

Base de Datos y Programación Web – Descripción del Proyecto

Escuela de Ingeniería Civil Informática

Universidad de Valparaíso

Alejandro Dinamarca - Lucas Rojas

PLANTEAMIENTO Y DESARROLLO DE HISTORIAS DE USUARIOS

1 CONTEXTO GENERAL

ConsultorioDigital es la plataforma que facilitará el acceso a los centros asistenciales primarios - tales como consultorios o Centros de Salud Familiar (CESFAM) - de salud en Chile, a partir de medios digitales.

Normalmente, el primer punto de contacto de los pacientes con el sistema de salud es a partir de los consultorios, CESFAM, postas rurales, etc. No obstante, el acceso a ellos es siempre en formato presencial, dado su característica de atender, con mayor frecuencia, urgencias, accidentes, entre otros.

No obstante, en la pandemia del COVID-19, se dejó en evidencia la falta de cobertura y acceso a estos centros, como, por ejemplo, en la implementación de vacunatorios móviles para acercar la vacunación a las personas, o inclusive, en los PCR Móvil para tener mayor muestra de los infectados en pandemia. Esto, dado que los centros de salud eran vistos como factor de riesgo para contagiarse con el virus.

Una publicación oficial del Colegio Médico de Chile declara que, a partir de la primera ola de la pandemia, la población estaba "...temerosa de asistir a los centros asistenciales (...)", y además que existía la necesidad de "...considerar procesos asistenciales que permitan mantener una menor presencia física en los centros" [1].

Es aquí donde ConsultorioDigital permite aliviar las afluencias a los servicios de salud, a partir de la gestión de citas, solicitando hora directamente en el sitio web donde el sistema internamente coordina los horarios con los consultorios comunales.

2 TECNOLOGÍAS

- **Django:** Un framework web de alto nivel en Python que será el núcleo del backend, gestionando la lógica, la autenticación, y la interacción con la base de datos.
- **Python:** El lenguaje de programación principal del proyecto, utilizado para desarrollar el backend en Django.
- **JavaScript:** Utilizado en el frontend para mejorar la interactividad y la experiencia del usuario en la plataforma.
- **SQLite3:** Una base de datos ligera que se utilizará para almacenar la información relacionada con usuarios, centros asistenciales, citas y demás datos críticos del sistema.

3 COMPONENTES WEB

3.1 REQUISITOS FUNCIONALES Y NO FUNCIONALES

Con el objetivo de definir y detallar lo que el sistema debe hacer a partir de las necesidades previamente indicadas, los siguientes requisitos funcionales y no funcionales son descritos

Los requisitos funcionales son servicios que el sistema debe ser capaz de proveer al usuario, entre los cuales encontramos:

1. Reserva, Cancelación y Reprogramación de Citas y Turnos:

- El sistema debe permitir a los usuarios reservar citas médicas a través de la plataforma.
- El sistema debe permitir a los usuarios cancelar citas previamente reservadas.
- El sistema debe permitir a los usuarios reprogramar citas, cambiando la fecha y hora según disponibilidad.

2. Búsqueda de Centros Asistenciales:

- El sistema debe permitir a los usuarios buscar centros asistenciales según su ubicación actual o una dirección específica.
- El sistema debe permitir a los usuarios filtrar los centros asistenciales por el tipo de servicio requerido (e.g., especialidades médicas).
- El sistema debe mostrar los centros asistenciales con el menor tiempo de espera disponible.
- El sistema debe permitir a los usuarios buscar y seleccionar centros asistenciales utilizando criterios relevantes, como la distancia, calificación de otros usuarios, y horarios de atención.

3. Reportes, Estadísticas y Gestión de Usuarios para Administradores:

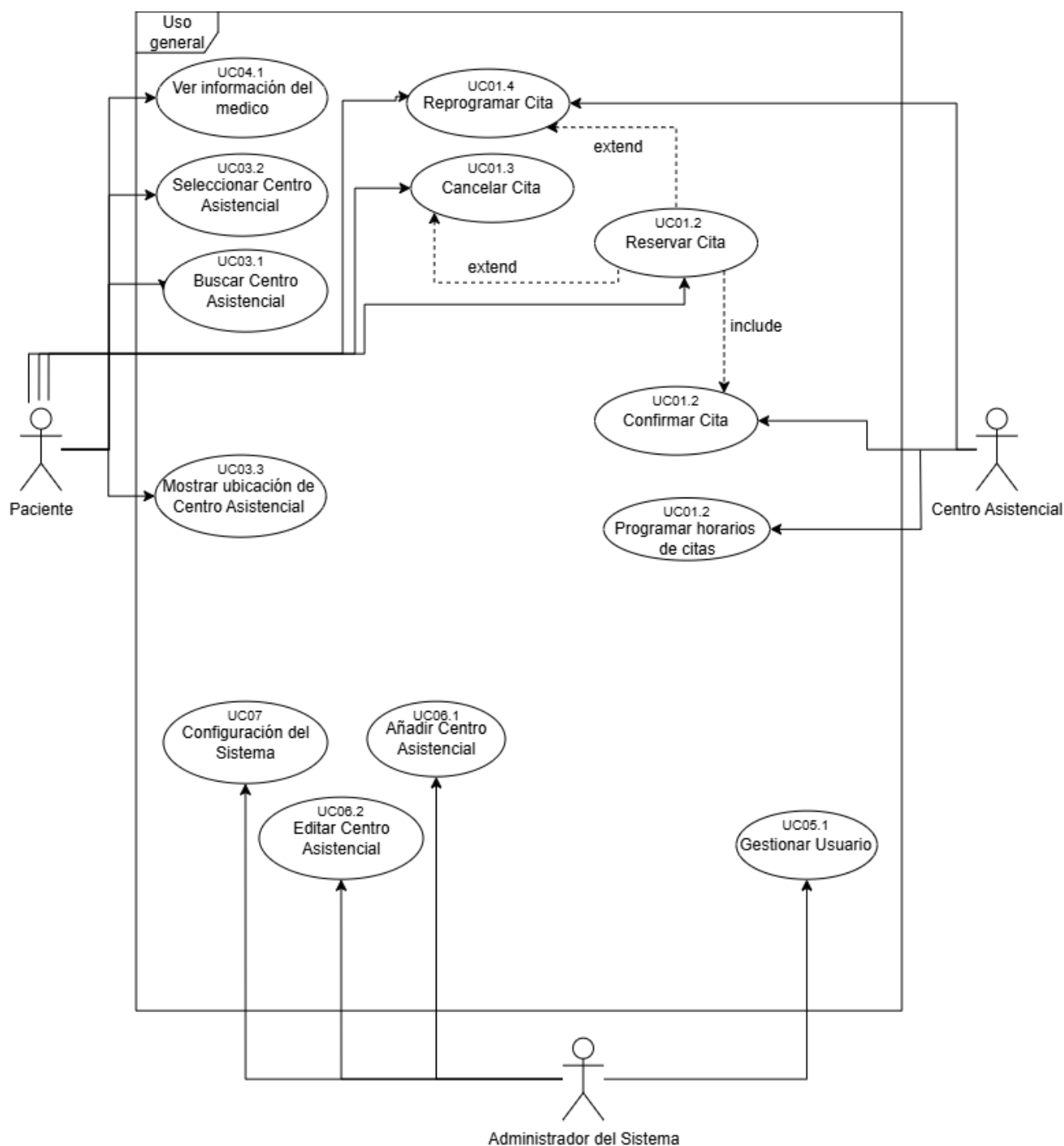
- El sistema debe proporcionar a los administradores reportes y estadísticas sobre la utilización de servicios, número de citas reservadas, canceladas y reprogramadas.
- El sistema debe permitir a los administradores gestionar la información de los usuarios, incluyendo la creación, modificación y eliminación de cuentas.
- El sistema debe permitir a los administradores gestionar y actualizar la información de los centros asistenciales, incluyendo especialidades y horarios de atención.

