



TECNOLÓGICO
NACIONAL DE MÉXICO

TecNM



TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Mexicali

Ing. Sistemas Computacionales

Desarrollo de Aplicaciones Web

Docente: Valenzuela Bogarin Jose Ramon

“Proyecto Mind Care – Sprint 2”

Integrantes:

Rodriguez Herrera Maria Fernanda – 21490574

Hernández López Jose Carlos – 22150084

Favila Sánchez Angel Jair – 21490074

Mexicali, B.C.

24-Oct-2025

Índice

Problemática.....	2
Objetivo del proyecto.....	2
Historias de usuario	2
Módulos funcionales y no funcionales	3
Tipo de arquitectura	4
Wireframes	4
Plan de Sprints.....	7
Criterios aceptables.....	8
Sprint 1: 29 sep - 10 oct.....	9
Sprint 2: 13 oct - 24 oct.....	17

Problemática

La salud mental es un tema de suma importancia, ya que afecta en el día a día de las personas, tanto en sus relaciones sociales como personales.

Es por ello que Mind Care busca que las personas puedan agendar sesiones con un psicólogo, para manejar sus citas desde la comodidad de su casa.

Objetivo del proyecto

Crear una aplicación web en la cual los usuarios puedan agendar citas para sesiones psicológicas, enviándoles un recordatorio 24 horas antes a su cita para confirmarla o cancelarla.

Historias de usuario

Usuario

- Como paciente nuevo, quiero poder registrarme para tener una cuenta y posteriormente iniciar sesión
- Como paciente, quiero ver los días y horas disponibles para agendar mi cita.
- Como paciente, quiero agendar una cita seleccionando fecha y hora.
- Como paciente, quiero visualizar mis citas agendadas.
- Como paciente, quiero que se me notifique 24 horas un recordatorio de mi cita por medio de email.
- Como paciente, quiero poder confirmar y cancelar mis citas por medio del recordatorio.

Psicólogo

- Como psicólogo nuevo, quiero poder registrarme para tener una cuenta y posteriormente iniciar sesión
- Como psicólogo, quiero visualizar las citas agendadas del día.
- Como psicólogo, quiero crear notas para tener un historial de mis pacientes.
- Como psicólogo, quiero ver el historial de mis pacientes.
- Como psicólogo, quiero marcar días inhábiles.

Módulos funcionales y no funcionales

• Funcionales - Pacientes:

- **RF01:** Registro
- **RF02:** Inicio de sesión
- **RF03:** Autenticación
- **RF04:** Agendar citas (Una por día)
- **RF05:** Aceptar citas (por correo)
- **RF06:** Cancelar citas (por correo)
- **RF07:** Recordatorio de cita 24 horas antes (por correo)
- **RF08:** Historial de citas
- **RF09:** Calendario con disponibilidad de citas (día y hora)
- **RF10:** Correo notificando la cancelación de una cita, por día inhábil del psicólogo

• Funcionales - Psicólogo:

- **RF01:** Registro de psicólogo
- **RF02:** Inicio de sesión
- **RF03:** Autenticación
- **RF04:** Visualización de citas (Calendario)
- **RF05:** Historial de citas
- **RF06:** Creación de notas para el historial del paciente
- **RF07:** Modificar notas
- **RF08:** Crear de días inhábiles, y cancelación de citas en ese mismo día.
- **RF09:** Ver expediente de mis pacientes.

• No funcionales:

- **RNF01:** Interfaz intuitiva, accesible y rápida de entender
- **RNF02:** Privacidad de acceso a notas del psicólogo
- **RNF03:** Restricciones de horario para sistema de recordatorios

Tipo de arquitectura

Monolítica – Multicapa

1-. Monolítica

- **¿Qué es?** Arquitectura donde toda la aplicación se desarrolla, construye y despliega como una sola unidad, incluyendo la interfaz de usuario, la lógica de negocio y el acceso a datos, todo en un solo bloque de ejecución.

- **Responsabilidad:** Centraliza toda la funcionalidad en un solo sistema, acoplando las capas.

- **Ejemplo proyecto:** Dentro de la aplicación se harían todas las funcionalidades (inicio de sesión, envío de correos, etc.), a la vez que la base de datos se compartiría por todos los módulos.

2-. Multicapa:

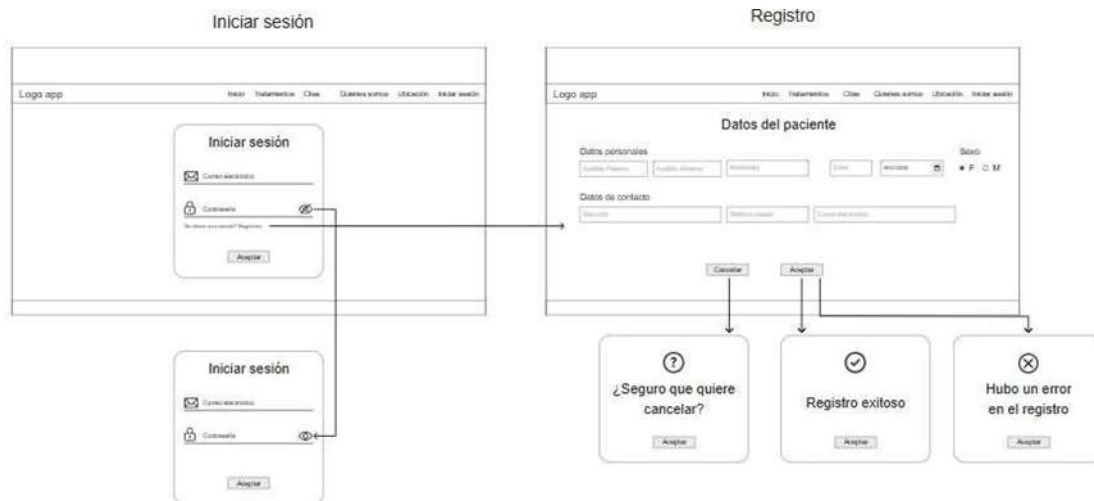
- **¿Qué es?** Divide la aplicación en capas lógicas independientes, teniendo cada una su responsabilidad específica.

- **Responsabilidad:** Separar las responsabilidades para que cada capa se centre en una tarea específica, mejorando la mantenibilidad, escalabilidad y reutilización.

