LÍMITE DE ENTREGA: JUEVES 27 de Noviembre a las 11:00 am (LIMA - COLOMBIA).

1. Contexto

Solo puedes usar la **web pública de Doctoralia** para obtener información, y luego debes cargar esa información en tu propia base de datos.

Esta prueba busca evaluar cómo:

- Analizas una web real para obtener datos de forma automatizada.
- Modelas la información en una base de datos relacional existente.
- Diseñas scripts y procesos de migración.
- Orquestas todo con Docker para que el proceso sea reproducible.
- Escribes código limpio y mantenible en **TypeScript** usando **Prisma ORM**.

---

2. Objetivo general

Construir un **pipeline de migración de datos** que:

1. Obtenga de forma automatizada datos públicos de **médicos en Doctoralia**.
2. A partir de esos datos, genere información adicional ficticia (pacientes y citas).
3. Inserte todo en una base de datos **PostgreSQL** usando el esquema que te entregamos.
4. Se ejecute de forma automática a través de **Docker Compose**, de tal manera que con un solo comando se levante la base de datos, se cree el esquema y se ejecute todo el proceso de migración.

Importante: no se trata de hacer una aplicación web, sino de construir **scripts y procesos de backend / data migration** bien organizados.

SE RECOMIENDA EL USO INTENSIVO DE LAS HERRAMIENTAS DE IA QUE PREFIERAN.

---

3. Alcance funcional

3.1. Datos que debes obtener de Doctoralia

De forma automatizada (usando la técnica que estimes conveniente), debes obtener información pública de médicos que aparecen en Doctoralia, incluyendo al menos:

- **Médico**
    - Nombre completo
    - Especialidad principal
    - Ciudad
    - Dirección o centro médico principal
    - Teléfono con prefijo de país (ej. +51 y número)
    - Valoración / estrellas
    - Número de opiniones
    - URL del perfil público en Doctoralia
- **Tratamientos / servicios** ofrecidos por cada médico
    - Nombre del tratamiento / servicio
    - (Si es posible) precio
    - (Si es posible) moneda
    - (Si es posible) duración aproximada
- **Disponibilidad / horarios de atención**
    - Bloques de tiempo en los que el médico atiende
    - Tipo de atención, si es relevante (presencial / online)

No necesitas cubrir todo Doctoralia. Es suficiente si trabajas con **un subconjunto razonable** (por ejemplo, 2–3 ciudades y una o varias especialidades).

La idea es que el volumen de datos sea suficiente para poblar la base de datos con:

- Varios médicos (por ejemplo un par de docenas si es posible).
- Varios tratamientos por médico.
- Varias franjas de disponibilidad por médico.

3.2. Datos que debes generar tú

Además de lo que obtengas desde Doctoralia, debes **generar datos ficticios** para completar el modelo:

- **Pacientes**
    - Nombre completo
    - Documento/ID ficticio
    - Teléfono
    - Email
- **Citas médicas (appointments)**
    - Cada cita debe:
        - Estar asociada a un médico existente.
        - Estar asociada a un paciente existente.
        - Estar asociada a un tratamiento que **pertenezca a ese médico**.
        - Caer dentro de una franja de **disponibilidad real** del médico (que hayas obtenido de Doctoralia).
        - Tener fecha/hora inicio y fin razonables.
        - Tener un estado (por ejemplo: programada, completada, cancelada).

Puedes decidir el volumen de citas y pacientes, pero debería ser suficiente para ver el modelo funcionando (por ejemplo, cientos de citas distribuidas entre distintos médicos y pacientes).

---

4. Base de datos y esquema

Utilizarás **PostgreSQL** como motor de base de datos.

Te proporcionamos un script SQL que define el esquema mínimo requerido (tablas, tipos y relaciones). **No debes modificar este esquema** (no alteres columnas, tipos, relaciones ni nombres de tablas).

- Tu trabajo es **adaptar tus scripts y tu modelo Prisma** a este esquema.
- Puedes agregar índices adicionales vía Prisma si lo requieres a nivel lógico, pero la estructura base (tablas/campos) debe coincidir con el SQL proporcionado.

