Service Design and Engineering

Academic Year 2025/2026

Microservices Medical App

Architectural and Technical Report

Student:
Omar Bonfanti
Student ID: 268407

1 Introduction and Architecture

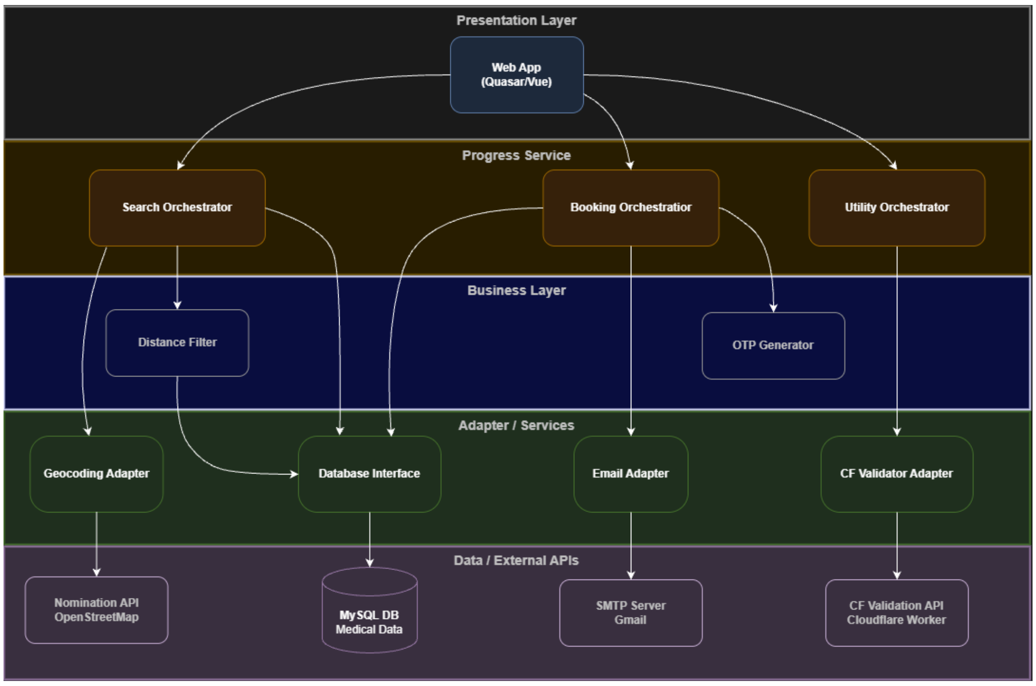
The **Medical App** project is a web platform designed for searching and booking medical appointments, developed strictly following the **SOA (Service Oriented Architecture)** paradigms. It permits users to find nearby doctors based on their location, book available time slots, and receive confirmation via email with a secure OTP code for two-factor authentication. It consists of four distinct microservices, isolated within **Docker** containers, communicating via HTTP/REST protocols.

1.1 Microservices Diagram

The Figure 1.1 illustrates the detailed decomposition of the system components and their hierarchical interactions across the five architectural layers.

As shown in the diagram, the architecture is strictly hierarchical:

- **Presentation Layer:** The *Web App (Quasar/Vue)* resides at the top and interacts exclusively with the Process Service layer.
- **Process Service:** This layer acts as the API Gateway and orchestrates calls to the underlying services without containing any business logic itself.
- **Business Layer:** Contains the pure logic components, specifically the *Distance Filter* for geospatial calculations and the *OTP Generator* for security codes.
- **Adapter / Services:** Isolates the system from external technologies. It includes specific adapters for Geocoding, Email (SMTP), Cloudflare Validation (CF), and the Database Interface.
- **Data / External APIs:** Represents the physical resources and external services consumed by the system, including the MySQL Database, OpenStreetMap API, Gmail SMTP, and the Cloudflare Worker.



2 Description of the 4 Layers

The backend strictly adheres to the logical layer subdivision required by the project specifications.

2.1 Process Centric Services Layer (Gateway)

Port: 3000
This service acts as the centralized **API Gateway** and **Process Orchestrator**, serving as the single entry point for the Frontend application (Quasar).

- **Role:** Service Orchestration and Aggregation. Unlike a simple proxy, this layer manages complex business workflows by chaining requests across multiple microservices (Data, Business, Adapter) and composing their responses into a unified result.
- **Orchestration Flows:** As illustrated in the architectural diagram, this layer implements three distinct logical flows:
  - **Search Orchestrator:** Coordinates the discovery process by fetching geocoordinates (Adapter), retrieving raw appointment slots (Data), and applying distance filtering logic (Business).
  - **Booking Orchestrator:** Manages the transactional sequence of reserving a slot, retrieving appointment details, and triggering the email confirmation (Adapter).
  - **Utility Orchestrator:** Handles auxiliary validation tasks, such as the Doctor's Tax Code (CF) verification and address autocomplete.
- **Security:** It acts as the security boundary for the backend, implementing an authentication middleware that verifies the `x-api-key` header before allowing access to the protected internal network.