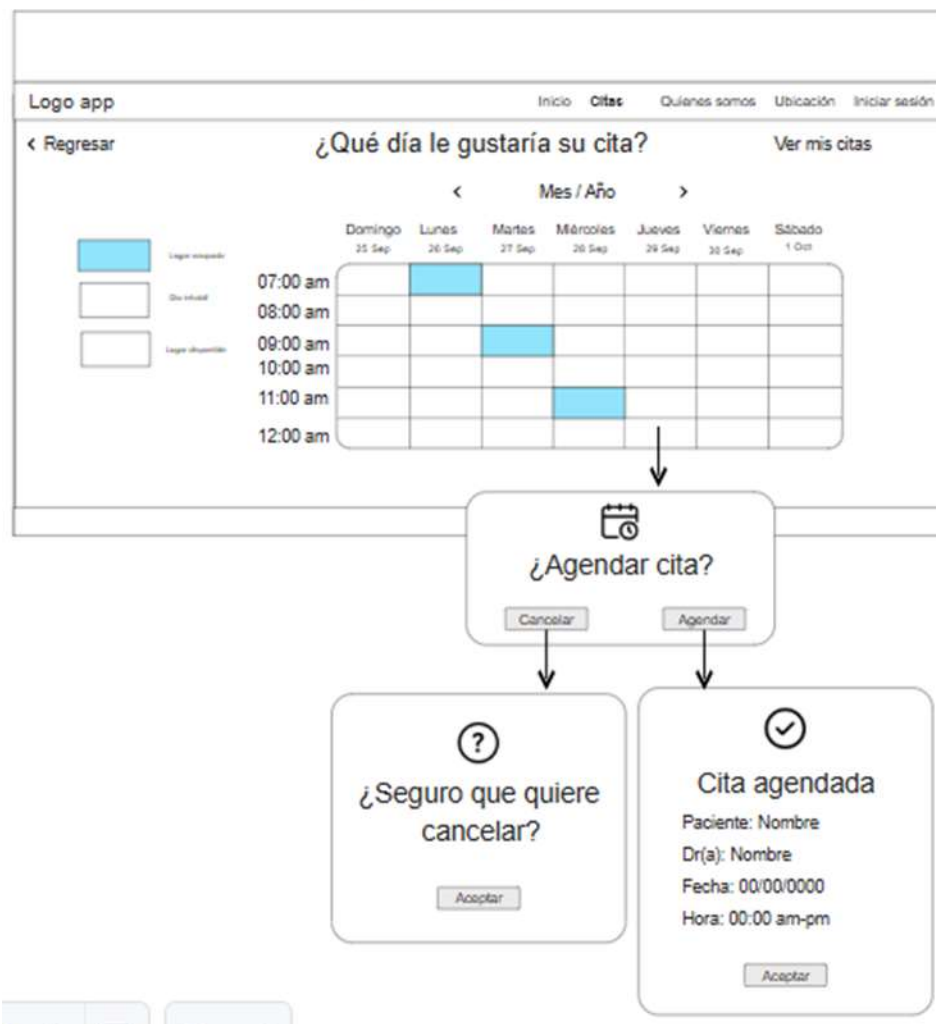
- **Ejemplo proyecto:** Separar la aplicación en capas de presentación, negocio y datos; para el proceso del manejo de sesiones.

Wireframes





Agendar cita - Paciente



Mis citas - Paciente

Logo app

Inicio Agendar Quienes somos Ubicación Iniciar sesión

Ver mis citas

ID Cita	Nombre	Fecha	Hora	Status
1	Jose Perez	2/2/2025	10:30	aceptada
2	Jose Perez	2/2/2025	10:30	cancelada

Regresar

Visualizar citas - Psicólogo

Logo app

Inicio Citas Quienes somos Ubicación Iniciar sesión

Regresar

Historial de citas

Pacientes a atender

Crear dia inahabil

Citas

< Mes / Año >

Domingo 25 Sep

Lunes 26 Sep

Martes 27 Sep

Miércoles 28 Sep

Jueves 29 Sep

Viernes 30 Sep

Sábado 1 Oct

07:00 am						
08:00 am						
09:00 am						
10:00 am						
11:00 am						
12:00 am						

"Recepcionista virtual"

Logo app

ID Cita	Nombre	HoraInicio	HoraFin
1	Jose Perez	12:00 am	1:00 am
2	Jose Fuentes	1:00 pm	2:00 pm

Plan de Sprints

* Sprint 1: 29 sep → 10 oct

Entregables:

- Menú de navegación (Inicio, Citas, Ubicación, Quienes somos, Iniciar sesión).
- Páginas estáticas: Inicio, Quienes somos, Ubicación, Iniciar sesión y Registro.
- Diseño de la base de datos.

* Sprint 2: 13 oct → 24 oct

Entregables:

- Creación de la base de datos.
- Registro e inicio de sesión funcionales.

* Sprint 3: 27 oct → 7 nov

Entregables:

- Vista de Calendario para el usuario.
- Función de agendar cita (Sin correo de recordatorio).
- Estados de cita: Pendiente, Aceptada (Si es el mismo día).
- Historial de citas: Solamente las aceptadas, sin notas por parte del psicólogo.

* Sprint 4: 10 nov → 21 nov

Entregables:

- Vista del calendario para el psicólogo (Aceptadas el mismo día, pendientes).
- Crear días inhábiles (Cancela las citas existentes de ese día).
- Historial de citas aceptadas el mismo día o canceladas por día inhábil.

* Sprint 5: 24 nov → 5 dic

Entregables:

- Configuración de servicio de correo en Flask.

- Envío automático para confirmaciones/cancelaciones
- Envío de correo de cancelación de cita por día inhábil del psicólogo.
- Visualización de expediente de pacientes.
- Cancelación y confirmación desde correo (Actualiza la BD y el calendario)
- Tabla con los pacientes a atender
- Deployment

Criterios aceptables

- Permitir a los pacientes y psicólogo registrarse para tener una cuenta y, posteriormente iniciar sesión con su cuenta para utilizar la aplicación.
- Permitir a los pacientes crear y visualizar sus citas.
- Implementar un sistema de recordatorios por email de 24 horas antes para las citas.
- Permitir al psicólogo cancelar citas y marcar días inhábiles.
- Permitir al psicólogo crear y modificar notas para el historial de sus pacientes.
- Permitir a los pacientes y psicólogo cerrar la sesión de su cuenta.

