Documento de Requerimientos Técnicos: Motor de Disponibilidad Concurrente Global

1. Arquitectura de Datos y Sincronización Horaria

Para garantizar la precisión entre pacientes en cualquier parte del mundo y doctores en México, se establece una política de **UTC Absoluto**.

1.1 Reglas de Oro

- **Persistencia:** Todas las fechas y horas en la base de datos (específicamente citas confirmadas) deben almacenarse en formato **UTC (ISO 8601)**.
- **Normalización:** La disponibilidad recurrente de los doctores se almacena en su hora local, pero debe incluir el ID de su zona horaria (ej: America/Mexico_City).

1.2 Estructura de Tablas

- **doctors**: id, name, timezone (VARCHAR), estado.
- **doctor_availability**: id, doctor_id, day_of_week (0-6), timezone (VARCHAR), start_time (TIME), end_time (TIME).
- **doctor_exceptions**: id,doctor_id, start_datetime_utc (DATETIME), end_datetime_utc (DATETIME),type (ENUM: 'vacation', 'holiday', 'emergency', 'personal'),scope (ENUM: 'full_day', 'partial')
- **appointments**: id, doctor_id, patient_id, scheduled_at_utc (DATETIME), status, payment_status.
- **Timezones:** id, nombre, offset_utc
- **certificates**: id, appointment_id, doctor_id, patient_id, documento_url, fecha_emission
- **payments**: id, appointment_id, monto, estado (habilitado, deshabilitado, pendiente, completado)
- **payment_logs**: id, payment_id, evento, fecha, detalle

---

2. Lógica del Sistema Backend (Doctores) – Ignora esta parte Johnson

El objetivo es capturar la "oferta" de servicios de manera flexible.

- **Múltiples Rangos:** El sistema debe permitir al doctor definir más de un intervalo de tiempo por día (ej. Turno Mañana y Turno Tarde).
- **Metadatos de Zona:** Al guardar el horario, el backend debe capturar la zona horaria activa del doctor. Si el doctor cambia de zona, el sistema debe preguntar si desea ajustar sus horas o mantener las actuales.

---

3. Lógica del Motor de Búsqueda (Pacientes)

El sistema debe calcular la disponibilidad basándose en la **concurrencia de doctores** y el **desfase horario**.

3.1 Proceso de Cálculo de Disponibilidad (Algoritmo)

Cuando un paciente en cualquier parte del mundo solicita disponibilidad para una fecha específica:

1. **Detección de Zona:** El sistema detecta el timezone del paciente (ej: Europe/Madrid).
2. **Generación de Ventana de Consulta:** Se genera una ventana de 24 horas en el tiempo del paciente y se traduce a su equivalente en UTC.
3. **Conversión de Oferta (Mapeo UTC):**
    - El sistema busca en doctor_availability todos los doctores activos.
    - Convierte los horarios locales de cada doctor a UTC de forma dinámica.
    - *Ejemplo:* Un doctor en CDMX (UTC-6) disponible a las 08:00 AM se mapea internamente como las 02:00 PM UTC.
4. Cálculo de Concurrencia:

    Para cada intervalo (Slot) de consulta (ej. cada 30 min), el sistema ejecuta:

    - **Capacidad Bruta (C_B):** Sumar cuántos doctores tienen disponibilidad en ese slot exacto tras la conversión a UTC.
    - **Doctores Bloqueados (B):** Contar cuántos de esos 6 doctores tienen un registro en doctor_exceptions que cubra ese lunes a las 10:00 AM UTC.
    - Ejemplo: 1 doctor se fue de vacaciones.
    - **Capacidad Bruta Real (C_B):** $C_B - B$
    - Cálculo: $6 - 1 = 5$ doctores realmente trabajando.
    - **Ocupación Actual (O):** Contar en la tabla appointments cuántas citas existen en ese slot UTC con estatus 'Confirmado'.
    - **Disponibilidad Neta (D_N):** $D_N = C_B - O$.

3.2 Regla de Negocio del "Séptimo Paciente"

- Si un intervalo tiene $D_N > 0$, el paciente ve el horario como **"Disponible"**.
- Si 6 doctores están disponibles y ya hay 6 citas reservadas ($D_N = 0$), el slot se marca como **"Agotado"** para el paciente 7.

