Timeline Service Implementation Documentation

Chronic Diabetes Management System

---

1. Overview

The Timeline Service manages patient enrollment and appointment timelines for chronic disease management. It serves as the central component for tracking patient care continuity from initial enrollment through ongoing protocol-driven appointments.

Core Responsibilities

- Patient enrollment and registration validation
- Timeline generation (precedenti/oggi/successivo)
- Appointment status management
- Integration with Wirgilio EMR system
- Protocol-based care scheduling

---

2. System Architecture

Input Flow

```
Doctor Input → Timeline Service → Patient Lookup → Registration Decision → Timeline Display
```

Data Sources

- **Primary Database**: MongoDB (timeline-specific data)
- **External Integration**: Wirgilio EMR (patient demographics)
- **Hardcoded Elements**: Doctor credentials, protocol definitions

Service Communication

- **Frontend**: React form inputs and timeline display
- **Backend**: FastAPI with async MongoDB operations
- **Database**: MongoDB collections for patients and appointments

---

3. User Input Requirements

Required Inputs

- **Doctor ID**: Text input (e.g., "DOC001")
- **Patient CF**: Italian fiscal code input (e.g., "RSSMRA85M01H501Z")
- **Patologia**: Dropdown selection from predefined list

Predefined Pathologies

```python
PATHOLOGIES = [
  "diabetes_mellitus_type1",
  "diabetes_mellitus_type2",
  "diabetes_gestational",
  "hypertension_primary",
  "hypertension_secondary",
  "cardiovascular_disease",
  "chronic_kidney_disease"
]
```

Hardcoded Elements (Temporary)

- Doctor credentials and digital signatures
- Wirgilio authentication tokens
- Protocol definitions for each pathology

---

4. Database Schema

Collection: patients

```javascript
```

```
{
  _id: ObjectId("..."),
  cf_paziente: "RSSMRA85M01H501Z",
  id_medico: "DOC001",
  patologia: "diabetes_mellitus_type2",

  // Demographics (fetched from Wirgilio)
  demographics: {
    nome: "Mario",
    cognome: "Rossi",
    data_nascita: ISODate("1985-03-01"),
    telefono: "+39987654321",
    email: "mario.rossi@email.com"
  },

  // Enrollment metadata
  enrollment_date: ISODate("2024-01-15T08:30:00Z"),
  status: "active",
  created_at: ISODate("2024-01-15T08:30:00Z"),
  updated_at: ISODate("2024-01-15T08:30:00Z")
}
```

Collection: appointments

javascript

```
{
  _id: ObjectId("..."),
  cf_paziente: "RSSMRA85M01H501Z",
  appointment_type: "visita_diabetologica",
  scheduled_date: ISODate("2024-03-15T09:00:00Z"),
  status: "scheduled", // scheduled, completed, cancelled

  // Timeline categorization
  created_at: ISODate("2024-01-15T10:00:00Z"),
  completed_at: null,

  // Protocol tracking
  protocol_generated: true,
  notes: "Initial diabetes assessment"
}
```

Database Indexes

```javascript
// Patient lookups
db.patients.createIndex({"cf_paziente": 1}, {unique: true});
db.patients.createIndex({"id_medico": 1});

// Timeline queries
db.appointments.createIndex({"cf_paziente": 1, "scheduled_date": 1});
db.appointments.createIndex({"scheduled_date": 1});
```

---

5. Service Flow Logic

Step 1: Patient Lookup

1. Receive doctor_id, cf_paziente, patologia from frontend
2. Query patients collection for existing cf_paziente
3. Return patient status (found/not_found)

Step 2A: New Patient Flow

1. Display registration prompt to doctor
2. If confirmed, fetch demographics from Wirgilio
3. Create patient record with protocol-based appointments
4. Generate initial timeline

Step 2B: Existing Patient Flow

1. Query appointments collection for patient
2. Categorize appointments by date relative to today
3. Return structured timeline data

Step 3: Timeline Categorization

```python
```

```
today = datetime.now().date()

precedenti = appointments where scheduled_date < today
oggi = appointments where scheduled_date == today
successivo = first appointment where scheduled_date > today
```

---

6. API Endpoints

POST /patient/lookup

Purpose: Check if patient exists in timeline system **Input**:

```
json

{
  "cf_paziente": "RSSMRA85M01H501Z",
  "id_medico": "DOC001",
  "patologia": "diabetes_mellitus_type2"
}
```

Output:

```
json

{
  "exists": false,
  "message": "Patient not registered in timeline system"
}
```

POST /patient/register

Purpose: Register new patient and create initial timeline **Input**: Same as lookup + confirmation
Output: Patient record + initial timeline

GET /timeline/{cf_paziente}

Purpose: Retrieve patient timeline **Output**:

```
json
```

```
{
  "patient_id": "RSSMRA85M01H501Z",
  "precedenti": [...],
  "oggi": [...],
  "successivo": [...]
}
```

---

7. Frontend Components

PatientLookupForm

- Input fields for doctor_id, cf_paziente, patologia
- Submit button triggering patient lookup
- Loading state management

RegistrationDialog

- Display patient not found message
- Confirmation buttons (Yes/No)
- Patient demographics preview

TimelineDisplay

- Three-column layout (precedenti/oggi/successivo)
- Appointment cards with status indicators
- Navigation controls for historical data

---

8. Implementation Plan

Phase 1: Database Layer

1. Update MongoDB connection configuration
2. Create Patient and Appointment data models
3. Implement CRUD operations with Motor driver
4. Setup database indexes

Phase 2: Backend Services

1. Replace mock endpoints with real database operations
2. Implement patient lookup logic
3. Add registration flow with Wirgilio integration stub
4. Create timeline generation algorithm

Phase 3: Frontend Updates

1. Replace placeholder React components
2. Create form inputs and validation
3. Implement registration dialog flow
4. Build timeline display components

Phase 4: Integration

1. Connect frontend to backend APIs
2. Test complete user flow
3. Handle error cases and edge conditions
4. Performance optimization

---

9. Technical Considerations

Error Handling

- Invalid CF format validation
- Database connection failures
- Missing patient data scenarios
- Appointment scheduling conflicts

Performance Requirements

- Patient lookup response < 200ms
- Timeline generation < 300ms
- Efficient database queries with proper indexing
- Optimized frontend rendering

Security Aspects

- Input validation and sanitization
- Database query injection prevention
- Proper error message handling
- Audit trail for patient operations

---

10. Future Integration Points

Wirgilio EMR Integration

- Replace hardcoded credentials with real authentication
- Implement real-time patient data synchronization
- Add appointment completion notifications
- Handle protocol updates from EMR system

Scheduler Service Communication

- Send appointment scheduling requests
- Receive booking confirmations
- Handle appointment modifications
- Manage slot availability

Analytics Service Data

- Provide appointment history for compliance tracking
- Send protocol adherence metrics
- Support clinical outcome analysis
- Timeline-based reporting features

---

11. Development Notes

Current Implementation Status

- Basic FastAPI structure exists
- Mock data currently in use
- Database models not implemented

- Frontend uses placeholder components

Next Steps

1. Implement database models and connections
2. Create patient lookup and registration endpoints
3. Build frontend form components
4. Test complete patient enrollment flow
5. Add timeline display functionality

Testing Strategy

- Unit tests for database operations
- Integration tests for API endpoints
- Frontend component testing
- End-to-end user flow validation

This documentation serves as the blueprint for implementing a professional medical timeline system that ensures continuity of care for chronic disease patients while maintaining proper audit trails and integration capabilities.