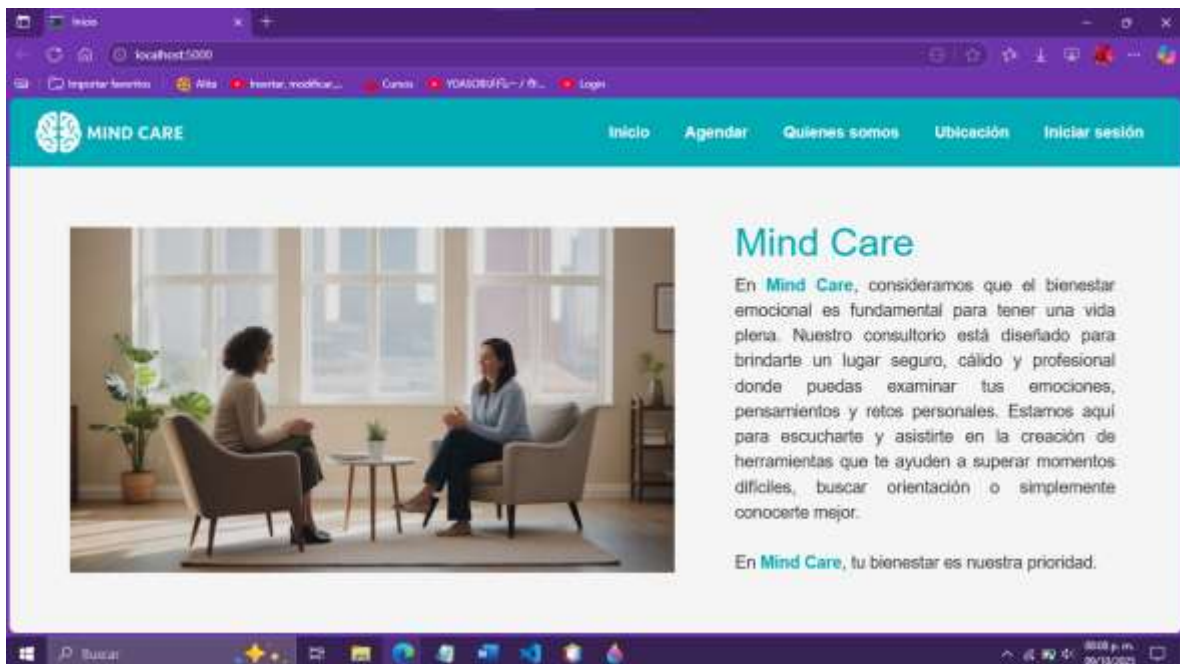
Sprint 1: 29 sep - 10 oct

Entregables:

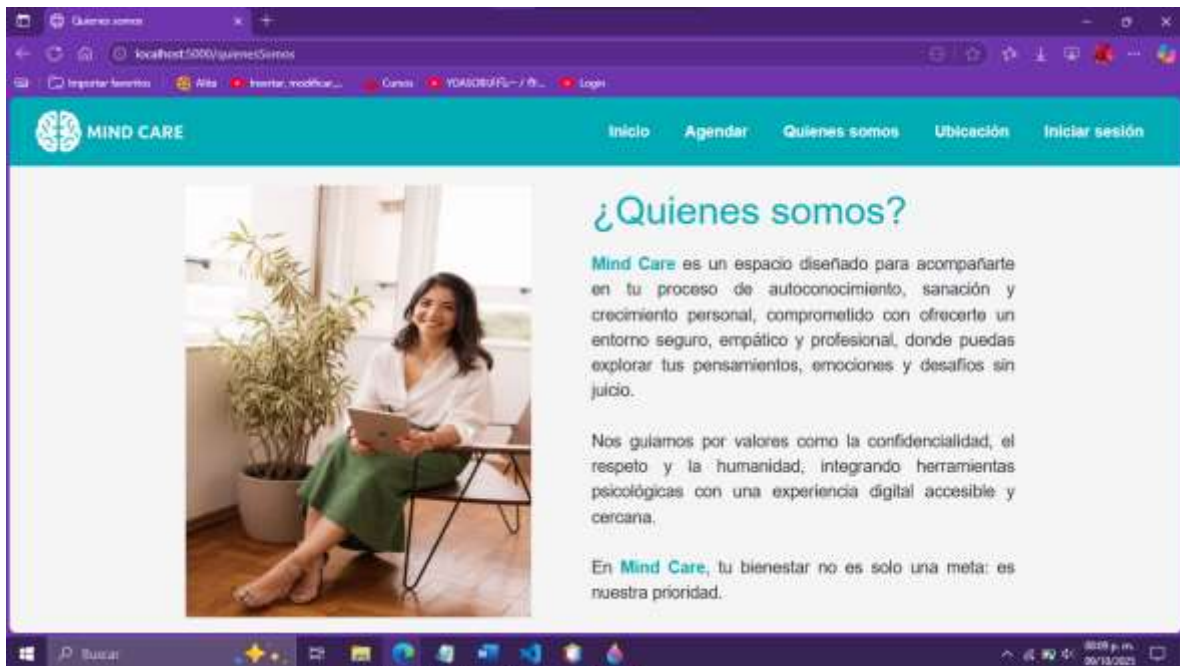
- Menú de navegación (Inicio, Citas, Ubicación, Quienes somos, Iniciar sesión).