4. Logica para el calculo con Bloqueos especiales:

A. Prioridad de Bloqueo

La tabla doctor_exceptions tiene jerarquía sobre doctor_availability. Si existe un bloqueo parcial o total para un médico en un rango de tiempo, ese médico debe ser excluido del conteo de disponibilidad para el Pool de pacientes en ese intervalo exacto.

B. Impacto en Citas Existentes

Al crear un bloqueo en el backend de doctores, el sistema debe verificar si ya existen registros en appointments para ese rango. De ser así, el sistema debe retornar un mensaje de advertencia: *'Existen X citas programadas en este rango. ¿Desea cancelarlas o reasignarlas antes de confirmar el bloqueo?'*."

C. Bloqueos Globales

Si el administrador del sistema define un día festivo nacional, se debe permitir crear un registro en doctor_exceptions donde doctor_id = NULL (o un flag global), lo cual restará la disponibilidad de **todos** los médicos para esa fecha en el sistema de pacientes.

4.1. Flujo Visual de la Lógica Completa

1. **Paciente solicita:** Lunes 15 de Octubre, 10:00 AM (Zona Horaria Paciente).
2. **Conversión:** Sistema traduce a UTC (ej. 16:00 UTC).
3. **Filtro 1 (Horarios):** Busca doctores con horario de lunes que incluya las 16:00 UTC.
4. **Filtro 2 (Bloqueos):** De esos doctores, quita a los que tengan un bloqueo (vacaciones/emergencia) a las 16:00 UTC.
5. **Filtro 3 (Citas):** Del número restante, resta las citas ya guardadas en appointments a esa hora.
6. **Respuesta:** Si el número es > 0, se muestra el horario al paciente.

Backend de Doctores:

En la página que diseñamos al inicio, cuando el doctor use el calendario para "Bloquear", el frontend debe enviar los datos al endpoint de doctor_exceptions. Es vital que el selector de fecha de bloqueos también maneje la zona horaria del doctor para que la conversión a UTC sea perfecta.

---

5. Especificación de API (Endpoint Pacientes)

Request:

GET /api/v1/availability?date=2025-10-15&timezone=Europe/Madrid&specialty=general

Lógica de Respuesta:

El JSON debe devolver los horarios ya convertidos a la hora del paciente.

JSON

```
{
  "date": "2025-10-15",
  "patient_timezone": "Europe/Madrid",
  "available_slots": [
    { "time": "14:00", "status": "available", "remaining_slots": 4 },
    { "time": "14:30", "status": "full", "remaining_slots": 0 }
  ]
}
```

---

6. Consideraciones de Implementación para el Equipo

- **Balanceo de Carga:** Al confirmar la cita, el backend debe asignar automáticamente al doctor que tenga el menor número de citas ese día para equilibrar el trabajo.
- **Cambios de Horario de Verano (DST):** Es obligatorio usar librerías que manejen el historial de cambios de hora (como Moment-Timezone o Carbon con base de datos IANA actualizada).
- **Performance:** Debido a que el cálculo de disponibilidad es intensivo, se recomienda usar **Caching (Redis)** para los resultados de capacidad bruta por día.

7. Calculo de disponibilidad para agenda de pacientes

No se calcula "celda por celda", se calcula por "Rangos"

En lugar de preguntar 168 veces (24h x 7 días), el backend debe hacer **una sola consulta SQL inteligente** que traiga todos los datos del rango de fechas solicitado (ej. del 1 al 7 de enero).

La lógica de "Aplanamiento"

En lugar de un bucle infinito, los programadores deben seguir esta estrategia:

1. **Obtener la Oferta Bruta:** Una sola consulta trae todos los horarios de los doctores que coincidan con la especialidad.

   Esta consulta trae todos los horarios semanales de los doctores activos. No filtramos por hora, solo por especialidad si es necesario.

```
SELECT
   d.id AS doctor_id,
   d.timezone AS doctor_tz,
   da.day_of_week,
   da.start_time,
   da.end_time
FROM doctors d
JOIN doctor_availability da ON d.id = da.doctor_id
```

WHERE d.active = 1 AND d.specialty_id = :specialty_id;

2. **Obtener las Excepciones y Citas:** Una sola consulta trae todos los bloqueos y citas ocupadas en ese rango de 7 o 28 días.