Los requisitos no funcionales son aquellos que no se encuentran relacionados directamente a los servicios ofrecidos, pero si pueden relacionarse con propiedades emergentes del sistema:

- Proteger la información de los usuarios
- Se debe tener una disponibilidad 24/7 con un tiempo entre fallas mínimo.
- Capacidad de adaptabilidad y escalabilidad, manteniendo lo antes mencionado.
- El sistema debe mantener su rendimiento óptimo y cumplir con los requisitos de seguridad y disponibilidad durante su escalabilidad.

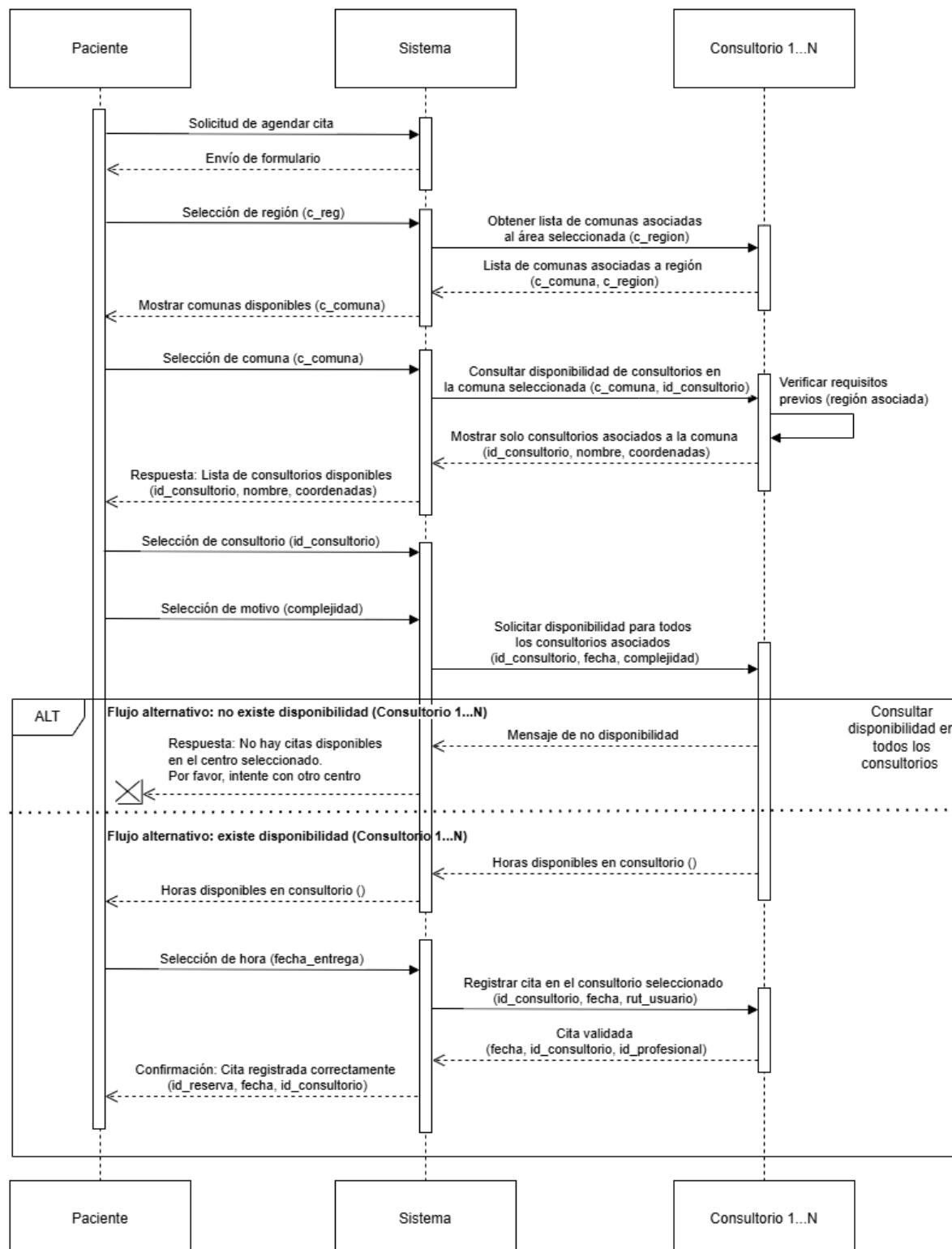
Los componentes web se encuentran en el repositorio GitHub solicitado:
https://github.com/adinamarca/salud_publica_digital/