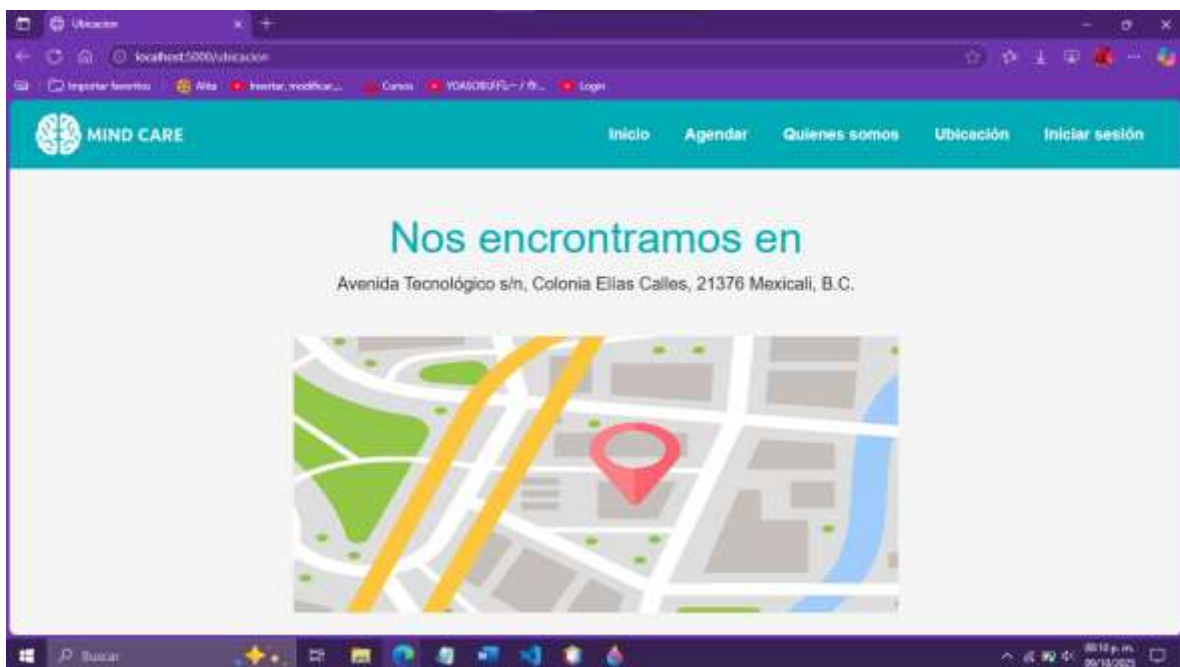
- Páginas estáticas: Inicio



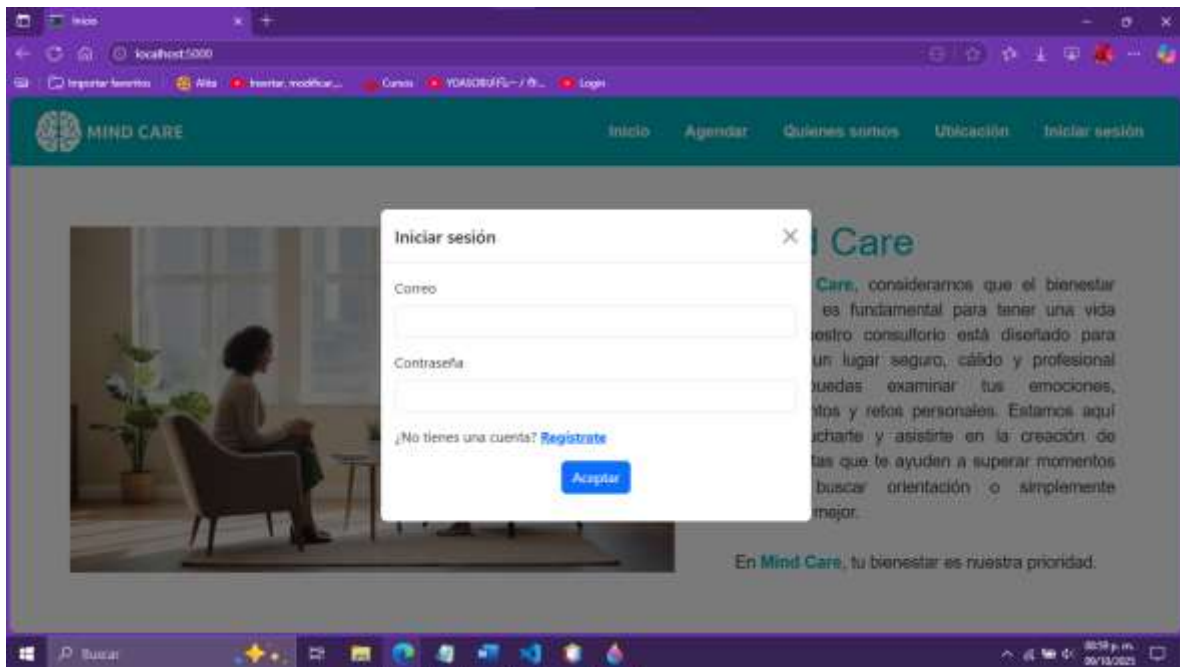
- Páginas estáticas: Quienes somos



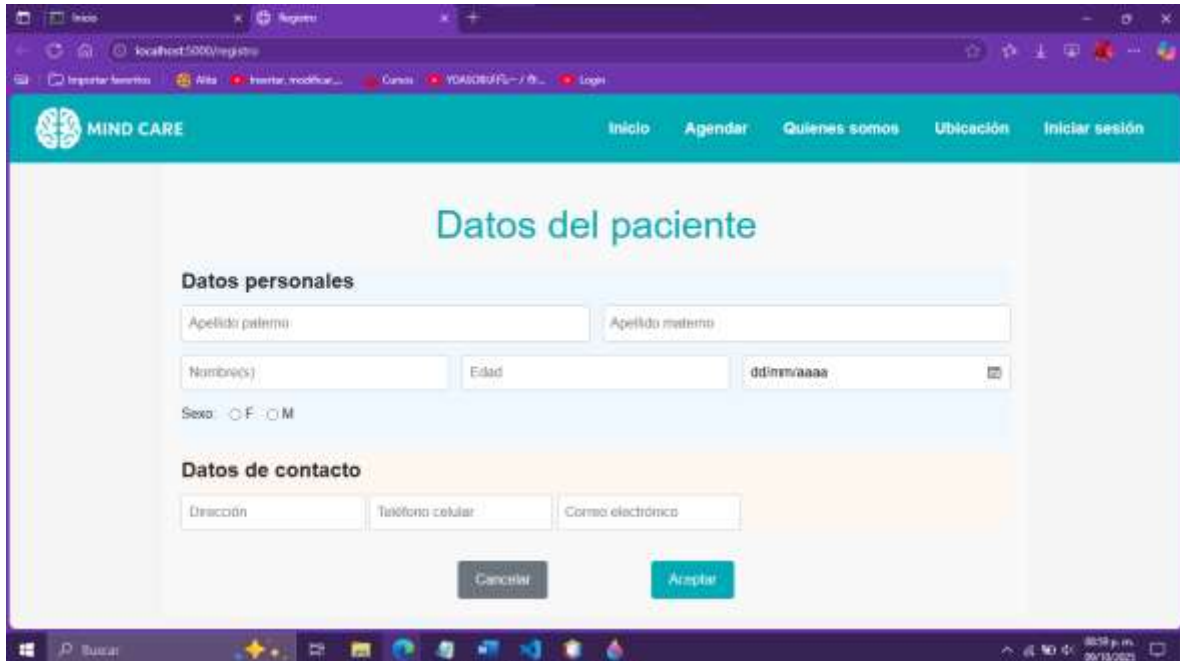
- Páginas estáticas: Ubicación



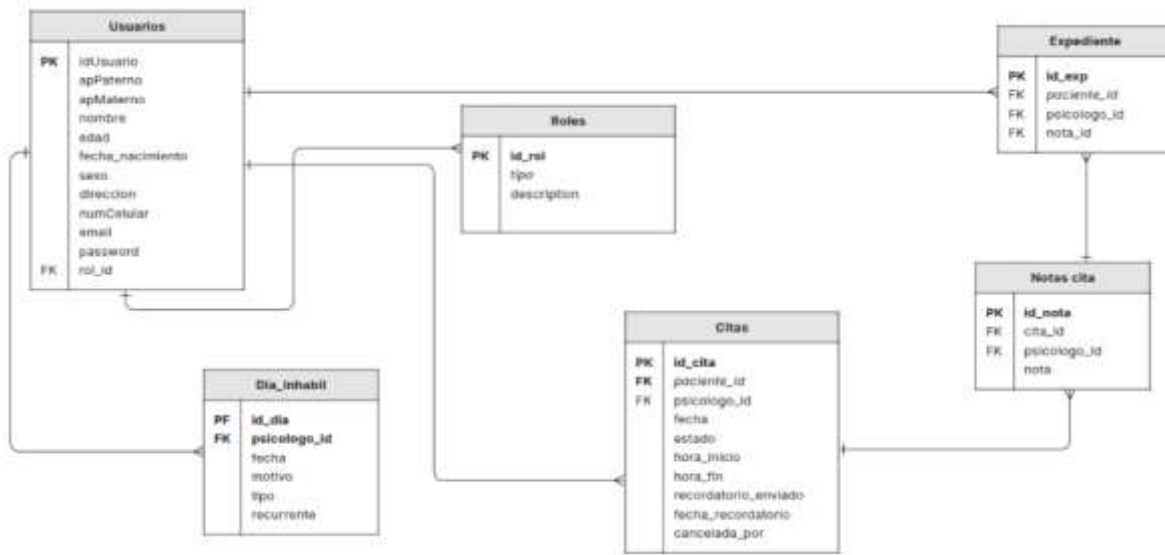
- Páginas estáticas: Iniciar sesión



- Páginas estáticas: Registro

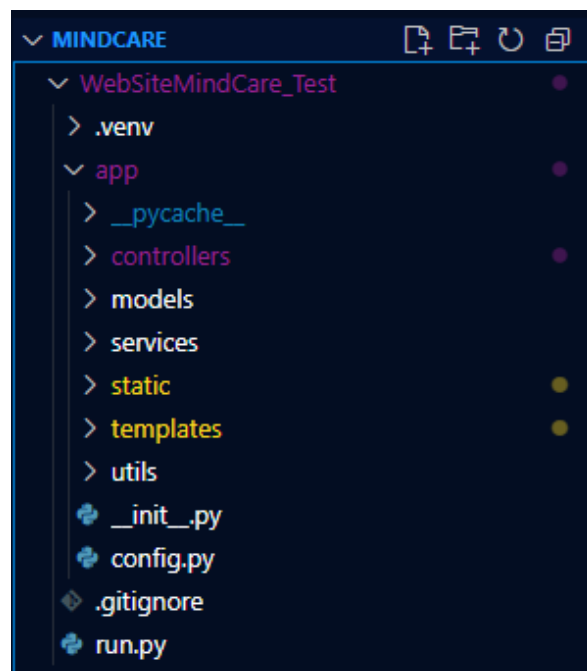


- Diseño de la base de datos.



Manejo de arquitectura Monolítica-Multicapa

Como se nos pidió trabajar nuestro proyecto en base a las arquitecturas monolítica y multicapa, se dividió el proyecto en las siguientes carpetas: ***controllers***, ***models***, ***services***, ***static***, ***templates*** y ***utils***, en las cuales de momento se han trabajado con ***controllers***, ***static*** y ***templates***, ya que se tienen los archivos ***html***, ***css*** e ***imágenes*** utilizadas en la aplicación, así como la definición de rutas para la navegación de la misma:



En nuestro archivo **main_routes.py** definimos las rutas para la navegación de nuestra aplicación, las cuales dirigen al usuario a las páginas de inicio, quienes somos, etc. La herramienta de **Blueprint** sirve para organizar la aplicación al dividirla en componentes más pequeños y manejables:

```
from flask import Blueprint, render_template

main_bp = Blueprint('main_bp', __name__)

#Rutas para archivos principales
@main_bp.route('/')
def inicio():
    return render_template('public/inicio.html')

@main_bp.route('/agendar')
def citas():
    return render_template('public/agendar.html')

@main_bp.route('/quienesSomos')
def quienesSomos():
    return render_template('public/quienesSomos.html')

@main_bp.route('/ubicacion')
def ubicacion():
    return render_template('public/ubicacion.html')

@main_bp.route('/login')
def login():
    return render_template('public/login.html')

@main_bp.route('/registro')
def registro():
    return render_template('public/registro.html')
```

En nuestra carpeta de **templates** (utilizada y reconocida por **Flask** para renderizar los archivos html) **/public** creamos un archivo llamado **menu.html**, el cual contiene la estructura y links de navegación para la aplicación. El propósito de esta es reutilizar el archivo mandándolo a llamar en las demás páginas para ahorrar código y aprovechar el motor de plantilla de **Flask** que es **Jinja2**:


```

O menu.html M X
WebSiteMindCare_Test > app > templates > public > O menu.html > ...
1 <nav class="navbar navbar-expand-lg bg-body-tertiary">
2   <div class="container-fluid">
3     <div class="logo">
4       <a href="{{ url_for('main_bp.inicio') }}">
5         
6       </a>
7     </div>
8     <div class="collapse navbar-collapse" id="navbarNav">
9       <ul class="navbar-nav">
10        <li class="nav-item">
11          <a class="nav-link" href="{{ url_for('main_bp.inicio') }}">Inicio</a>
12        </li>
13        <li class="nav-item">
14          <a class="nav-link" data-bs-toggle="modal" data-bs-target="#loginModal">Agendar</a>
15        </li>
16        <li class="nav-item">
17          <a class="nav-link" href="{{ url_for('main_bp.quienesSomos') }}">Quienes somos</a>
18        </li>
19        <li class="nav-item">
20          <a class="nav-link" href="{{ url_for('main_bp.ubicacion') }}">Ubicación</a>
21        </li>
22        <li class="nav-item">
23          <a class="nav-link" data-bs-toggle="modal" data-bs-target="#loginModal">Iniciar sesión</a>
24        </li>
25      </ul>
26    </div>
27  </div>
28 </nav>

```

De igual manera, en nuestra carpeta **templates/public** creamos un archivo llamado **base.html**, el cual contiene los links a **Bootstrap** y al **style.css** para facilitar los diseños, así como incluir al archivo **menu.html** para reutilizarlo, y en este mismo incluye **login_modal.html** que contiene la estructura de nuestro Pop up del login. Se utilizó **include** ya que sirve para reutilizar fragmentos de código HTML, como el menú, mediante el motor de plantillas **Jinja2** (como se mencionó anteriormente):

```

O base.html X
WebSiteMindCare_Test > app > templates > O base.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title> {% block title %} Pagina base {% endblock %} </title>
7   <!-- Bootstrap -->
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
9   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
10  <!-- Style -->
11  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
12  {% block style %}
13
14  {% endblock %}
15
16 </head>
17 <body>
18   {% include 'public/menu.html' %}
19   {% block content %}
20
21   {% endblock %}
22   {% include 'components/login_modal.html' %}
23 </body>
24 </html>

