**Sistema 1: Plataforma de Gestión de Citas Médicas**

**Problema a Resolver**

En hospitales y clínicas, la gestión de citas médicas puede ser ineficiente, con problemas de duplicación, cancelaciones tardías y dificultades para ajustar horarios. Este sistema facilita la gestión de citas, el seguimiento de pacientes y la optimización del tiempo de los médicos.

**Base de Datos Usada**

1. **MongoDB**: Para almacenar los datos de pacientes, médicos y citas.
2. **Redis**: Para manejar colas de espera y notificaciones en tiempo real.

**Arquitectura**

* **Frontend**: Aplicación web o móvil (React Native) donde pacientes pueden reservar y gestionar citas.
* **Backend**: API desarrollada en Node.js.
* **Bases de Datos**:
  + MongoDB: Estructura los datos de los pacientes, médicos y citas.
  + Redis: Implementa colas de espera para gestionar citas canceladas y envía notificaciones en tiempo real.

**Diagramas**

**Diagrama de Arquitectura**

* Usuario → React Native → API (Node.js) → MongoDB/Redis

**Modelo de Datos (MongoDB)**

{

"cita\_id": "001",

"paciente": {

"id": "123",

"nombre": "Álvaro",

"edad": 30,

"historial\_médico": ["diabetes"]

},

"médico": {

"id": "456",

"nombre": "Dr. López",

"especialidad": "Endocrinología"

},

"fecha\_hora": "2024-11-20T10:00:00",

"estado": "Confirmada"

}

**Funcionamiento del Sistema**

1. Un paciente reserva una cita.
2. El backend verifica disponibilidad en MongoDB.
3. Si no hay disponibilidad, se añade a la cola de espera en Redis.
4. Redis envía notificaciones en tiempo real si hay cancelaciones.

**Mockup**

Se simula la reserva de una cita para el "Dr. López", mostrando disponibilidad y el envío de una notificación de confirmación.

**Sistema 2: Monitoreo de Pacientes en Tiempo Real**

**Problema a Resolver**

Los hospitales necesitan monitorear las constantes vitales de pacientes en tiempo real para detectar anomalías y prevenir emergencias.

**Base de Datos Usada**

1. **Cassandra**: Para almacenar los datos históricos de las constantes vitales de los pacientes.
2. **Elasticsearch**: Para buscar y filtrar rápidamente a los pacientes que necesitan atención inmediata.

**Arquitectura**

* **Frontend**: Dashboard en Angular donde médicos pueden visualizar las constantes vitales.
* **Backend**: API desarrollada en Python (FastAPI) que procesa datos en tiempo real.
* **Bases de Datos**:
  + Cassandra: Almacena datos históricos como pulso, presión arterial y saturación.
  + Elasticsearch: Realiza búsquedas rápidas de pacientes con valores fuera de rango.

**Diagramas**

**Diagrama de Arquitectura**

* Dispositivos IoT → Backend (FastAPI) → Cassandra/Elasticsearch → Dashboard

**Modelo de Datos (Cassandra)**

CREATE TABLE constantes\_vitales (

paciente\_id UUID,

timestamp TIMESTAMP,

pulso INT,

presión\_sistólica INT,

presión\_diastólica INT,

saturación\_oxígeno FLOAT,

PRIMARY KEY (paciente\_id, timestamp)

);

**Modelo de Datos (Elasticsearch)**

{

"paciente\_id": "123",

"nombre": "Álvaro",

"pulso": 120,

"presión\_sistólica": 150,

"presión\_diastólica": 95,

"saturación\_oxígeno": 89,

"estado": "Emergencia"

}

**Funcionamiento del Sistema**

1. Dispositivos IoT envían datos de constantes vitales.
2. Cassandra almacena los datos históricos.
3. Elasticsearch identifica a los pacientes con valores críticos.
4. El dashboard muestra alertas a los médicos para atención inmediata.

**Mockup**

Se simula que el dispositivo IoT detecta un pulso alto (120 bpm) y genera una alerta que se muestra en el dashboard.