3.2 DIAGRAMA DE CASO DE USO

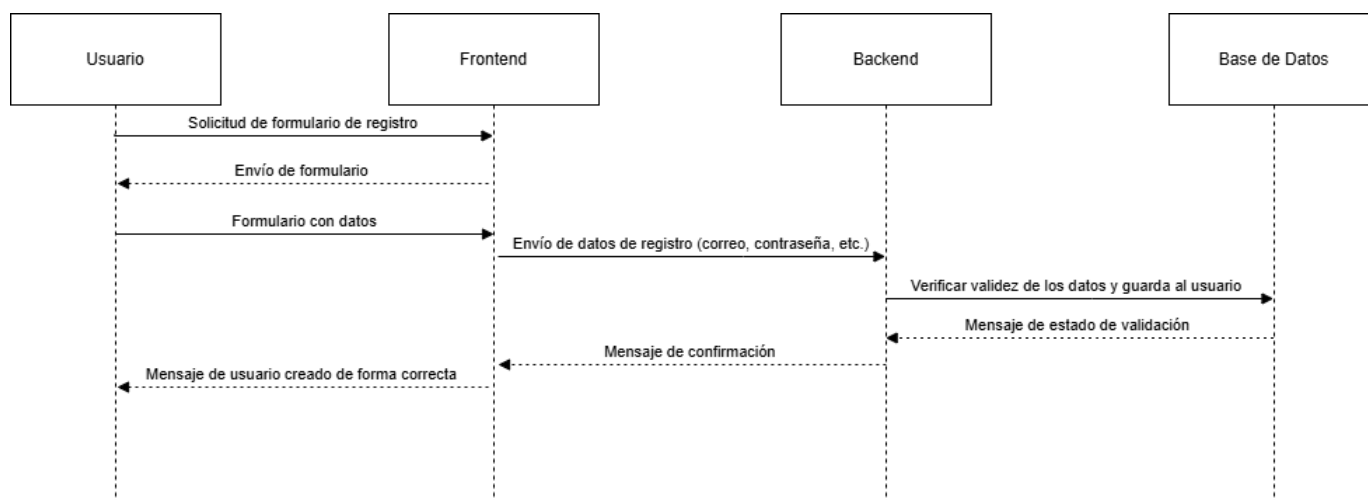


3.3 DIAGRAMAS DE SECUENCIA

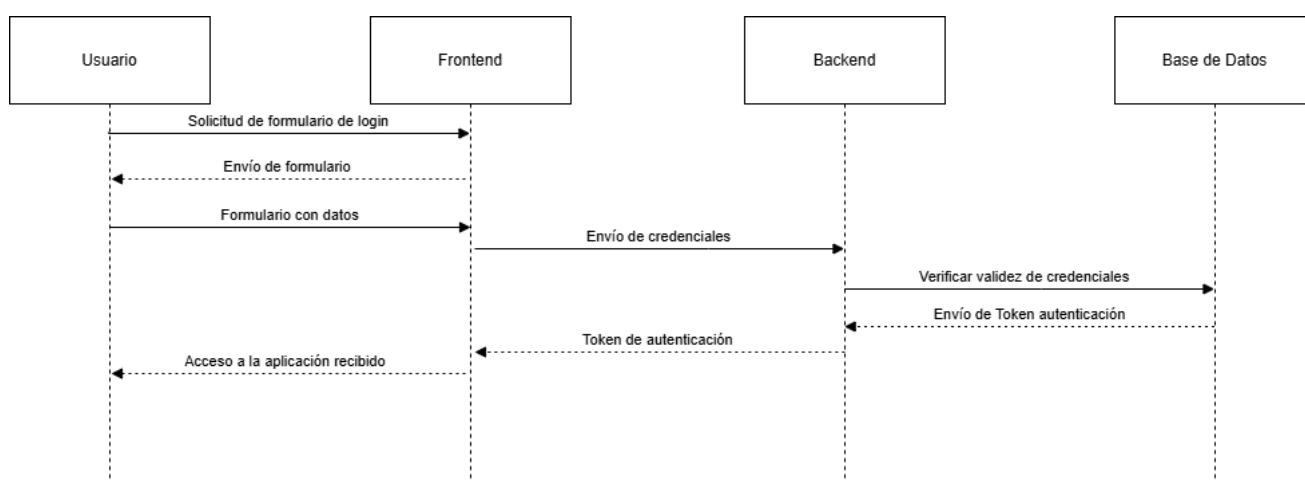
-Agendar una cita:



-Registro de pacientes:

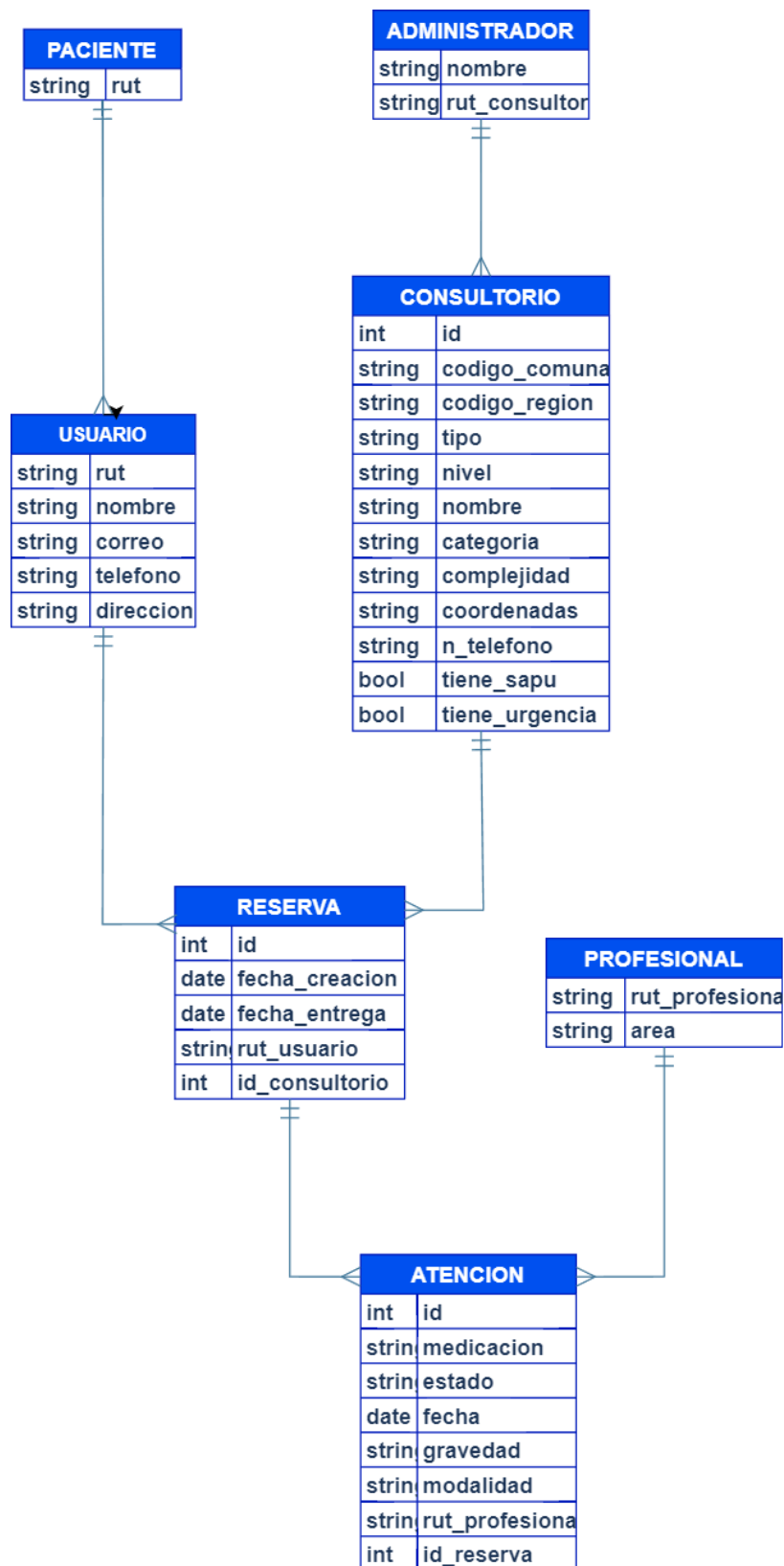


-Inicio de sesión:



4 COMPONENTES BD

4.1 MODELO RELACIONAL



5 BACKEND – API REST

5.1 INTRODUCCIÓN

El Backend de ConsultorioDigital ya se encuentra implementado haciendo uso de Django, un framework que facilita la creación de aplicaciones web de manera escalable. Esta API REST gestiona la lógica de negocio, autenticación y base de datos asociada al proyecto.

5.2 ENDPOINTS Y ARQUITECTURA

La API ofrece endpoints que permiten operaciones CRUD sobre las entidades ya definidas como usuarios, citas y centros asistenciales. Los endpoints facilitan el acceso a nuestros recursos por medio de métodos HTTP de uso estándar (POST, GET, DELETE Y PUT).