```

Para la estructura y diseño de nuestras páginas se utilizó HTML y CSS, así como estructuras de control que ayudan a controlar el flujo de ejecución del contenido que se renderiza, permitiendo mostrar los elementos especificados. Así se manejó la estructura para las demás pantallas (quienes somos, ubicación):

```
inicio.html X
WebSiteMindCare_Test > app > templates > public > inicio.html > ...
1
2 {% block title %} Inicio {% endblock %}
3
4 {% block style %}
5     <link rel="stylesheet" href="{{ url_for('static', filename='css/style_inicio.css') }}">
6 {% endblock %}
7
8
9 {% extends 'base.html' %}
10 {% block content %}
11
12     <div class="content">
13         
14         <div class="text">
15             <h1>Mind Care</h1>
16             <p>En <strong>Mind Care</strong>, consideramos que el bienestar emocional es
17             fundamental para tener una vida plena. Nuestro consultorio está diseñado
18             para brindarte un lugar seguro, cálido y profesional donde puedas examinar
19             tus emociones, pensamientos y retos personales. Estamos aquí para escucharte
20             y asistirte en la creación de herramientas que te ayuden a superar momentos
21             difíciles, buscar orientación o simplemente conocerte mejor.
22             <br><br>
23             En <strong>Mind Care</strong>, tu bienestar es nuestra prioridad.</p>
24         </div>
25     </div>
26 {% endblock %}
```

```
ubicacion.html X
WebSiteMindCare_Test > app > templates > public > ubicacion.html > ...
1 {% block title %} Ubicacion {% endblock %}
2
3 {% block style %}
4     <!--Style-->
5     <link rel="stylesheet" href="{{ url_for('static', filename='css/style_ubicacion.css') }}">
6 {% endblock %}
7
8 {% extends 'base.html' %}
9 {% block content %}
10     <div class="content">
11         <h1>Nos encontramos en</h1>
12         <p>Avenida Tecnológico s/n, Colonia Elías Calles, 21376 Mexicali, B.C.</p>
13         
14     </div>
15 {% endblock %}
```

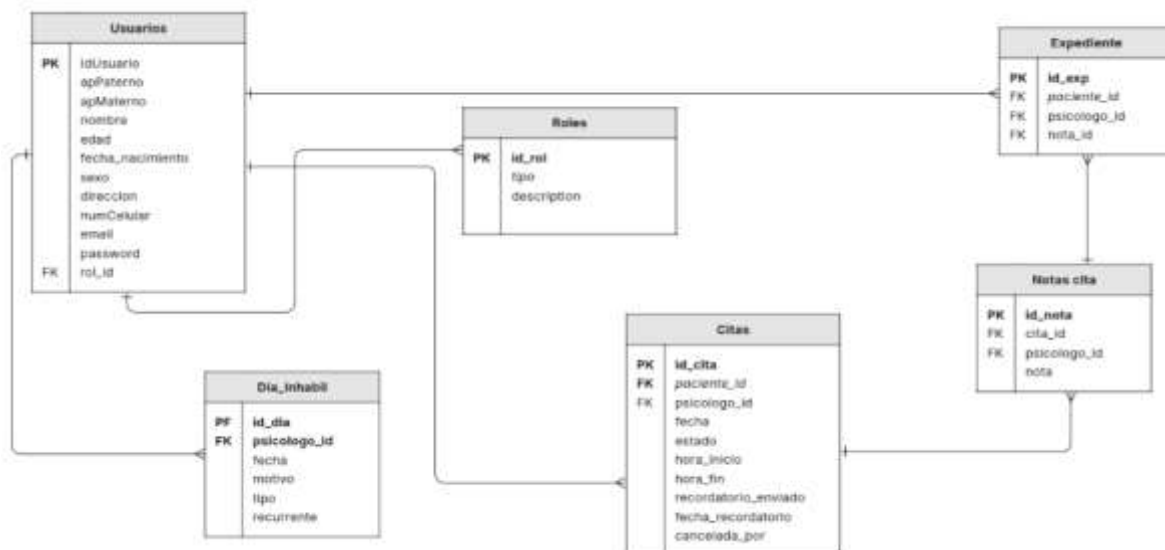


```

<> quienesSomos.html M X
WebSiteMindCare_Test > app > templates > public > <> quienesSomos.html > ...
1
2 {% block title %} Quienes somos {% endblock %}
3 {% block style %}
4 <!--Style-->
5 <link rel="stylesheet" href="{{ url_for('static', filename='css/style_quienesSomos.css') }}">
6 {% endblock %}
7
8 {% extends 'base.html' %}
9 {% block content %}
10 <div class="content">
11 
12 <div class="text">
13 <h1>¿Quienes somos?</h1>
14 <p><strong>Mind Care</strong> es un espacio diseñado para acompañarte
15 en tu proceso de autoconocimiento, sanación y crecimiento personal,
16 comprometido con ofrecerte un entorno seguro, empático y profesional,
17 donde puedas explorar tus pensamientos, emociones y desafíos sin juicio.
18 <br><br>
19 Nos guiamos por valores como la confidencialidad, el respeto y la
20 humanidad, integrando herramientas psicológicas con una experiencia
21 digital accesible y cercana.
22 <br><br>
23 En <strong>Mind Care</strong>, tu bienestar no es solo una meta: es nuestra prioridad.</p>
24 </div>
25 </div>
26 {% endblock %}

```

Y por la parte del diseño de nuestra base de datos, se contemplaron las tablas para **usuarios**, **roles**, **dia_inhabil**, **citas**, **notas_citas** y **expediente**:



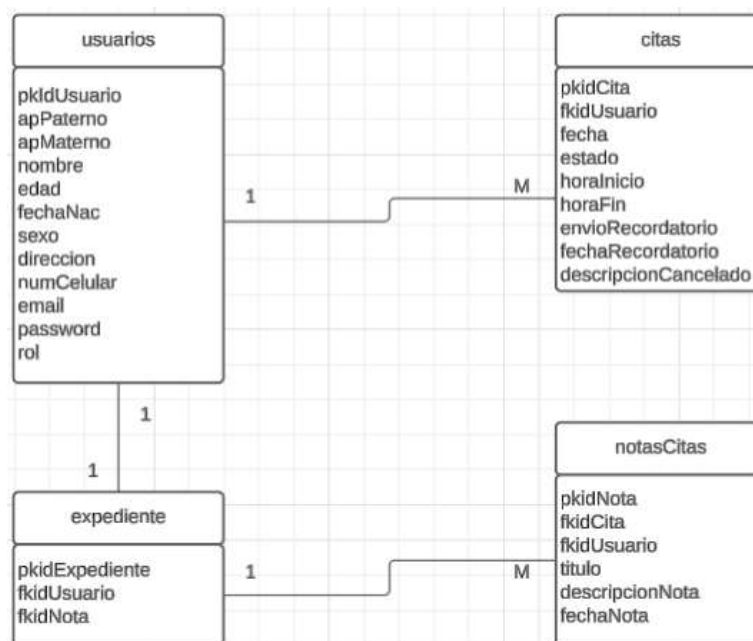
Debido al problema de redundancia con las relaciones en las tablas de **Usuarios**, **Citas**, **Notas cita** y **expediente** detectado al presentar el sprint 1, el diseño de la base de datos se modificará y se presentará en el segundo sprint (24 de octubre del 2025), ya que se necesita realizar un mejor análisis en la estructura de esta misma.

