
 previousCycles: number,  
 diagnosis: string[]  
): number => {  
 let baseRate = 0.4; // 40% base success rate  
   
 // Age factor  
 if (age < 30) baseRate \*= 1.3;  
 else if (age < 35) baseRate \*= 1.1;  
 else if (age < 40) baseRate \*= 0.9;  
 else baseRate \*= 0.6;  
   
 // AMH factor  
 if (amh > 3.0) baseRate \*= 1.2;  
 else if (amh < 1.0) baseRate \*= 0.7;  
   
 // Previous cycle factor  
 baseRate \*= Math.pow(0.9, previousCycles);  
   
 // Diagnosis factor  
 const poorPrognosisConditions = [  
 'severe male factor',  
 'endometriosis',  
 'poor ovarian reserve'  
 ];  
   
 if (diagnosis.some(d => poorPrognosisConditions.includes(d))) {  
 baseRate \*= 0.8;  
 }  
   
 return Math.min(Math.max(baseRate, 0.05), 0.85); // 5-85% range  
};

## Resource Optimization Algorithms

### Scheduling Algorithm

class SchedulingOptimizer {  
 private staff: StaffResource[];  
 private equipment: EquipmentResource[];  
 private rooms: RoomResource[];  
   
 optimizeSchedule(  
 tasks: Task[],  
 constraints: SchedulingConstraints  
 ): OptimizationResult {  
 // 1. Sort tasks by priority and dependencies  
 const sortedTasks = this.prioritizeTasks(tasks);  
   
 // 2. For each task, find optimal time slot  
 const schedule: ScheduledTask[] = [];  
 const conflicts: Conflict[] = [];  
   
 for (const task of sortedTasks) {  
 const slot = this.findOptimalSlot(task, schedule, constraints);  
   
 if (slot.isValid) {  
 schedule.push({ ...task, ...slot });  
 } else {  
 conflicts.push({  
 task,  
 reason: slot.conflictReason,  
 alternatives: this.generateAlternatives(task, schedule)  
 });  
 }  
 }  
   
 return {  
 schedule,  
 conflicts,  
 utilizationScore: this.calculateUtilization(schedule),  
 recommendations: this.generateRecommendations(conflicts)  
 };  
 }  
   
 private findOptimalSlot(  
 task: Task,  
 existingSchedule: ScheduledTask[],  
 constraints: SchedulingConstraints  
 ): TimeSlot {  
 const requiredDuration = this.calculateDuration(task);  
 const availableSlots = this.getAvailableSlots(  
 task.preferredDate,  
 requiredDuration,  
 existingSchedule  
 );  
   
 // Score each slot based on multiple factors  
 const scoredSlots = availableSlots.map(slot => ({  
 ...slot,  
 score: this.scoreTimeSlot(slot, task, existingSchedule, constraints)  
 }));  
   
 // Return highest scoring slot  
 return scoredSlots.sort((a, b) => b.score - a.score)[0];  
 }  
   
 private scoreTimeSlot(  
 slot: TimeSlot,  
 task: Task,  
 schedule: ScheduledTask[],  
 constraints: SchedulingConstraints  
 ): number {  
 let score = 100; // Base score  
   
 // Prefer earlier times for urgent tasks  
 if (task.priority === 'high') {  
 score += (18 - slot.startHour) \* 2; // Earlier is better  
 }  
   
 // Penalize overtime hours  
 if (slot.startHour < 8 || slot.endHour > 18) {  
 score -= 20;  
 }  
   
 // Reward efficient resource utilization  
 const utilization = this.calculateSlotUtilization(slot, schedule);  
 score += utilization \* 10;  
   
 // Penalize staff fatigue  
 const staffFatigue = this.calculateStaffFatigue(slot, task, schedule);  
 score -= staffFatigue \* 5;  
   
 return score;  
 }  
}

### Staff Wellness Algorithm

class WellnessMonitor {  
 calculateWellnessScore(  
 staffId: string,  
 tasks: Task[],  
 timeframe: TimeRange  
 ): WellnessMetrics {  
 const staffTasks = tasks.filter(t => t.assignedTo === staffId);  
   
 // Calculate workload metrics  
 const totalHours = this.calculateTotalHours(staffTasks);  
 const fatigueScore = this.calculateFatigueScore(staffTasks);  
 const stressLevel = this.calculateStressLevel(staffTasks, timeframe);  
   
 // Determine wellness score (0-100)  
 let wellnessScore = 100;  
   
 // Deduct for excessive hours  
 const maxHours = this.getMaxHours(staffId);  
 if (totalHours > maxHours) {  
 wellnessScore -= (totalHours - maxHours) \* 10;  
 }  
   