Pasando al aspecto de la implementación de los **puntos finales, métodos HTTP admitidos, parámetros de entrada, respuestas esperadas y códigos de estado** presentes en la solución desarrollada podemos encontrar:

5.2.1 Obtener Token de Acceso

- **Endpoint:** `/api/v1/token/`
- **Método:** POST
- **Descripción:** Devuelve un Token de acceso.
- **Parámetros:** `codigo_region`
 - `username:` (string) Nombre de usuario del usuario.
 - `password:` (string) Contraseña del usuario.
- **Respuesta:**
 - **Código de estado:** 200 OK
 - **Estructura de respuesta:**

```
[
  {
    "token": "string"
  },
  ...
]
```

- **Código de estado:** 400 Bad Request

```
{
  "error": "Credenciales inválidas"
},
...
]
```


5.2.2. Obtener Token de Refresco

- **Endpoint:** `/api/v1/token/refresh/`
- **Método:** POST
- **Descripción:** Devuelve un Token de refresco.
- **Parámetros:**
 - `refresh`: (string) Token de refresco.
- **Respuesta:**
 - **Código de estado:** 200 OK
 - **Estructura de respuesta:**

```
[  
  {  
    "access": "string"  
  },  
  ...  
]
```

5.2.3. Listar Regiones

- **Endpoint:** `/api/v1/region/`
- **Método:** GET
- **Descripción:** Devuelve una lista de consultorios para una comuna específica.
- **Parámetros:** Ninguno
- **Respuesta:**
 - **Código de estado:** 200 OK
 - **Estructura de respuesta:**

```
[  
  {  
    "c_reg": 1,  
    "nom_reg": "<Nombre de la Región>",  
    ...  
  },  
  ...  
]
```

5.2.4. Obtener Comunas por Región

- **Endpoint:** `/api/v1/comuna/<int:c_reg>/`
- **Método:** GET
- **Descripción:** Devuelve las comunas presentes en la región.
- **Parámetros:**
 - `c_reg`: (int) Código de la región.
- **Respuesta:**
 - **Código de estado:** 200 OK

- **Estructura de respuesta:**

```
{ [
  {
    "c_com": "Código Comuna",
    "nom_com": "Nombre de la Comuna"
  },
  ...
]
```

- **Código de estado:** 404 Not Found , Si no encuentran comunas.

2.5.5. Obtener Consultorios por Comuna

- **Endpoint:** /api/v1/consultorio/<str:c_com>/
- **Método:** GET
- **Parámetros:**
 - c_com: (int) Código de la comuna.
- **Respuesta:**
 - **Código de estado:** 201 Created si la reserva es exitosa.
 - **Estructura de respuesta:**

```
{
  {
    "c_com": "Código Comuna",
    "nombre": "Nombre del Consultorio",
    ...
  },
  ...
}
```

- **Código de estado:** 404 Not Found si no se encuentran consultorios.

2.5.6. Registro

- **Endpoint:** /registro/
- **Método:** POST
- **Parámetros:**
 - c_com: (int) Código de la comuna.
- **Respuesta:**
 - **Código de estado:** 200 Created Respuesta HTML o JSON.
 - **Código de estado:** 400 Bad Request Mensaje de error si el registro falla.

2.5.7. Autenticación

- **Endpoint:** /accounts/login/
- **Método:** GET y POST
- **Parámetros:**
 - Al hacer POST:
 - **username:** (string) Nombre de usuario.
 - **password:** (string) Contraseña.
- **Respuesta:**
 - **Código de estado:** 200 Created Redirección a la página deseada tras inicio de sesión.
 - **Código de estado:** 401 Unauthorized Mensaje de error si las credenciales son incorrectas

5.3. AUTENTICACIÓN Y AUTORIZACIÓN

Todos los endpoints que requieren autenticación deben incluir un token de sesión o un mecanismo de autenticación, esto es implementado tal que:

Autenticación

- **Autenticación:** Todos los endpoints que requieren autenticación deben recibir un.
 - **Proceso de Autenticación:**
 1. El usuario envía sus credenciales (usuario y contraseña) al endpoint de login.
 2. Si las credenciales son válidas, el servidor responde con un **token de acceso** (JWT).
 3. El cliente puede acceder a los apartados a los cuales está autorizado.
- **Expiración y Renovación del Token:** Define un tiempo de expiración para el token y utiliza un sistema de renovación para sesiones prolongadas, minimizando el riesgo de acceso no autorizado.
- **Verificación de Permisos:** Antes de procesar cada solicitud, el servidor verifica si el usuario tiene permisos para acceder al recurso solicitado. Esto asegura que un usuario solo pueda ver o modificar datos a los que está autorizado.
- **Ejemplo de Autenticación y Autorización**

1. Login:

- El cliente envía una solicitud POST /api/login con las credenciales en el cuerpo.
- El servidor responde con un token JWT.

2. Acceso a un Endpoint Protegido:

- El cliente realiza una solicitud GET /api/citas e incluye el token en la cabecera.
- El servidor verifica el token y los permisos del usuario antes de responder.

5.4. ERRORES

Podemos documentar los códigos/mensajes de error que se encuentran dentro de nuestra API REST de manera que se tienen mensajes propios del framework que dirigen a mensajes personalizados que son almacenados dentro de los propios formularios creados, tal que:

- **400 Bad Request:** La solicitud del cliente contiene datos inválidos/faltantes, esto indica la falta de datos en campos obligatorios o datos que no cumplen con validaciones.
 - **Ejemplos de mensajes:**
 - "Ingrese un RUT válido"
 - "Ingrese un correo válido"
 - "Este campo es obligatorio"
- **401 Unauthorized:** El cliente no está autenticado o el token de acceso es inválido, esto debido a credenciales invalidas. Este error se utiliza en endpoints de autorización.
 - **Ejemplos de mensajes:**
 - "Token de acceso no proporcionado"
 - "Token de acceso inválido"
 - "Token de acceso expirado" (error más frecuente)
 - "Autenticación fallida"
- **403 Forbidden:** El cliente no tiene los permisos necesarios para acceder al recurso solicitado, esto al incumplir con los permisos asociados al usuario autenticado, esto al intentar acceder a un recurso al que no posee permisos de acceso.

- Ejemplos de mensajes:
 - "No tiene permiso para acceder a este recurso"
 - "Acceso denegado"
- **404 Not Found:** El recurso solicitado no existe, esto al solicitar un recurso inexistente.
 - Ejemplos de mensajes:
 - "El recurso solicitado no fue encontrado"
 - "Consultorio no encontrado"
- **405 Method Not Allowed:** Se intentó acceder a un recurso con un método HTTP no permitido.
 - Ejemplos de mensajes:
 - "Error interno. Por favor, contacte al soporte."
- **409 Conflict:** Ocurre al tener conflicto de datos, puede ser útil para manejar los mismos.
 - Ejemplos de mensajes:
 - "Ya existe una reserva para el mismo paciente en esta fecha"
 - "El RUT ya está registrado en el sistema"
 - "Conflicto en los datos proporcionados"
- **500 Internal Server Error:** Ocurrió un error inesperado en el servidor, esto al existir un fallo en la base de datos o lógica implementada.
 - Ejemplos de mensajes:
 - "Ha ocurrido un error en el servidor. Inténtelo de nuevo más tarde."
 - "Error interno. Por favor, contacte al soporte."

Dentro del repositorio de GitHub es posible encontrar todos los formularios personalizados que fueron creados, estos funcionan en concurrencia con los mensajes predeterminados y pueden ser vistos en el siguiente link:

https://github.com/adinamarca/salud_publica_digital/blob/main/registro/forms.py

5.2. CONSIDERACIONES DE SEGURIDAD

Garantizar la seguridad de los datos manejados y la comunicación es una de las principales preocupaciones a futuro en el desarrollo de ConsultorioDigital, esto debido a las extensas exigencias legales que regulan el uso de información privilegiada de los pacientes a atender, esto genera una necesidad de consideraciones de carácter más avanzado.

Por tanto, el manejo de los datos privados del usuario debe ser seguro y solo ofrecido al personal de salud que atiende al paciente en cuestión. Esto es implementado de la siguiente manera:

1. **Autenticación y Autorización:**

- Implementa autorización basada en roles para limitar el acceso a datos sensibles.

2. **Control de Acceso:**

- Limita los accesos a personal autorizado y registra todas las actividades de los usuarios.