Sprint 2: 13 oct - 24 oct

Entregables:

- Creación de la base de datos.

En este segundo sprint, de acuerdo a la observación hecha por el profesor en el sprint pasado acerca del diseño de la base de datos, se realizó un nuevo análisis acerca de las relaciones y campos de las tablas con ayuda de nuestro docente, llegando a la siguiente conclusión:



Se decidió quitar las tablas de **roles** y **dia_inhabil**, dejándolas como campos en las tablas de **usuario (rol)** y **citas (dia_inhabil)**, llegando a una mayor limpieza en nuestro diseño de base de datos. Por lo que, para empezar a trabajar en nuestro proyecto, en nuestro archivo **requirements.txt** se agregó lo siguiente:

```
# --- Base de datos ---
Flask-SQLAlchemy==3.1.1
psycopg2-binary==2.9.9
```

Las cuales son librerías para trabajar con bases de datos en **Python**.

Posteriormente, se creó un archivo llamado **config.py** para establecer la conexión con nuestra base de datos, la cual contiene la **SQLALCHEMY_DATABASE_URI** que especifica el usuario, contraseña, puerto y base de datos a la que se va a conectar. Seguido de esto se tiene **SQLALCHEMY_TRACK_MODIFICATIONS** en false, lo cual sirve para ahorrar memoria y mejorar el rendimiento, y por último **SECRET_KEY** que es para firmar cookies y sesiones en nuestra app.

```
1  import os
2
3  class Config:
4      SQLALCHEMY_DATABASE_URI = 'postgresql://user_postgres:password_postgres@localhost:5433/db_postgres'
5      SQLALCHEMY_TRACK_MODIFICATIONS = False
6      SECRET_KEY = 'mind_care_project'
```

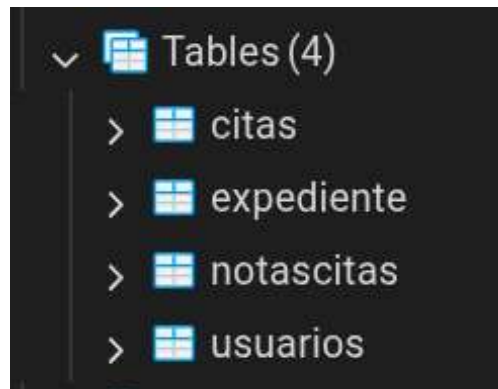
Una vez teniendo esto listo, se procedió a realizar un contenedor llamado **docker-compose.yml** para levantar una instancia de **postgres** y poder trabajar con nuestra base de datos, teniendo definidos datos como la imagen (postgres 16), el nombre del contenedor, así como el usuario, contraseña, nombre de la base de datos y el puerto. De igual manera, se estableció una imagen para usar **pgadmin**, la cual es la interfaz gráfica en donde trabajaremos de una mejor manera nuestra base de datos, teniendo establecidos el **email** y **password** con los que iniciaremos para conectarnos a nuestro servidor.

```

1  version: '3.8'
2
3  services:
4    db:
5      image: postgres:16
6      container_name: postgres_db
7      restart: always
8      environment:
9        POSTGRES_USER: user_postgres
10       POSTGRES_PASSWORD: password_postgres
11       POSTGRES_DB: db_postgres
12      ports:
13        - "5433:5432"
14      volumes:
15        - ./postgres_data:/var/lib/postgresql/data
16
17    pgadmin:
18      image: dpage/pgadmin4
19      container_name: pgadmin
20      restart: always
21      environment:
22        PGADMIN_DEFAULT_EMAIL: admin@admin.com
23        PGADMIN_DEFAULT_PASSWORD: admin
24      ports:
25        - "5050:80"
26      depends_on:
27        - db

```

Entonces, para ingresar a **pgadmin**, en nuestro navegador ingresamos **localhost:5050** (que es el puerto de pgadmin), seguido del **email** y **contraseña** especificadas en nuestro **docker-composed.yml**. Al entrar se accederá al servidor con el usuario, contraseña y nombre de la base de datos especificados en este mismo, para luego crear una base de datos en la cual procederemos a crear las tablas para nuestro proyecto (con sus respectivos campos y relaciones) quedando de la siguiente manera:



- Registro e inicio de sesión funcionales.

Para nuestras secciones de **registro** y **login** funcionales, en nuestro archivo de **main_routes.py** para el **login** se establecieron los métodos **GET** y **POST** para mostrar y procesar los datos enviados en el formulario. Luego, si el método es igual a **POST** extrae lo que el usuario escribió en los campos de **email** y **contraseña**, lo cual llama a una función **authenticate_user** (se verá más adelante) para verificar que los datos coincidan con un usuario previamente registrado, si es exitoso muestra un mensaje de "Inicio de sesión exitoso" y redirige a la página principal, en caso de que el email o contraseña no coincidan (o el usuario no esté previamente

registrado) mostrará un mensaje de “Correo o password incorrectos” y lo devolverá a la página de inicio.

```
27 @main_bp.route('/login', methods=['GET', 'POST'])
28 def login():
29
30     if request.method == 'POST':
31         email = request.form['email']
32         password = request.form['password']
33
34         user = authenticate_user(email, password)
35
36         if user:
37             session['user_id'] = user.id
38             session['user_email'] = user.email
39             session['user_nombre'] = f"{user.nombre}"
40             flash('Inicio de sesion exitoso', 'Success')
41             return redirect(url_for('main_bp.inicio'))
42         else:
43             flash('Correo o password incorrectos', 'danger')
44             return render_template('public/inicio.html', email='', password='')
```

Y para cerrar la sesión, con ***sesión.clear()*** se limpian los datos guardados durante la sesión, desconectando al usuario de esta y mostrando un mensaje de “Sesión cerrada correctamente” para redirigirlo a la página principal.

```
50 @main_bp.route('/logout')
51 def logout():
52     session.clear()
53     flash('Sesión cerrada correctamente', 'success')
54     return redirect(url_for('main_bp.inicio'))
```

