MAV - Medical Access & Vision

Intelligent Clinical Decision Support System

Technical Documentation

Version 1.0.0

Project Type:	Clinical Decision Support System
Platform:	Web Application
Backend:	Python Flask
Frontend:	React 18 (Vite)
Database:	MongoDB
AI Engine:	Google Gemini

January 2026

Contents

Abstract

MAV is an advanced, production-ready software solution designed to bridge the gap between raw clinical data and modern Artificial Intelligence. This comprehensive system serves as an intelligent routing and processing engine that ingests multi-modal data (text, vital signs, and medical images), validates the information, enriches it with calculated metadata, and leverages state-of-the-art AI to assist healthcare professionals in early disease diagnosis.

The platform addresses critical challenges in modern healthcare environments: data fragmentation through unified data source management, latency through real-time decision support, privacy through strict HIPAA/GDPR compliance mechanisms, and reliability through graceful failure handling with fallback mechanisms.

This documentation provides a complete technical overview of the system architecture, implementation details, deployment procedures, and future improvement opportunities for the MAV platform.

Chapter 1

Introduction

1.1 Overview and Vision

The healthcare industry faces unprecedented challenges in managing the ever-increasing volume of clinical data while maintaining high standards of patient care. MAV emerges as a sophisticated solution to address these multifaceted challenges, providing healthcare professionals with an intelligent platform that not only manages patient records but also offers AI-powered diagnostic assistance.

1.1.1 Problem Statement

Modern healthcare systems struggle with several critical issues:

1. **Data Fragmentation**: Patient information is often scattered across multiple systems, making it difficult to obtain a comprehensive view of a patient's medical history.

2. **Decision Latency**: Healthcare professionals need rapid access to relevant information and analytical insights, especially in emergency triage situations.

3. **Privacy Concerns**: The integration of cloud-based AI services requires stringent data protection mechanisms to comply with regulations such as HIPAA and GDPR.

4. **System Reliability**: Medical systems must handle failures gracefully without compromising patient care continuity.

1.1.2 Solution Approach

MAV addresses these challenges through a carefully designed architecture that implements:

- A unified gateway pattern for clinical data processing
- Real-time AI-powered diagnostic assistance using Google Gemini
- Robust authentication and authorization mechanisms

- Comprehensive audit logging for accountability
- Flexible deployment options via Docker containerization

1.2 Key Features

The platform offers a comprehensive feature set designed to enhance clinical workflows:

- **Patient Management**: Complete patient lifecycle management with support for both Italian citizens (using Fiscal Code validation) and foreign patients (automatic ID generation).
- **Clinical Record Management**: Comprehensive clinical record creation, editing, and retrieval with support for vital signs, symptoms, notes, and file attachments.
- **AI-Powered Diagnostics**: Integration with Google Gemini for multimodal analysis of clinical data and medical images.
- **Medical Chatbot**: An AI-powered conversational assistant for medical queries.
- **Professional Reporting**: PDF export capabilities for clinical documentation.
- **Doctor Authentication**: Secure registration and login system with unique mnemonic doctor IDs.

Chapter 2

Technical Architecture

2.1 System Overview

MAV follows a **Layered Architecture** with a central Gateway component orchestrating the entire data flow pipeline. This architectural approach ensures separation of concerns, maintainability, and scalability.

2.1.1 The Gateway Concept

The core of the backend is the `ClinicalGateway`, which differs fundamentally from a standard CRUD controller. The Gateway treats every incoming clinical request as a payload that must pass through a strict pipeline of handlers, ensuring that no data is ever processed without proper validation, anonymization, and auditing.

2.1.2 Architecture Diagram

2.2 Data Flow Pipeline

The clinical data processing follows a well-defined pipeline:

Step 1: **Ingestion**: The React frontend sends a Patient Record containing symptoms, vital signs, and attachments via HTTP POST request.

Step 2: **Gateway Entry**: The request enters the `ClinicalGateway` through the Flask API layer, where initial request parsing occurs.