3. **Protección contra Ataques Comunes:**

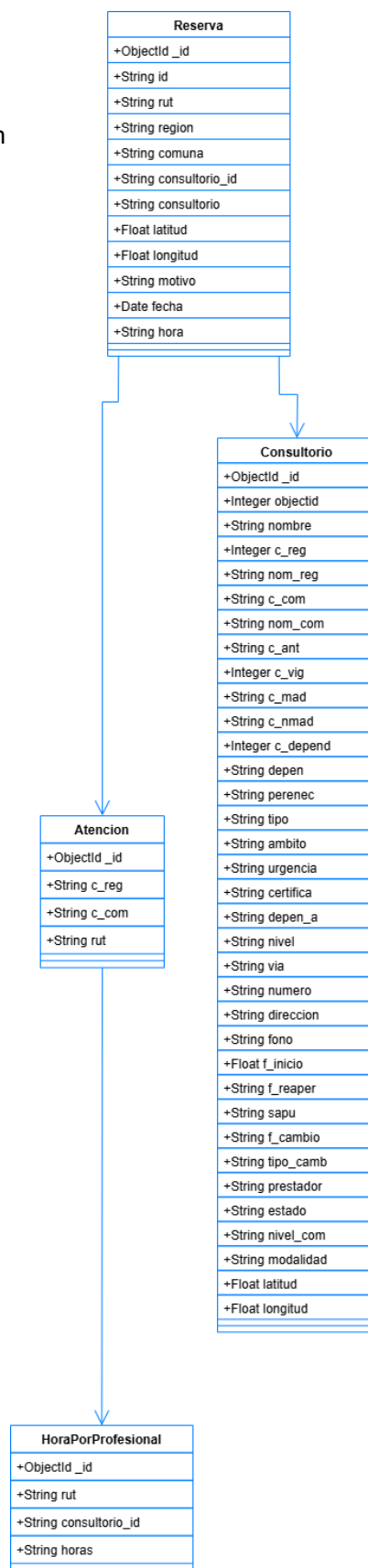
- Usa validación estricta y sanitización de datos de entrada.

4. **Gestión de Sesiones:**

- Define tiempos de expiración de tokens de autenticación y usa renovación de tokens para mitigar accesos prolongados.
- Implementa la revocación de tokens para usuarios que cambian credenciales o presentan actividades sospechosas.

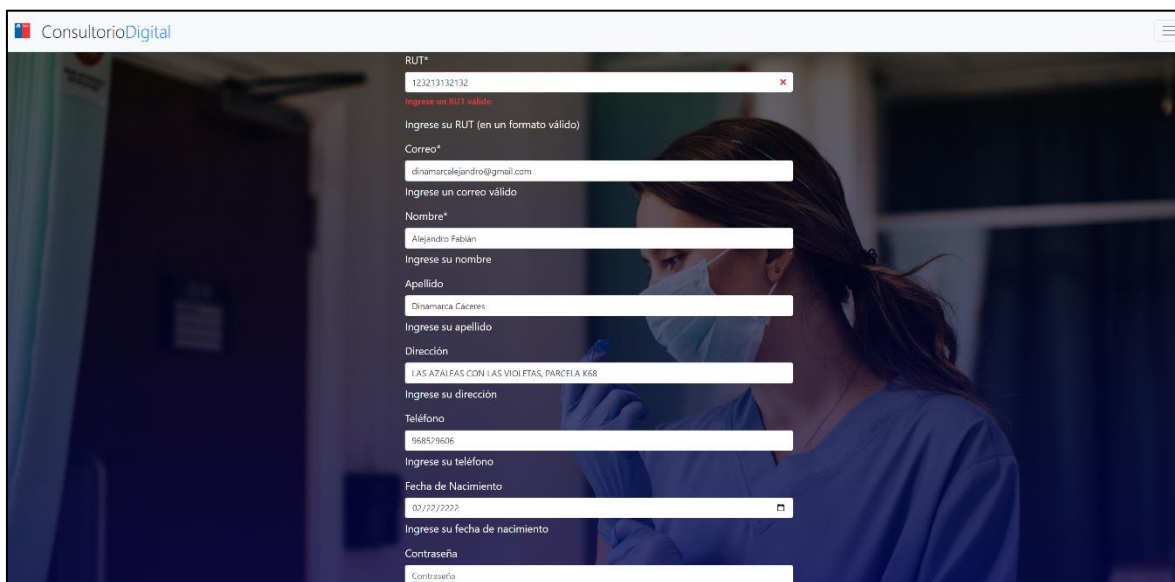
6 MODELO NoSQL

- Notar reducción de tablas en comparación con el modelo relacional NoSQL.



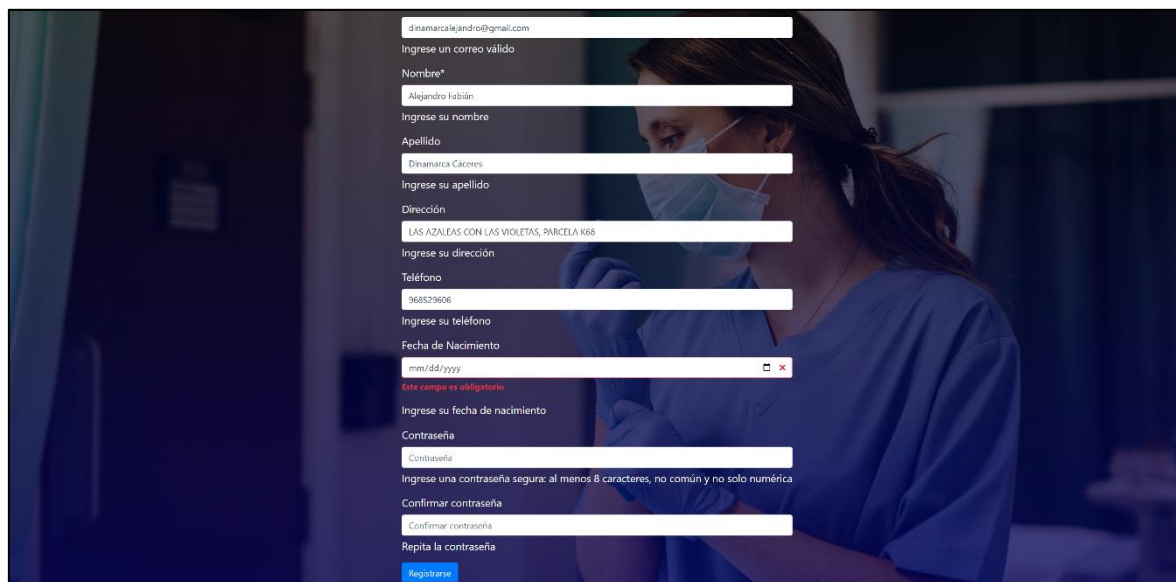
7 VERIFICACIÓN Y FUNCIONALIDADES ADICIONALES

Se han implementado restricciones y verificaciones adicionales para asegurar el correcto funcionamiento y mejorar la respuestas a fallas del sistema.



The screenshot shows the 'ConsultorioDigital' registration form. The 'RUT*' field contains '123213132132' and displays a red error message: 'Ingrese un RUT válido'. Below this, the form continues with fields for 'Correo*', 'Nombre*', 'Apellido', 'Dirección', 'Teléfono', 'Fecha de Nacimiento', and 'Contraseña'. The background image shows a healthcare worker in a clinical setting.

Figura 1. Restricción de Rut invalido.



The screenshot shows the 'ConsultorioDigital' registration form with the 'Fecha de Nacimiento' field highlighted. It contains the date 'mm/dd/yyyy' and a red error message: 'Este campo es obligatorio'. The form also includes fields for 'Correo*', 'Nombre*', 'Apellido', 'Dirección', 'Teléfono', 'Contraseña', and a 'Confirmar contraseña' field. A 'Registrar' button is at the bottom. The background image shows a healthcare worker in a clinical setting.