Ahora, para nuestro registro creamos un modelo llamado ***user_model.py***, el cual defina la estructura de nuestra tabla ***usuarios*** en postgres. Primeramente se tiene el ***enum*** con los roles que pueden tener los usuarios (paciente, psicólogo y general), seguido de los campos de nuestra tabla (indicando la llave primaria, el tipo de dato, longitud de este, etc.)

```

1  from app.models import db
2  from datetime import datetime
3  from sqlalchemy import Enum
4  import enum
5
6  class RoleEnum(enum.Enum):
7      paciente = "paciente"
8      psicologo = "psicologo"
9      general = "general"
10
11  class User(db.Model):
12      __tablename__ = 'usuarios'
13      id = db.Column(db.Integer, primary_key=True)
14      paterno = db.Column(db.String(15), nullable=False)
15      materno = db.Column(db.String(15))
16      nombre = db.Column(db.String(30), nullable=False)
17      fecha_nacimiento = db.Column(db.Date, nullable=True)
18      sexo = db.Column(db.String(1), nullable=True)
19      direccion = db.Column(db.String(60), nullable=True)
20      celular = db.Column(db.String(10), nullable=True)
21      email = db.Column(db.String(254), unique=True, nullable=False)
22      password = db.Column(db.String(128), nullable=False)
23      role = db.Column(db.Enum(RoleEnum), default=RoleEnum.paciente, nullable=False)
24      created_at = db.Column(db.DateTime, default=datetime.utcnow)
25
26      def __repr__(self):
27          return f"<User {self.email}>"

```

Luego en nuestro archivo **main_routes.py** tenemos nuestra parte para el registro de usuarios, donde también se implementaron los métodos de **GET** y **POST**, si el método es igual a **POST**, extrae los datos que el usuario ingresó en el formulario (apellidos paterno y materno, nombre, edad, etc.) de acuerdo a los campos de la base de datos en la tabla **usuarios**.

```

57 @main_bp.route('/registro', methods=['GET', 'POST'])
58 ✓ def registro():
59     from app.models.user_model import RoleEnum
60
61     if request.method == 'POST':
62         email = request.form['email']
63
64         try:
65             # Captura de datos del formulario
66             paterno = request.form['paterno']
67             materno = request.form.get('materno') # opcional
68             nombre = request.form['nombre']
69             edad = request.form.get('edad')
70             fecha_nacimiento = request.form.get('fecha_nacimiento')
71             sexo = request.form.get('sexo')
72             direccion = request.form.get('direccion')
73             celular = request.form.get('celular')
74             email = request.form['email']
75             password = request.form['password']
76             role=RoleEnum.paciente

```

Lo siguiente es llamar al servicio (se mostrará más adelante) para registrar al usuario, asignando los datos capturados del formulario y pasarlos como argumentos con nombre.

```

78     # Llamar al service
79     user, errores = register_user(
80         paterno=paterno,
81         materno=materno,
82         nombre=nombre,
83         fecha_nacimiento=fecha_nacimiento,
84         sexo=sexo,
85         direccion=direccion,
86         celular=celular,
87         email=email,
88         password=password
89     )

```

Y después se hacen excepciones para indicar si los campos de email o celular ya están registrados (ya que deben ser únicos) y renderiza el formulario de registro. Si

no están repetidos muestra un mensaje de “Registro exitoso. Ahora puedes iniciar sesión”, renderizando la página de inicio donde se encuentra la opción de **login**.

```
91         if not user:
92             for e in errores:
93                 if e == "email":
94                     flash("El correo ya está registrado.", "warning")
95                 elif e == "celular":
96                     flash("El número de celular ya está registrado.", "warning")
97             return render_template("public/registro.html")
98         flash('Registro exitoso. Ahora puedes iniciar sesión.', 'success')
99         return redirect(url_for('main_bp.inicio'))
```

Entonces, ¿Dónde se crean las funciones de **authenticate_user** y **register_user**? Para ellas creamos un archivo llamado **auth_service.py**, donde creamos los dos métodos. Para **authenticate_user** recibe los parámetros de **email** y **password** y por parte de SLQALCHEMY nos brinda **User.query.filter_by** para buscar registros en la base de datos usando condiciones específicas (que en este caso busca el email y password).

```
1  # services/auth_service.py
2  from app.models.user_model import User
3  from app.models import db
4
5  def authenticate_user(email, password):
6      """
7      Busca un usuario que coincida con el email y password.
8      Retorna el objeto User si existe, o None si no.
9      """
10     user = User.query.filter_by(email=email, password=password).first()
11     return user
```

Y para el registro tenemos **register_user**, el cual recibe los datos del formulario y se asegura que el email y celular no estén repetidos, o dará un error.

```

13  def register_user(paterno, materno, nombre, fecha_nacimiento, sexo,
14                      direccion, celular, email, password):
15      """
16      Registra un usuario nuevo.
17      Retorna:
18          - User: si se creó correctamente
19          - None: si el email ya estaba registrado
20      """
21      errores = []
22
23      if User.query.filter_by(email=email).first():
24          errores.append("email")
25
26      if celular and User.query.filter_by(celular=celular).first():
27          errores.append("celular")
28
29      if errores:
30          return None, errores

```

Algo importante es la sesión de la base de datos, con **`db.session.add(user)`** prepara al usuario para ser guardado, agregándolo a la sesión de SQLAlchemy, y con **`db.session.commit()`** ya se guarda al usuario, devolviendo a este último e indicando que no hubo errores.

```

45      # Guardar en la base de datos
46      db.session.add(user)
47      db.session.commit()
48      return user, None

```

Y así se mira nuestra aplicación funcionando. Registramos a un paciente:

The screenshot shows a web form titled "Datos del paciente" (Patient Data). It is divided into two sections: "Datos personales" (Personal Data) and "Datos de contacto" (Contact Data). In the "Datos personales" section, the last name "Medina" and first name "Espinoza" are entered. The first name field also contains "Alejandra". The age is "18" and the birth date is "06/20/2007". The sex is selected as "F" (Female). In the "Datos de contacto" section, the address "Ejido Cuernavaca", phone number "6866641234", and email "ale@gmail.com" are entered. There is a password field with "XXXXXX" as a placeholder. At the bottom, there are two buttons: "Cancelar" (Cancel) and "Aceptar" (Accept).

Y queda registrado en la base de datos:

8	18	Rodriguez	Herrera	Maria Fernanda	2003-02-06	F	Pedregal
9	19	Medina	Espinosa	Alejandra	2007-06-20	F	Ejido Cuernavaca
6861234567	fer@gmail.com	fer123	paciente	2025-10-25 02:16:56.444546			
6866641234	ale@gmail.com	ale123	paciente	2025-10-25 04:52:26.481563			

Y cuando quiere iniciar sesión, lo hace de manera exitosa.

Iniciar sesión

Correo

ale@gmail.com

Contraseña

¿No tienes una cuenta? [Regístrate](#)

Aceptar

Inicio de sesion exitoso

OK

Y cuando inicia sesión, en el menú se muestra el nombre del paciente, así como su correo y la opción de cerrar sesión mediante un menú desplegable.

Alejandra

ale@gmail.com

Cerrar sesión

Sesión cerrada correctamente

OK

La observación de este sprint fue de nuevo la base de datos creada, ya que contiene relaciones que no estaban previstas en el diseño, volviendo a generar problemas de redundancia. Se arreglará y presentará para el tercer sprint que es el día 7 de noviembre del 2025.