Step 3: **Processing Chain**: The request passes through a chain of specialized handlers:

- `ValidationHandler`: Verifies data integrity, including valid Fiscal Codes and realistic vital sign ranges.
- `EnrichmentHandler`: Adds calculated metadata such as patient age from date of birth and BMI calculations.
- `PrivacyHandler`: Anonymizes sensitive fields before external AI processing.

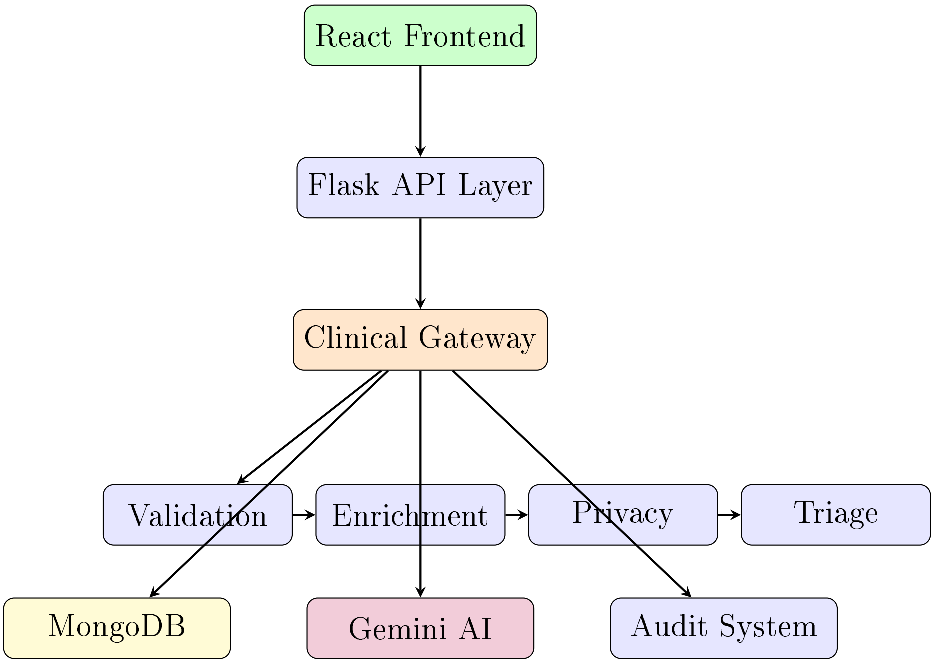

Figure 2.1: MAV System Architecture

- `TriageHandler`: Calculates initial urgency scores based on configured clinical rules.

Step 4: **Strategy Execution**: The system selects the appropriate AI strategy (e.g., `GeminiStrategy`) to analyze the clinical data.

Step 5: **Observer Notification**: Auditing and Metrics systems record the transaction results for compliance and analytics.

Step 6: **Persistence**: Validated and enriched data is stored in MongoDB collections.

Step 7: **Response**: The complete, analyzed result is returned to the Frontend for display.

2.3 Component Interaction

2.3.1 Frontend-Backend Communication

The frontend communicates with the backend through a RESTful API interface. All requests include credentials for session management:

```
const response = await fetch(getApiUrl('/api/patient/search'), {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    credentials: 'include',
    body: JSON.stringify({ fiscal_code: searchCode })
});
```

Listing 2.1: Frontend API Call Example

2.3.2 CORS Configuration

Cross-Origin Resource Sharing is configured to allow the frontend application to communicate with the backend API:

```
CORS(app,
     resources={
         r"/api/*": {
             "origins": allowed_origins,
             "methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],
             "allow_headers": ["Content-Type", "Authorization"],
             "supports_credentials": True
         }
     })
```

Listing 2.2: CORS Configuration

Chapter 3

Design Patterns Analysis

The system implements several GoF (Gang of Four) design patterns to ensure maintainability and extensibility.

3.1 Implemented Patterns

3.1.1 Strategy Pattern (Partially Implemented)

The AI module is designed to support multiple strategies, currently using Google Gemini as the sole provider. The architecture easily allows adding other AI providers (OpenAI, Claude, etc.) through common interfaces.