Figura 2. Restricción de campo obligatorio.

Un ejemplo de la implementación de estas verificaciones es:

```
def is_valid(self):
    valid = super().is_valid()

    if not valid:
        return False

    cleaned_data = self.clean()
    dni = cleaned_data.get('username')

    if not validate_chilean_dni(dni):
        self.add_error("username", "Ingrese un RUT válido")
        return False

    # If any is empty, return False
    for key, value in cleaned_data.items():
        if (value == "") or (value is None):
            self.add_error(key, "Este campo es obligatorio")
            return False

    return True
```

Figura 3. Fragmento de código usado para verificaciones.

Otro apartado que requiere especial atención es la limitación de acceso y diferenciación de permisos entre las distintas vistas implementadas (paciente / profesional).

Un ejemplo de estas limitaciones implementadas se encuentra en la imposibilidad de acceder a componentes propios de una vista desde fuera de la misma, un ejemplo:

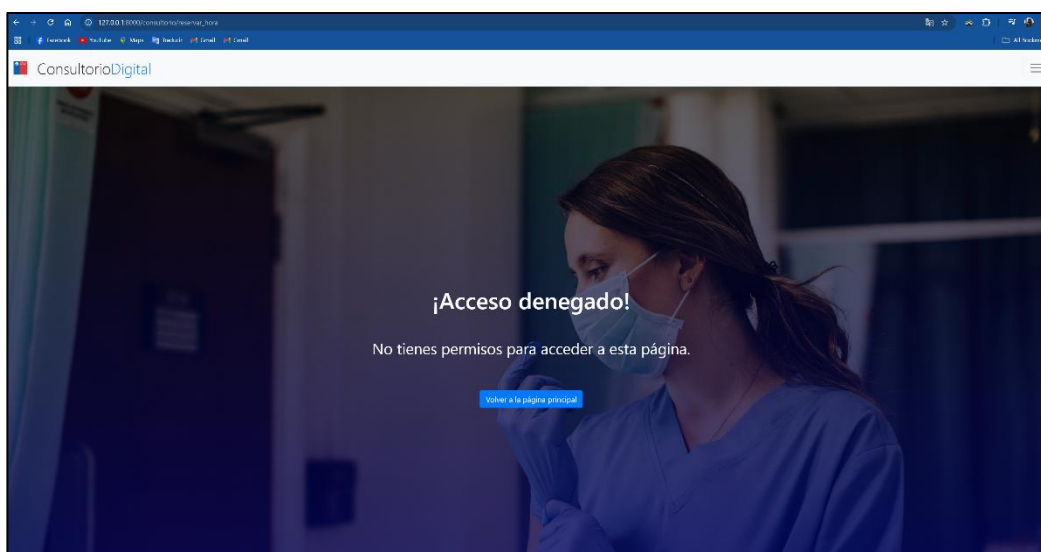


Figura 4. Intento de reservar cita como profesional.

Los formularios han sido mejorados para depender del tipo de usuario, en el caso del formulario para profesional, se adjuntan campos para determinar su área de especialidad, y otros datos de interés.

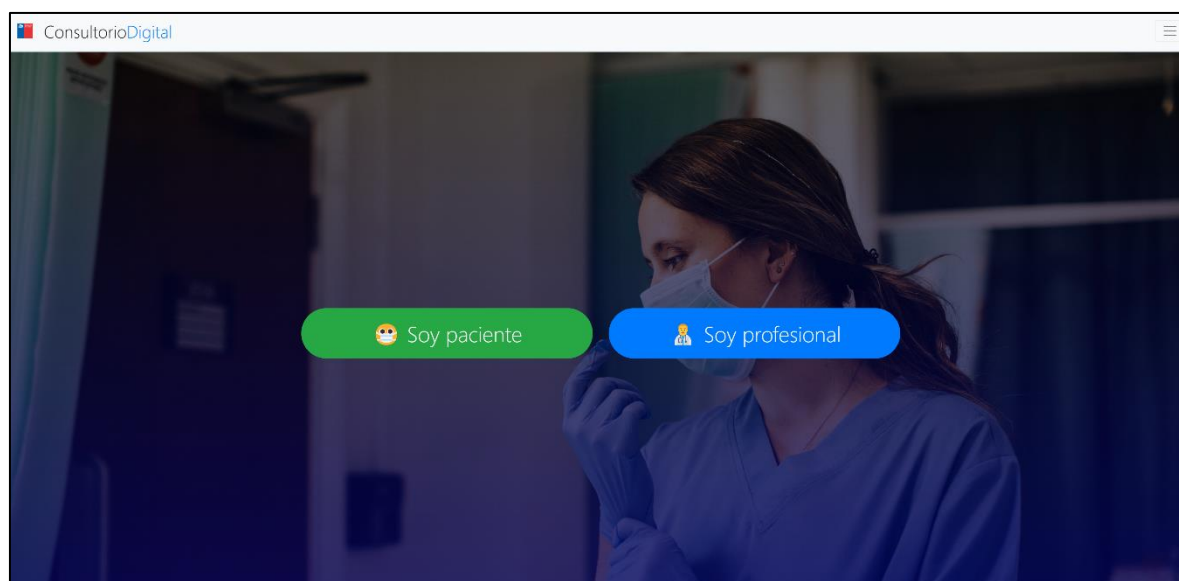


Figura 5. Selección de tipo de usuario.

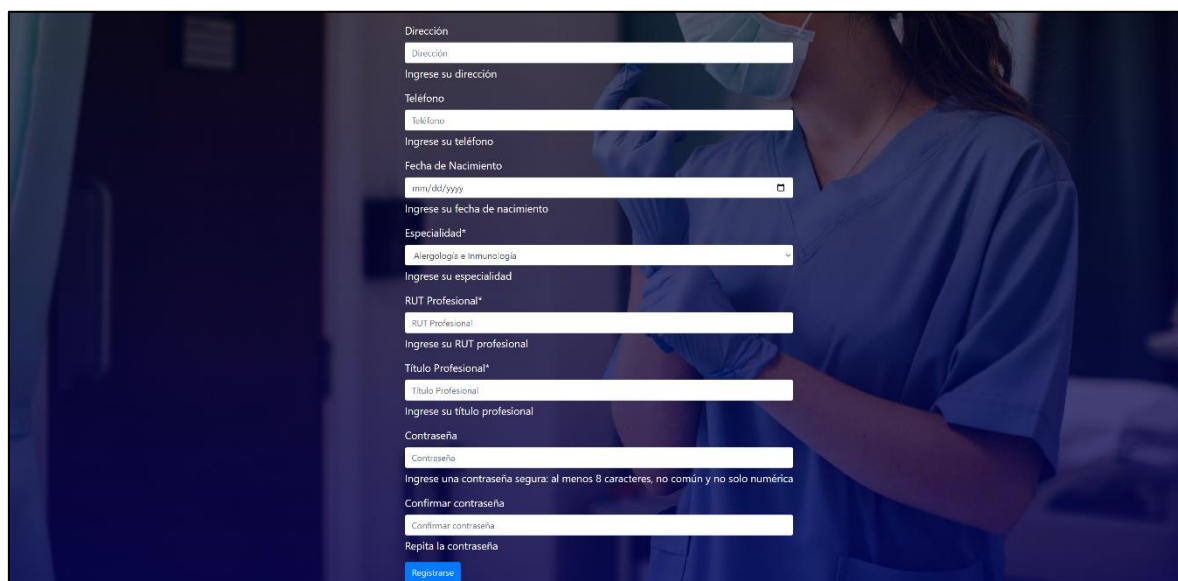


Figura 6. Formulario de registro para profesional.