> En el documento que recibes, debajo de estas instrucciones encontrarás el script `schema.sql` con la definición completa de:
>
> - `clinic.doctors`
> - `clinic.treatments`
> - `clinic.doctor_availability`
> - `clinic.patients`
> - `clinic.appointments`
>
> Ese script es el **punto de verdad** del diseño de la base de datos.

---

5. Requisitos técnicos

5.1. Lenguaje y stack

Debes usar obligatoriamente:

- **Node.js** (versión reciente LTS o superior, por ejemplo 20.x).
- **TypeScript**.
- **Prisma ORM** para interactuar con la base de datos PostgreSQL.
- **Docker** y **Docker Compose** para orquestar el entorno.

Está permitido utilizar librerías adicionales de NPM que consideres útiles (axios, node-fetch, zod, etc.), siempre que las declares correctamente en `package.json`.

5.2. Uso de Prisma

- El acceso a la base de datos debe hacerse con **Prisma**.
- Debes definir un `schema.prisma` compatible con la base de datos generada por nuestro `schema.sql`.
- Puedes usar enfoques como:
    - `prisma db pull` para generar el schema desde la base ya creada; o
    - escribir el `schema.prisma` a mano, asegurándote de que coincida con la estructura SQL.
- Los inserts / queries de tu código deben hacerse usando el **Client de Prisma** (`@prisma/client`).

5.3. Contenedores

Esperamos al menos dos servicios en `docker-compose.yml`:

- `db`: contenedor de PostgreSQL con el esquema creado a partir del SQL proporcionado.
- `app`: contenedor Node.js/TypeScript que contiene tu código y ejecuta todo el proceso de migración.

Al levantar `docker-compose up`, el flujo esperado es:

1. Se levanta la base de datos.
2. Se ejecuta el script `schema.sql` para crear el esquema.
3. El servicio `app` espera a que la base de datos esté lista.
4. El servicio `app` ejecuta **en orden**:
    - Obtención de datos desde Doctoralia.
    - Generación de pacientes y citas ficticias.
    - Inserción de todos los datos en PostgreSQL usando Prisma.

No debe ser necesario ejecutar comandos manuales dentro de los contenedores para completar la migración.

5.4. Seguridad y configuración

- No incluyas credenciales sensibles en el repositorio.
- Usa variables de entorno (`.env`) para datos como:
    - URL de la base de datos.
    - Cualquier dato de configuración sensible.
- Asegúrate de que sea fácil levantar el proyecto siguiendo las instrucciones del README.

---

6. Flujo de trabajo sugerido (no obligatorio)

Esta es una guía opcional para que organices tu trabajo. No estás obligado a seguirla tal cual, pero puede ayudarte a estructurar la solución.

Paso 1 – Preparar entorno con Docker y PostgreSQL

1. Crear el archivo `schema.sql` copiando el script que te proporcionamos.

2. Crear `docker-compose.yml` para levantar PostgreSQL.

3. Verificar que la base se puede levantar y que el esquema se aplica correctamente.

Paso 2 – Configurar Prisma con TypeScript

1. Crear un proyecto Node/TypeScript (por ejemplo, usando `npm init` + configuración TS).

2. Instalar y configurar Prisma:

    - `npm install prisma @prisma/client`

    - `npx prisma init`

3. Asegurarte de que `schema.prisma` refleja el esquema de la base de datos.

4. Probar una conexión simple e insertar un registro de prueba (puede ser manual) para validar que todo está bien conectado.

Paso 3 – Obtención de datos de Doctoralia

1. Analizar Doctoralia y decidir cómo vas a obtener de forma automatizada:

    - médicos,

    - tratamientos/servicios,

    - disponibilidad/horarios.

2. Implementar uno o varios scripts en TypeScript que:

    - obtengan la información,

    - la estructuren en objetos/estructuras en memoria,

    - opcionalmente la guarden en archivos JSON intermedios (`data/doctors.json`, `data/availability.json`, etc.).