 // Deduct for high fatigue  
 if (fatigueScore > 60) {  
 wellnessScore -= (fatigueScore - 60) \* 2;  
 }  
   
 // Deduct for high stress  
 wellnessScore -= stressLevel \* 5;  
   
 return {  
 staffId,  
 totalHours,  
 fatigueScore,  
 stressLevel,  
 wellnessScore: Math.max(0, wellnessScore),  
 recommendations: this.generateWellnessRecommendations(  
 totalHours,  
 fatigueScore,  
 stressLevel  
 )  
 };  
 }  
   
 private calculateFatigueScore(tasks: Task[]): number {  
 return tasks.reduce((total, task) => {  
 const procedure = PROCEDURES[task.type];  
 return total + (procedure?.fatigueScore || 0);  
 }, 0);  
 }  
   
 private generateWellnessRecommendations(  
 hours: number,  
 fatigue: number,  
 stress: number  
 ): string[] {  
 const recommendations: string[] = [];  
   
 if (hours > 8) {  
 recommendations.push('Consider redistributing workload');  
 }  
   
 if (fatigue > 50) {  
 recommendations.push('Schedule mandatory break period');  
 }  
   
 if (stress > 7) {  
 recommendations.push('Review task complexity and support needs');  
 }  
   
 return recommendations;  
 }  
}

## Configuration Management

### Configuration Schema

interface SystemConfiguration {  
 clinic: ClinicSettings;  
 scheduling: SchedulingSettings;  
 resources: ResourceSettings;  
 wellness: WellnessSettings;  
 notifications: NotificationSettings;  
 security: SecuritySettings;  
 integrations: IntegrationSettings;  
}  
  
interface ClinicSettings {  
 name: string;  
 timezone: string;  
 workingHours: {  
 start: string; // HH:MM format  
 end: string;  
 };  
 workingDays: DayOfWeek[];  
 holidays: Date[];  
 emergencyContacts: Contact[];  
}  
  
interface SchedulingSettings {  
 defaultSlotDuration: number; // minutes  
 bufferTime: number; // minutes between appointments  
 maxAdvanceBooking: number; // days  
 allowOverbooking: boolean;  
 overbookingPercentage: number; // 0-100  
 autoConfirmation: boolean;  
 reminderSettings: {  
 enabled: boolean;  
 daysBefore: number[];  
 methods: NotificationMethod[];  
 };  
}

### Configuration Validation

class ConfigurationValidator {  
 validate(config: SystemConfiguration): ValidationResult {  
 const errors: string[] = [];  
   
 // Validate working hours  
 if (config.clinic.workingHours.start >= config.clinic.workingHours.end) {  
 errors.push('Working hours: Start time must be before end time');  
 }  
   
 // Validate scheduling settings  
 if (config.scheduling.defaultSlotDuration < 15 ||   
 config.scheduling.defaultSlotDuration > 240) {  
 errors.push('Default slot duration must be between 15 and 240 minutes');  
 }  
   
 // Validate wellness thresholds  
 const wellness = config.wellness.thresholds;  
 if (wellness.excellent <= wellness.good ||   
 wellness.good <= wellness.warning ||   
 wellness.warning <= wellness.critical) {  
 errors.push('Wellness thresholds must be in descending order');  
 }  
   
 return {  
 isValid: errors.length === 0,  
 errors,  
 warnings: this.generateWarnings(config)  
 };  
 }  
}

## API Documentation

### Authentication Endpoints

// POST /api/auth/login  
interface LoginRequest {  
 email: string;  
 password: string;  
 mfaCode?: string;  
}  
  
interface LoginResponse {  
 accessToken: string;  
 refreshToken: string;  
 user: UserProfile;  
 permissions: Permission[];  
}  
  
// POST /api/auth/refresh  
interface RefreshRequest {  
 refreshToken: string;  
}  
  
interface RefreshResponse {  
 accessToken: string;  
 expiresIn: number;  
}

### Patient Management Endpoints

// GET /api/patients  
interface GetPatientsRequest {  
 page?: number;  
 limit?: number;  
 search?: string;  
 status?: PatientStatus;  
 sortBy?: 'name' | 'age' | 'lastVisit';  
 sortOrder?: 'asc' | 'desc';  
}  
  
// POST /api/patients  
interface CreatePatientRequest {  
 name: string;  
 dateOfBirth: string;  
 contactInfo: ContactInfo;  
 medicalHistory: MedicalHistory;  
 insuranceInfo: InsuranceInfo;  
}  
  
// PUT /api/patients/:id  
interface UpdatePatientRequest {  
 name?: string;  
 contactInfo?: Partial<ContactInfo>;  
 medicalHistory?: Partial<MedicalHistory>;  
 insuranceInfo?: Partial<InsuranceInfo>;  
}