   ### Obtener Excepciones y Citas (SQL)

   Aquí traemos todo lo que "resta" capacidad en el rango de fechas que el paciente está viendo (ej: una semana).

```
-- Consultar Citas Ocupadas
SELECT doctor_id, scheduled_at_utc
FROM appointments
WHERE scheduled_at_utc
BETWEEN :start_utc AND :end_utc AND status = 'confirmed';

-- Consultar Bloqueos/Vacaciones
SELECT doctor_id, start_datetime_utc, end_datetime_utc
FROM doctor_exceptions
WHERE (start_datetime_utc <= :end_utc AND end_datetime_utc >= :start_utc);
```

3. **Cruzar en Memoria (Backend):** El programador usa un "Mapa" o "Diccionario" en el código (no en la base de datos) para restar los valores. Esto es mil veces más rápido que hacer cientos de consultas SQL.

   El "secreto" es crear una matriz de tiempo. En una cuadrícula donde las columnas son los días y las filas son los intervalos de 30 minutos.

```php
<?php
// 1. Definir el rango de tiempo solicitado por el paciente (en su zona horaria)
$patientTimezone = new DateTimeZone('Europe/Madrid');
$startDate = new DateTime('2025-10-15 00:00:00', $patientTimezone);
$endDate = (clone $startDate)->modify('+7 days');

// 2. Estructura para almacenar el conteo de disponibilidad
$availabilityMap = []; // Ejemplo: $availabilityMap['2025-10-15']['10:00'] = count;

// 3. PROCESAR OFERTA BRUTA
// Iteramos sobre los días del rango y buscamos qué doctores trabajan ese "Día de la semana"
foreach ($doctorsAvailability as $doc) {
   // Convertir el horario del doctor a la zona del paciente para saber en qué momento cae
   // ... lógica de traslape de zonas horarias ...
   // Por cada slot de 30 min que el doctor trabaje, sumamos:
   $availabilityMap[$fecha][$hora]++;
}

// 4. RESTAR EXCEPCIONES (Bloqueos)
foreach ($exceptions as $ex) {
   $start = new DateTime($ex['start_datetime_utc'], new DateTimeZone('UTC'));
   $end = new DateTime($ex['end_datetime_utc'], new DateTimeZone('UTC'));
```

```
    // Convertir a zona del paciente e identificar qué slots bloquea
    // Restamos 1 a cada slot donde este doctor específico estaba "disponible"
    $availabilityMap[$fecha_p][$hora_p]--;
  }

  // 5. RESTAR CITAS (Ocupado)
  foreach ($appointments as $app) {
    $dateUtc = new DateTime($app['scheduled_at_utc'], new DateTimeZone('UTC'));
    $dateUtc->setTimezone($patientTimezone);

    $f = $dateUtc->format('Y-m-d');
    $h = $dateUtc->format('H:i');

    // Restamos la cita confirmada
    if (isset($availabilityMap[$f][$h])) {
      $availabilityMap[$f][$h]--;
    }
  }

  // 6. RESULTADO FINAL
  // Los slots que tengan valor > 0 son los que el paciente puede ver.
```

El concepto de "Slots Pre-generados"

El API de disponibilidad no debe calcular disponibilidad en tiempo real desde cero. Debe generar una **Matriz de Disponibilidad** para el rango solicitado, restando la ocupación de la capacidad teórica en una sola operación de conjunto.

4. Ejemplo de cómo se vería la respuesta del API (Optimizado)

En lugar de enviarle al frontend datos para que él calcule, el backend entrega los "Slots" ya procesados:

5. Recomendación de Rendimiento: Siguiente Versión

Como la disponibilidad de los doctores (la oferta) no cambia cada segundo, pero las citas (la ocupación) sí, se sugiere:

- **Cachear la Capacidad Bruta:** Guardar en una memoria rápida cuántos doctores hay por hora.
- **Restar en Tiempo Real:** Solo restar las citas activas al momento de la petición.

Resumen:

"Para la visualización del calendario del paciente (vista semanal o mensual), el backend debe implementar un **Query de Agregación**. Se debe evitar el 'N+1 problem' (hacer una consulta por cada hora). El endpoint de disponibilidad debe procesar el cruce de doctor_availability, doctor_exceptions y appointments en una sola transacción o bloque lógico, devolviendo un mapa de slots disponibles normalizados a la zona horaria del paciente."