Paso 4 – Generar pacientes y citas ficticias

1. Crear lógica para generar pacientes con datos ficticios.
2. Crear lógica para generar citas usando:
    - la disponibilidad real de los médicos,
    - los tratamientos que ofrece cada médico,
    - los pacientes generados.
3. Validar que las citas:
    - respetan los intervalos de disponibilidad,
    - siempre usan un tratamiento perteneciente al médico,
    - tienen relaciones coherentes (IDs válidos).

Paso 5 – Insertar datos con Prisma

1. Implementar la inserción de datos en la base de datos usando el Cliente de Prisma.
2. Insertar en el orden adecuado (primero médicos, luego tratamientos, disponibilidad, pacientes y finalmente citas).
3. Manejar posibles errores (duplicados, etc.) de forma segura.

Paso 6 – Orquestar todo en el contenedor `app`

1. Crear un `Dockerfile` para el servicio `app`.
2. Configurar el `entrypoint` (por ejemplo, un script `npm run migrate` o un archivo `index.ts` compilado a JS) que ejecute todo el pipeline de migración.
3. Integrar el flujo completo con `docker-compose.yml` para que un solo comando (`docker-compose up`) sea suficiente.

---

7. Entregables esperados

Tu entrega debe incluir, como mínimo:

1. **URL del repositorio público en github con todos los archivos que se indican a continuación.**
2. **Código fuente TypeScript** organizado (scripts para obtención, generación y carga de datos).
3. **Configuración de Prisma** (`schema.prisma`, migraciones si las usas, generación de client).
4. **`docker-compose.yml`** con los servicios `db` y `app`.
5. **`Dockerfile`** del servicio `app`.
6. Archivo `schema.sql` (copiado desde el documento que se te entrega).
7. Opcionalmente, archivos JSON intermedios (`data/*.json`) o scripts para generarlos.
8. Un archivo **`README.md`** claro, que explique:
    - Requisitos previos.
    - Cómo levantar el proyecto.
    - Qué hace el pipeline de migración.
    - Limitaciones o supuestos importantes.

---

8. Criterios de evaluación

Al revisar tu solución tendremos en cuenta, entre otros, los siguientes puntos:

8.1. Correctitud funcional

- ¿Los médicos en la base de datos corresponden a datos obtenidos de Doctoralia?
- ¿Se han cargado tratamientos por médico?
- ¿La disponibilidad de los médicos refleja (en lo posible) lo que aparece en Doctoralia?

- ¿Las citas usan:
    - un médico válido,
    - un paciente válido,
    - un tratamiento que pertenece a ese médico,
    - un horario coherente con la disponibilidad?

8.2. Uso de TypeScript y Prisma

- Uso adecuado de **tipos** en TypeScript.
- Manejo correcto de la conexión, consultas e inserciones con Prisma.
- Estructuración del código (modularización, separación de responsabilidades).

8.3. Calidad del diseño y del código

- Claridad y legibilidad del código.
- Manejo de errores y casos borde.
- Uso razonable de configuraciones y variables de entorno.

8.4. Docker y reproducibilidad

- ¿Se puede levantar el proyecto con un comando tipo `docker-compose up`?
- ¿El pipeline se ejecuta de principio a fin sin pasos manuales adicionales?
- ¿La documentación es suficiente para que otra persona pueda correrlo?

8.5. Extras (no obligatorios)

No son requeridos, pero los valoraremos positivamente si los incluyes:

- Logs claros del progreso de la migración.

- Tests automatizados (aunque sean parciales).
- Manejo de reintentos ante errores de red al obtener datos externos.
- Configuración de scripts de `npm` para facilitar la ejecución local.

---

9. Notas y buenas prácticas

- Por favor, respeta un uso razonable de la web de Doctoralia. El objetivo es educativo y de evaluación técnica.
- No es necesario que el volumen de datos sea masivo; prioriza la calidad, consistencia y automatización del proceso.
- Si encuentras limitaciones técnicas fuertes (por ejemplo, cambios en la web, mecanismos anti-automatización), documenta claramente:
    - qué intentaste,
    - hasta dónde llegaste,
    - qué harías en un entorno controlado.