### Scheduling Endpoints

// GET /api/schedule  
interface GetScheduleRequest {  
 date: string; // YYYY-MM-DD  
 staffId?: string;  
 roomId?: string;  
 view?: 'day' | 'week' | 'month';  
}  
  
// POST /api/schedule/optimize  
interface OptimizeScheduleRequest {  
 date: string;  
 constraints: SchedulingConstraints;  
 preferences: SchedulingPreferences;  
}  
  
interface OptimizeScheduleResponse {  
 optimizedSchedule: ScheduledTask[];  
 conflicts: Conflict[];  
 improvements: Improvement[];  
 utilizationScore: number;  
}

## Testing Strategy

### Unit Testing

// Example test for success probability calculation  
describe('calculateSuccessProbability', () => {  
 it('should return higher probability for younger patients', () => {  
 const youngPatient = calculateSuccessProbability(28, 3.5, 0, []);  
 const olderPatient = calculateSuccessProbability(42, 3.5, 0, []);  
   
 expect(youngPatient).toBeGreaterThan(olderPatient);  
 });  
   
 it('should decrease probability with more previous cycles', () => {  
 const firstCycle = calculateSuccessProbability(32, 2.5, 0, []);  
 const fourthCycle = calculateSuccessProbability(32, 2.5, 3, []);  
   
 expect(firstCycle).toBeGreaterThan(fourthCycle);  
 });  
   
 it('should handle edge cases appropriately', () => {  
 const result = calculateSuccessProbability(45, 0.5, 5, ['poor ovarian reserve']);  
   
 expect(result).toBeGreaterThanOrEqual(0.05);  
 expect(result).toBeLessThanOrEqual(0.85);  
 });  
});

### Integration Testing

// Example integration test for patient workflow  
describe('Patient Workflow Integration', () => {  
 it('should complete full patient registration and cycle initiation', async () => {  
 // 1. Create patient  
 const patient = await createPatient(mockPatientData);  
 expect(patient.id).toBeDefined();  
   
 // 2. Add medical history  
 await updateMedicalHistory(patient.id, mockMedicalHistory);  
   
 // 3. Initiate treatment cycle  
 const cycle = await initiateCycle(patient.id, mockCycleData);  
 expect(cycle.status).toBe('active');  
   
 // 4. Verify scheduling  
 const schedule = await getPatientSchedule(patient.id);  
 expect(schedule.length).toBeGreaterThan(0);  
   
 // 5. Verify notifications  
 const notifications = await getPatientNotifications(patient.id);  
 expect(notifications).toContain('cycle\_started');  
 });  
});

## Deployment Guide

### Docker Configuration

# Dockerfile for production deployment  
FROM node:18-alpine AS builder  
  
WORKDIR /app  
COPY package\*.json ./  
RUN npm ci --only=production  
  
COPY . .  
RUN npm run build  
  
FROM node:18-alpine AS runtime  
  
WORKDIR /app  
COPY --from=builder /app/dist ./dist  
COPY --from=builder /app/node\_modules ./node\_modules  
COPY --from=builder /app/package.json ./  
  
EXPOSE 3000  
CMD ["npm", "start"]

### Kubernetes Deployment

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: ivf-emr-app  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: ivf-emr  
 template:  
 metadata:  
 labels:  
 app: ivf-emr  
 spec:  
 containers:  
 - name: ivf-emr  
 image: ivf-emr:latest  
 ports:  
 - containerPort: 3000  
 env:  
 - name: DATABASE\_URL  
 valueFrom:  
 secretKeyRef:  
 name: db-secret  
 key: url  
 - name: JWT\_SECRET  
 valueFrom:  
 secretKeyRef:  
 name: auth-secret  
 key: jwt-secret  
 resources:  
 requests:  
 memory: "256Mi"  
 cpu: "250m"  
 limits:  
 memory: "512Mi"  
 cpu: "500m"

### Environment Configuration

# Production environment variables  
NODE\_ENV=production  
PORT=3000  
DATABASE\_URL=postgresql://user:pass@host:5432/ivf\_emr  
REDIS\_URL=redis://redis-host:6379  
JWT\_SECRET=your-jwt-secret  
JWT\_EXPIRES\_IN=15m  
REFRESH\_TOKEN\_EXPIRES\_IN=7d  
AWS\_REGION=us-east-1  
AWS\_S3\_BUCKET=ivf-emr-files  
SMTP\_HOST=smtp.example.com  
SMTP\_PORT=587  
SMTP\_USER=noreply@clinic.com  
SMTP\_PASS=smtp-password

This technical documentation provides comprehensive coverage of the system architecture, data models, algorithms, and deployment procedures for the IVF EMR system.