2.2 Business Logic Services Layer

Port: 3003
It contains the pure application logic (the "brain" of the application).

- **Responsibility:**
  - Geographic distance calculation (Haversine Formula) to filter doctors within a specific radius.
  - Secure generation of OTP codes for Two-Factor Authentication.

2.3 Data Services Layer

Port: 3001
It manages data persistence and is the only service authorized to communicate with the Database.

- **Database:** MySQL 8.0 (Containerized with Persistent Volume).
- **Responsibility:** Executes SQL queries to read free slots, update booking statuses, and manage registry data.

2.4 Adapter Services Layer

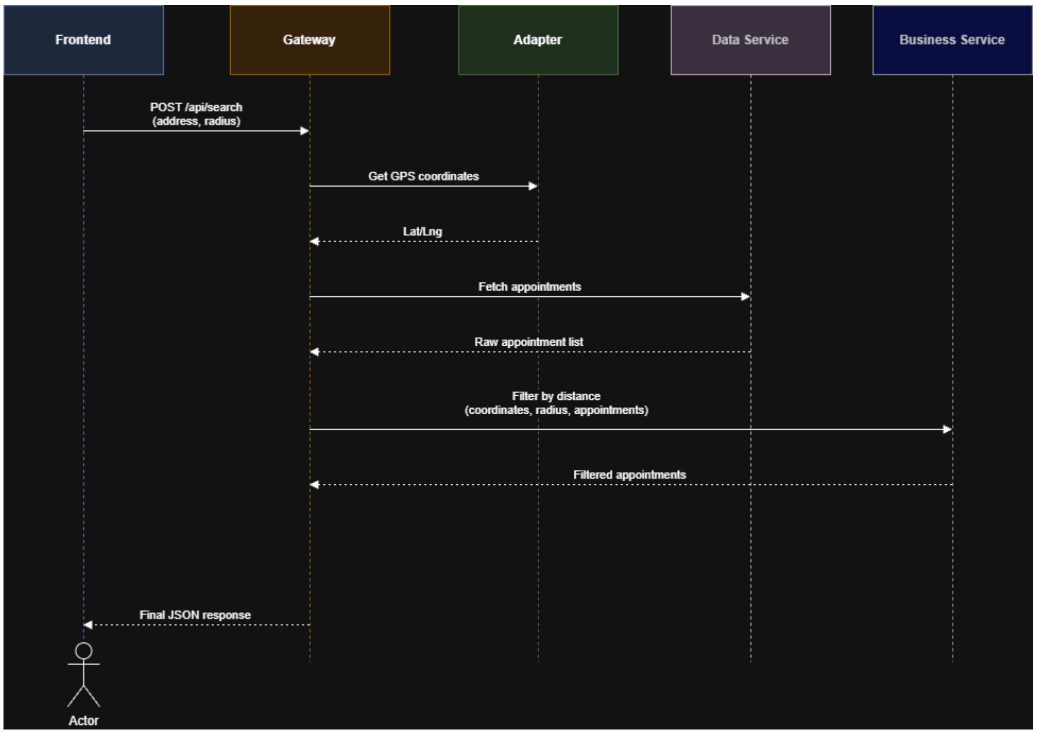
Port: 3002
It links the application to the outside world, isolating third-party dependencies.

- **Geocoding:** Interfaces with the Nominatim API (OpenStreetMap) to convert addresses into coordinates.
- **Email:** Interfaces with an SMTP server (e.g., Gmail) to send notifications.
- **CF Validation:** Interfaces with an external Cloudflare Worker API to verify the formal validity of the Doctor's Tax Code (Codice Fiscale).

3 Orchestration Flows

A key example of orchestration is the Search Flow (`POST /api/search`). The Gateway coordinates three different services to fulfill a single user request:

1. The Gateway receives the address and search radius.
2. It calls the **Adapter** to obtain GPS coordinates (Lat/Lng).
3. It calls the **Data Service** to fetch the raw list of appointments.
4. It sends everything to the **Business Service**, which filters out slots that are too far away.
5. It returns the final JSON to the Frontend.


Frontend
Gateway
Adapter
Data Service
Business Service
POST /api/search (address, radius)
Get GPS coordinates
Lat/Lng
Fetch appointments
Raw appointment list
Filter by distance (coordinates, radius, appointments)
Filtered appointments
Final JSON response
Actor