---

10. Esquema SQL

A continuación, el script `schema.sql` con la definición de la base de datos en PostgreSQL (esquema `clinic`).

> **Instrucción para el candidato:** copia ese bloque de SQL tal cual a un archivo `schema.sql` dentro de tu proyecto y úsalo para crear la base de datos.

-- Esquema base para la prueba técnica de migración
-- Motor: PostgreSQL

CREATE SCHEMA IF NOT EXISTS clinic;
SET search_path TO clinic, public;

```
-- Tipos auxiliares
CREATE TYPE appointment_status AS ENUM ('scheduled', 'completed', 'cancelled');
CREATE TYPE visit_modality     AS ENUM ('in_person', 'online');

-- Médicos obtenidos desde Doctoralia
CREATE TABLE doctors (
    id                  BIGSERIAL PRIMARY KEY,
    full_name           TEXT        NOT NULL,
    specialty           TEXT        NOT NULL,
    city                TEXT        NOT NULL,
    address             TEXT,
    phone_country_code  VARCHAR(8),     -- ej: "+51"
    phone_number        VARCHAR(32),    -- número sin el prefijo
    rating              NUMERIC(2,1),   -- ej: 4.7
    review_count        INTEGER,
    source_profile_url  TEXT        NOT NULL
);

-- Tratamientos / servicios que ofrece cada médico
CREATE TABLE treatments (
    id          BIGSERIAL PRIMARY KEY,
    doctor_id   BIGINT      NOT NULL REFERENCES doctors(id),
    name        TEXT        NOT NULL,
    price       NUMERIC(10,2),          -- opcional, si se puede inferir
    currency    CHAR(3),                -- ej: "PEN"
    duration_minutes SMALLINT           -- opcional
);

-- Evitar tratamientos duplicados por médico
CREATE UNIQUE INDEX ux_treatments_doctor_name
    ON treatments(doctor_id, name);

-- Disponibilidad real del médico (obtenida de Doctoralia)
-- Cada fila representa un bloque de tiempo disponible
CREATE TABLE doctor_availability (
    id          BIGSERIAL PRIMARY KEY,
    doctor_id   BIGINT          NOT NULL REFERENCES doctors(id),
    start_at    TIMESTAMPTZ     NOT NULL,
    end_at      TIMESTAMPTZ     NOT NULL,
    modality    visit_modality  NOT NULL
);

CREATE INDEX idx_doctor_availability_doctor_id
    ON doctor_availability(doctor_id);
```

```
CREATE INDEX idx_doctor_availability_start_at
   ON doctor_availability(start_at);

-- Pacientes ficticios generados por el candidato
CREATE TABLE patients (
   id              BIGSERIAL PRIMARY KEY,
   full_name       TEXT        NOT NULL,
   document_number VARCHAR(32),
   phone_number    VARCHAR(32),
   email           TEXT
);

-- Citas médicas ficticias, basadas en la disponibilidad real del médico
-- Cada cita:
--  - referencia un médico
--  - referencia un paciente
--  - referencia un tratamiento del mismo médico
CREATE TABLE appointments (
   id              BIGSERIAL PRIMARY KEY,
   doctor_id       BIGINT              NOT NULL REFERENCES doctors(id),
   patient_id      BIGINT              NOT NULL REFERENCES patients(id),
   treatment_id    BIGINT              NOT NULL REFERENCES treatments(id),
   start_at        TIMESTAMPTZ         NOT NULL,
   end_at          TIMESTAMPTZ         NOT NULL,
   status          appointment_status  NOT NULL DEFAULT 'scheduled'
);

CREATE INDEX idx_appointments_doctor_id
   ON appointments(doctor_id);

CREATE INDEX idx_appointments_patient_id
   ON appointments(patient_id);

CREATE INDEX idx_appointments_treatment_id
   ON appointments(treatment_id);

CREATE INDEX idx_appointments_start_at
   ON appointments(start_at);
```