3.1.2 Decorator Pattern (Implemented)

Used for authentication via the `@require_login` decorator that protects sensitive routes by verifying user session presence.

```
def require_login(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'doctor_id' not in session:
            return jsonify({'error': 'Unauthorized'}), 401
        return f(*args, **kwargs)
    return decorated_function
```

Listing 3.1: Authentication Decorator

3.2 Conceptual Patterns

3.2.1 Chain of Responsibility

Described in the architecture for data processing pipeline (validation, enrichment, privacy, triage). Currently implemented inline in routes, but ready for formal refactoring.

3.2.2 Observer Pattern

Planned for audit logging and metrics. Currently the system uses direct logging, but the architecture supports adding observers for system events.

3.2.3 Facade Pattern

Planned for future integrations with external systems (HL7/FHIR, PACS, LIS).

3.3 Summary

Pattern	Status	Notes
Strategy	Partial	AI module ready
Decorator	Implemented	Route authentication
Chain of Responsibility	Conceptual	Inline implementation
Observer	Conceptual	Direct logging
Facade	Planned	Future integrations

Table 3.1: Pattern Implementation Status

Chapter 4

Technology Stack

MAVutilizes a modern, well-integrated technology stack designed for reliability, scalability, and maintainability.

4.1 Backend Technologies

4.1.1 Python Flask Framework

The backend is built using Flask, a lightweight micro-framework that provides flexibility without imposing unnecessary constraints.

Component	Version/Details
Python	3.8+
Flask	3.0.0+
Flask-CORS	4.0.0+

Table 4.1: Core Backend Framework

4.1.2 AI Integration

- **Google Generative AI SDK**: `google-generativeai >= 0.3.0`
- Model: `gemini-3-flash-preview`
- Supports multimodal input (text + images)

4.1.3 Data Processing Libraries

- **NumPy**: Numerical computing
- **Pandas**: Data manipulation and analysis
- **PIL (Pillow)**: Image processing for medical attachments
- **PyPDF2**: PDF parsing and generation

- **ReportLab**: Professional PDF report generation

4.1.4 Database Connectivity

- **PyMongo**: MongoDB driver for Python (>= 4.6.0)
- SSL/TLS support for secure connections
- Connection pooling for performance

4.2 Frontend Technologies

4.2.1 React 18 with Vite

The frontend leverages modern React development practices:

Technology	Version/Details
React	18.2.0+
React DOM	18.2.0+
Vite	5.0.8+
@vitejs/plugin-react	4.2.1+

Table 4.2: Frontend Framework Stack

4.2.2 State Management

- React Hooks (useState, useEffect)
- Context API for global state
- SessionStorage for data persistence

4.2.3 Styling

- Modern vanilla CSS with Glassmorphism design language
- Responsive layout optimized for dark/light mode
- CSS custom properties for theming

4.3 Infrastructure

4.3.1 Database

- **MongoDB**: NoSQL database for flexible schema management
- MongoDB Atlas for cloud deployment
- Polymorphic clinical data storage

4.3.2 Containerization

- **Docker**: Container runtime
- **Docker Compose**: Multi-container orchestration
- Nginx for frontend static file serving

4.3.3 Dependencies Summary

```
# Core
flask>=3.0.0
flask-cors>=4.0.0
pymongo>=4.6.0
google-generativeai>=0.3.0

# Data Processing
numpy>=1.21.0
pandas>=1.3.0

# PDF Generation
reportlab>=4.0.0
PyPDF2>=3.0.0

# Security
cryptography>=41.0.0

# Italian Fiscal Code
codicefiscale>=0.9
```

Listing 4.1: Key Backend Dependencies

Chapter 5

Core Components

5.1 Patient Management

The module manages the entire patient lifecycle with support for:

- **Italian Citizens**: Identification via validated Fiscal Code
- **Foreign Citizens**: Automatic unique ID generation (format: XX-YYYY)
- Automatic age and metadata calculation
- Allergy and permanent disease management