El menú también dependerá del tipo de usuario. En el caso particular del profesional, tendrá las pestañas "Agregar hora" y "Configuración" para fijar su localización.

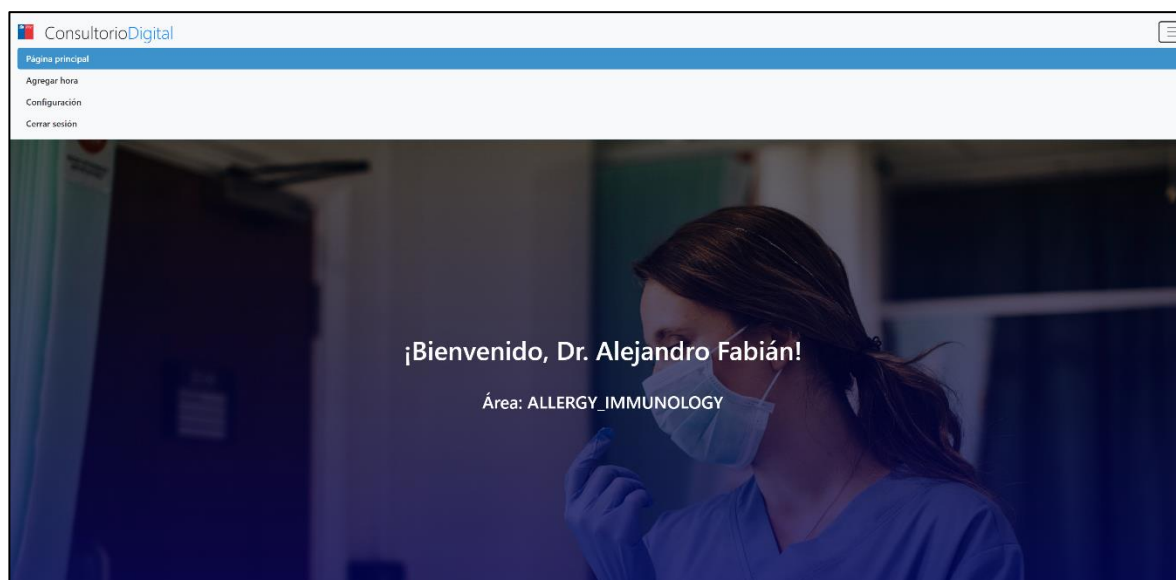


Figura 7. Inicio para profesional.

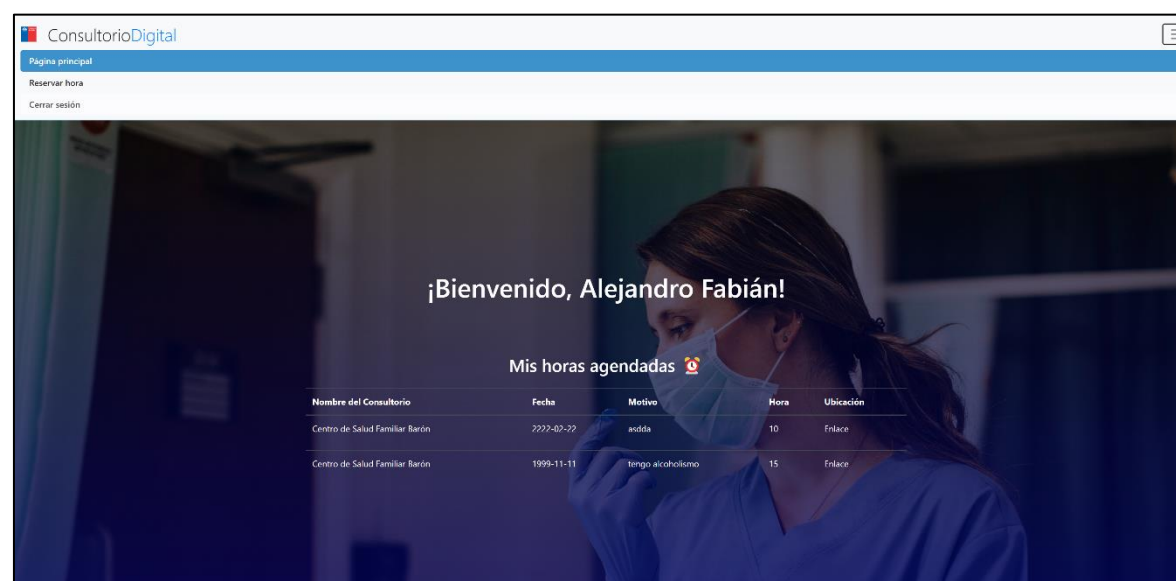


Figura 8. Inicio para paciente.

En el caso de la vista del profesional, si ingresa a "Agregar hora" sin haber configurado su localización, se le redirigirá automáticamente a "Configuración" para fijar su localización, ya que la vista de "Agregar hora" solo mostrará los consultorios que dependerán de la Región y comuna seleccionada por el profesional. Esta vista utiliza NOSQL con una operación "upsert", a modo que, un profesional no podrá tener más de dos localizaciones fijadas. De esta forma:

- Si el profesional fijará por primera vez su localización se insertará el dato.
- Si el profesional ya fijó su localización, y por ende ya existe el dato, este se actualizará.

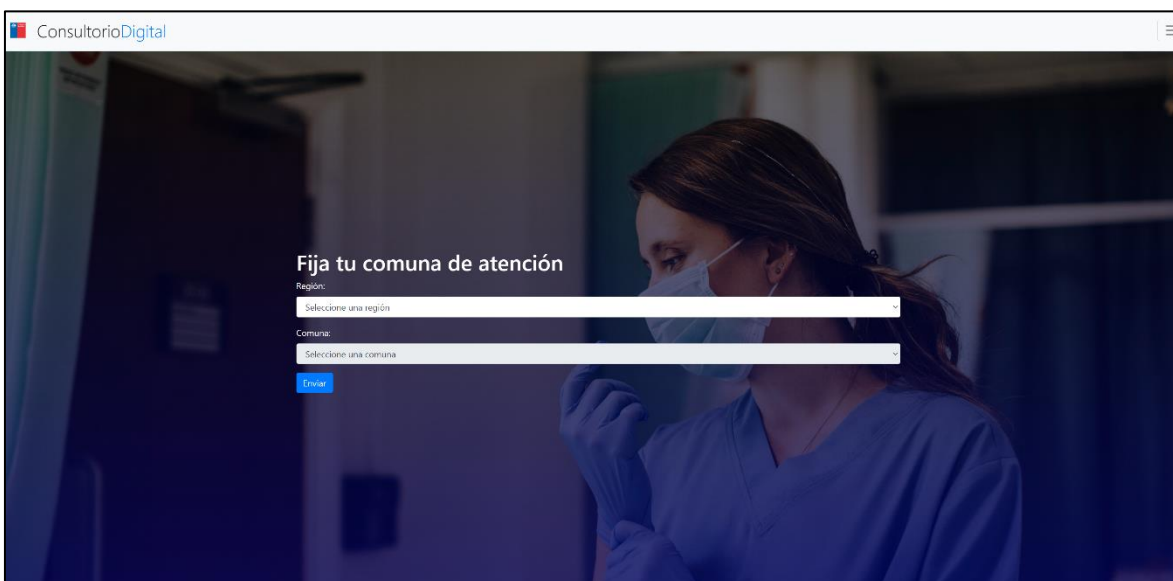


Figura 9. Selección de comuna para profesional.