5.2 Doctor Authentication

Authentication system for doctors with:

- Automatically generated unique mnemonic ID (e.g., MR7X9Z)
- SHA-256 password hashing
- Session management with secure cookies
- Specialization and affiliated hospital support

5.3 Clinical Records

Clinical records include:

- Visit information (ID, timestamp, priority)
- Chief complaint and symptoms
- Vital signs (blood pressure, heart rate, temperature, saturation)
- Attachments (medical images, documents)
- Clinical notes

5.3.1 Vital Signs Reference

Parameter	**Unit**	**Normal Range**
Blood Pressure	mmHg	90/60 - 120/80
Heart Rate	bpm	60 - 100
Temperature	°C	36.1 - 37.2
O2 Saturation	%	95 - 100

Table 5.1: Vital Signs Ranges

5.4 PDF Export

Professional report generation with ReportLab including patient data, clinical records, attached images and AI diagnoses.

Chapter 6

AI and Diagnostics Engine

6.1 Overview

The AI module provides structured diagnostic support to healthcare professionals through integration with Google Gemini.

6.2 Multimodal Capabilities

The system analyzes simultaneously:

- **Structured Clinical Data**: Patient demographics, medical history, current symptoms, vital signs
- **Medical Images**: X-rays, ECGs, dermatological photographs (Base64 format, automatic RGB conversion)

6.3 Structured Output

The AI generates complete clinical assessments including:

1. Clinical data analysis and image interpretation
2. Presumptive diagnosis with probability and clinical reasoning
3. Differential diagnoses with supporting evidence
4. Recommended diagnostic tests
5. Treatment plan (pharmacological and non-pharmacological)
6. Monitoring and follow-up plan
7. Urgency assessment and intervention timeline

6.4 Error Handling

Implementation of retry logic with exponential backoff (max 3 attempts) to ensure service reliability.

6.5 Medical Chatbot

Dedicated AI instance for medical queries with:

- Isolated API key
- Conversational interface
- Strict medical-only policy
- Real-time response generation

Chapter 7

Security and Privacy

7.1 Authentication System

7.1.1 Doctor Registration

- Unique mnemonic ID generation
- Password validation (minimum 6 characters)
- SHA-256 hashing
- Duplicate ID prevention

7.1.2 Session Management

Secure cookies with HTTPS-only configuration in production, HttpOnly, SameSite policy and 7-day duration.

7.2 Data Protection

7.2.1 Data at Rest

- PII separation from clinical data in MongoDB
- Encryption options via MongoDB Atlas
- Secure database credentials management

7.2.2 Data in Transit

- Forced HTTPS in production
- TLS/SSL for database connections
- Secure cookie transmission

7.2.3 Anonymization

Anonymization functionality for external processing: removal of identifying data while maintaining relevant clinical information (allergies, diseases).

7.3 Audit Logging

Logging of all significant actions:

- Patient record creation/modification
- Clinical record access
- Export operations
- Authentication events
- AI diagnostic requests

7.4 Compliance

The system is designed considering:

- **HIPAA**: Health Insurance Portability and Accountability Act
- **GDPR**: General Data Protection Regulation
- Audit trail maintenance
- Data minimization

Chapter 8

Database Architecture

8.1 MongoDB Collections

The database uses 5 main collections:

- **doctors**: Doctor credentials and profiles (unique doctor_id)
- **patients**: Patient demographics (unique codice_fiscale/patient_id)
- **patient_records**: Clinical records (unique encounter_id, linked to patient_id)
- **audit_logs**: Immutable system action history
- **chatbot_sessions**: Medical chatbot sessions

8.2 Indexing Strategy

Collection	**Field**	**Type**
patients	codice_fiscale	Unique
patients	patient_id	Unique
patient_records	encounter_id	Unique
patient_records	patient_id	Regular
doctors	doctor_id	Unique
audit_logs	timestamp	Regular

Table 8.1: Database Indexes

8.3 Connection Configuration

MongoDB connection with TLS/SSL, configurable timeouts and connection pooling for optimal performance.

Chapter 9

Deployment Guide

9.1 Docker Deployment

9.1.1 Quick Start

```
git clone repository
cp .env.example .env
# Configure environment variables
docker-compose up -d --build
```

9.2 Manual Deployment

9.2.1 Backend

```
cd backend
python -m venv venv
venv\Scripts\activate  # Windows
pip install -r requirements.txt
python webapp/app.py
```

9.2.2 Frontend

```
cd frontend
npm install
npm run dev
```

9.3 Environment Variables

9.4 Troubleshooting

- **CORS**: Check protocol and trailing slashes

Variable	Description
GEMINI_API_KEY	Google Gemini API key
MONGODB_CONNECTION_STRING	MongoDB Atlas connection string
FLASK_SECRET_KEY	Session encryption key
FRONTEND_URL	Frontend URL (CORS)
VITE_API_URL	Backend API URL

Table 9.1: Required Environment Variables

- **AI**: Images under 4MB, standard formats
- **Database**: IP whitelist MongoDB Atlas
- **Sessions**: Cookie configuration cross-domain

Chapter 10

Future Improvements

10.1 Enhanced Security

- Advanced password hashing (bcrypt/Argon2)
- Multi-Factor Authentication (TOTP, SMS, FIDO2)
- JWT-based authentication with token rotation
- Granular RBAC (Admin, Doctor, Nurse, Receptionist)

10.2 Native Applications

- **Desktop**: Electron, Tauri or Flutter Desktop
- **Mobile**: React Native or Flutter
- **PWA**: Service Workers, offline mode, push notifications

10.3 OCR Integration

OCR integration (Tesseract, Google Cloud Vision, Azure) for text extraction from:

- Medical prescriptions
- Laboratory reports
- Discharge letters
- Insurance documents

10.4 Additional Features

- **HL7 FHIR**: Interoperability with other healthcare systems
- **Analytics Dashboard**: Population statistics, disease trends
- **Telemedicine**: Video consultations, secure messaging
- **Enhanced AI**: Multiple providers, specialty-specific models
- **Appointment Scheduling**: Appointment and resource management
- **Prescription Management**: e-Prescription, drug interaction checking
- **Performance**: Redis caching, CDN, query optimization
- **Accessibility**: WCAG 2.1 AA compliance

10.5 Implementation Roadmap

Phase	Feature	Priority
Q1	Enhanced Security, OCR	High
Q2	PWA, HL7 FHIR	Medium
Q3	Desktop App, Analytics	Medium
Q4	Telemedicine, Mobile	Low

Table 10.1: Implementation Roadmap

Chapter 11

Conclusion

11.1 Summary

MAV represents an advanced clinical decision support system, combining modern web technologies with artificial intelligence to assist healthcare professionals.

Key achievements:

- **Robust Architecture**: Gateway pattern with well-defined design patterns
- **AI Diagnostics**: Google Gemini integration for multimodal analysis
- **Patient Management**: Complete management with support for Italian and foreign citizens
- **Security-First**: Authentication, authorization and audit logging
- **Modern Deployment**: Docker containerization

11.2 Technical Excellence

The project demonstrates adherence to best practices:

- Clean code architecture with separation of concerns
- Design pattern implementation
- Error handling with retry mechanisms
- Scalable database design

11.3 Future Vision

The roadmap positions MAV for continuous evolution:

- Enterprise-grade enhanced security

- Native applications for accessibility
- OCR for paper documents
- Healthcare interoperability standards

Document generated on January 1, 2026

Appendix A

API Reference

A.1 Principali Endpoint

Method	**Endpoint**	**Description**
POST	/api/auth/register	Registrazione medico
POST	/api/auth/login	Login medico
POST	/api/patient/search	Ricerca paziente
POST	/api/patient/create	Creazione paziente
POST	/api/record/add	Aggiunta cartella clinica
POST	/api/diagnosis/generate	Generazione diagnosi AI
GET	/api/export/:fiscal_code	Export PDF

Table A.1: API Endpoints Principali