Esta vista también utiliza NOSQL con una operación "upsert", a modo que, un profesional no podrá tener más de dos intervalos de tiempo fijados:

- Si el profesional fijará por primera vez su horario de atención, se insertará el dato.
- Si el profesional ya fijó su horario de atención, y por ende ya existe el dato, este se actualizará.

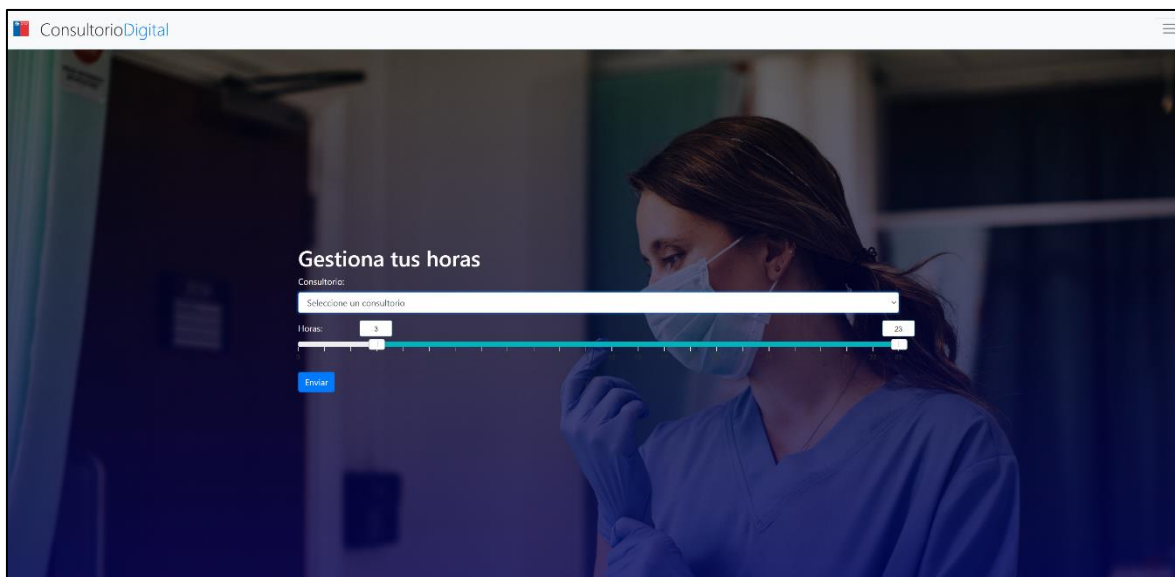


Figura 10. Selección de horas para profesional.

Hay que destacar que, si ningún profesional ha ingresado horas disponibles en un consultorio, y un paciente elige un consultorio sin profesionales disponibles, no podrá hacerlo hasta que existan horas en ese centro asistencial.

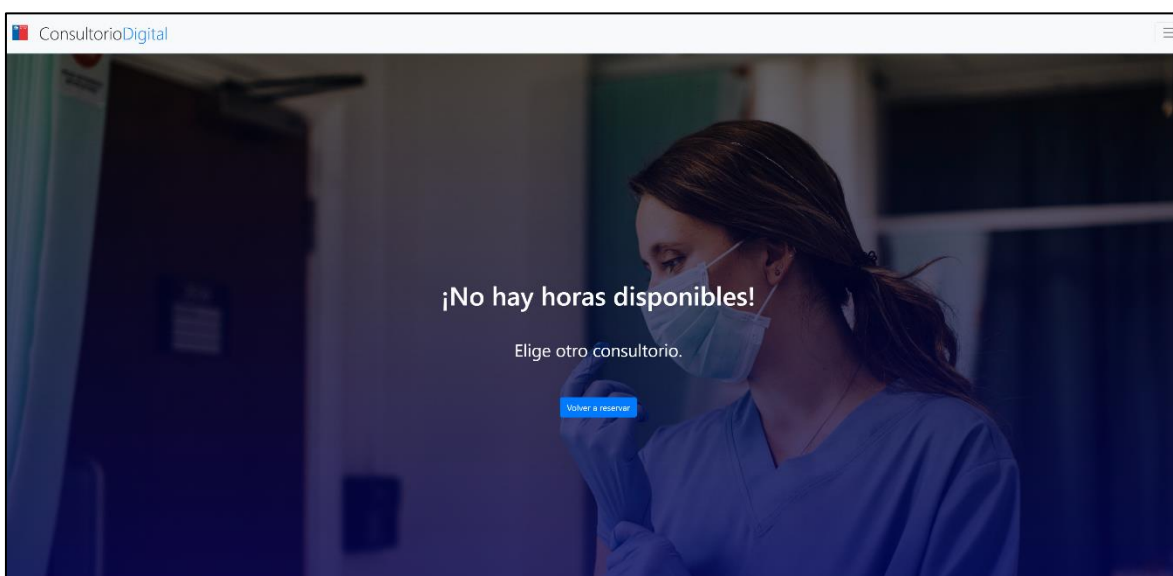


Figura 11. Intento de agendar cita sin disponibilidad.

8 ESTRUCTURA JSON FINAL (MONGODB)

```
{
  "_id": {
    "$oid": "675bc06844c8c6020b122872"
  },
  "id": "049e7b737403a66ce302403fa192761f",
  "rut": "74396232",
  "region": "5",
  "comuna": "05101",
  "consultorio_id": "270",
  "consultorio": "Centro de Salud Familiar Barón",
  "latitud": -33.039003,
  "longitud": -71.600794,
  "motivo": "asdda",
  "fecha": "2222-02-22",
  "hora": "10"
}
```

Figura 12. Estructura de una reserva.

```
{
  "_id": {
    "$oid": "675baf7b3b2b0cdb0786dcf4"
  },
  "rut": "74396232",
  "consultorio_id": "1914",
  "horas": "4,20"
}
```

Figura 13. Horas por profesionales.

```
{
  "_id": {
    "$oid": "675b7633919a10c2efd65b0a"
  },
  "objectid": 2559,
  "nombre": "SAPU DR. AGUSTIN CRUZ MELO",
  "c_reg": 13.0,
  "nom_reg": "Región Metropolitana de Santiago",
  "c_com": "13108",
  "nom_com": "Independencia",
  "c_ant": "201063",
  "c_vig": 201063.0,
  "c_mad": "No Aplica",
  "c_nmad": "No Aplica",
  "c_depend": 9.0,
  "depen": "Servicio de Salud Metropolitano Norte",
  "perenec": "Perteneiente",
  "tipo": "Servicio de Atención Primaria de Urgencia (SAPU)",
  "ambito": "Establecimiento de Salud",
  "urgencia": "SI",
  "certifica": "NO",
  "depen_a": "Municipal",
  "nivel": "Primario",
  "via": "Calle",
  "numero": "1269",
  "direccion": "Domingo Santa María",
  "fono": "Pendiente",
  "f_inicio": 0.0,
  "f_reaper": "No Aplica",
  "sapu": "No Aplica",
  "f_cambio": "No Aplica",
  "tipo_camb": "Sin cambio",
  "prestador": "Público",
  "estado": "Vigente en operación",
  "nivel_com": "Baja Complejidad",
  "modalidad": "Atención Abierta-Ambulatoria",
  "latitud": -33.415499,
  "longitud": -70.657603
}
```

Figura 14. Consultorios en NoSQL.