



Quico Blesa Penadés

2º DAW 24/25

IES Doctor Lluis Simarro la Cabra

Jose Castillo



ÍNDICE

INTRODUCCIÓN.....	3
OBJETIVOS DE LA APP.....	4
TECNOLOGÍAS UTILIZADAS.....	5
Frontend (Entorno Cliente).....	5
Backend (Entorno Servidor).....	5
React (Framework de Javascript).....	6
DESARROLLO DEL PROYECTO.....	7
Diagrama de entidades.....	7
Diagrama de clases.....	8
Casos de uso.....	9
IMPLEMENTACIÓN DEL PROYECTO.....	21
Primera fase del proyecto.....	21
Dominio y subDominios.....	22
Servidor en azure.....	23
Subida de código al servidor.....	25
Interesante sobre react.....	30
Interesante sobre PHP.....	35
Vistas y acciones siguiendo el flujo de la app.....	39
1. Landing Page.....	39
2. Admin Portal.....	40
a. Login Administrador.....	40
b. Inicio Administrador.....	41
c. Categorías.....	42
d. Productos.....	44
e. Usuarios Médicos.....	47
3. User Portal.....	49
a. Login Usuario.....	49
b. Complete Login.....	50
c. Inicio Usuario.....	51
d. Clientes.....	52
e. Calendario.....	54
CONCLUSIÓN.....	58
RECURSOS.....	59
BIBLIOGRAFÍA.....	60

INTRODUCCIÓN

Durante una pequeña investigación sobre el funcionamiento interno de las clínicas médicas y su presencia en internet, observamos que muchas aún operan de forma obsoleta. Por ejemplo, los productos se registran en hojas de Excel o incluso en papel, y los médicos anotan los datos de sus pacientes sin información detallada ni un sistema organizado. Las citas, además, se gestionan de forma ineficiente.

A raíz de esta problemática, una clínica nos contactó con la necesidad de desarrollar una aplicación web que les permitiera mostrar sus servicios y productos en internet, al mismo tiempo que facilitara la gestión interna. Esta solución debía permitir al administrador gestionar productos, categorías y personal médico de forma ágil. También debía contar con una interfaz para que los médicos pudieran registrar información de los pacientes, visualizar estadísticas de productos más recomendados y asignar citas de manera centralizada, lo que permitiría mantener todo organizado y disponible para futuros análisis.

En este contexto nace **MediConnect**, una solución digital diseñada para mejorar la gestión interna de una clínica médica y aumentar su visibilidad en internet.

Los beneficios de esta aplicación son múltiples: permitirá al administrador gestionar todos los recursos de la clínica de forma sencilla y dinámica, con acceso a estadísticas y gráficos sobre productos, categorías y rendimiento general, mejorando así el control y la toma de decisiones.

OBJETIVOS DE LA APP

MediConnect tiene como objetivo desarrollar una aplicación web integral para la gestión de clínicas médicas, con los siguientes fines:

1. **Mejorar la gestión interna** de productos, categorías y personal médico mediante un panel de administración intuitivo.
2. **Registrar y gestionar pacientes** con información detallada, accesible para el equipo médico.
3. **Asignar y organizar citas médicas** de manera centralizada, eficiente y visual.
4. **Ofrecer estadísticas y gráficos interactivos** sobre productos más recomendados y rendimiento de la clínica.
5. **Aumentar la presencia online** de la clínica mediante una interfaz visible para los clientes con los servicios y productos ofrecidos.
6. **Facilitar la toma de decisiones** gracias a una plataforma basada en datos y registros actualizados.
7. **Ofrecer una experiencia unificada** tanto para el administrador como para los médicos.

TECNOLOGÍAS UTILIZADAS

Para el desarrollo de la aplicación **MediConnect**, se ha optado por una arquitectura basada en la separación del backend y frontend, creando una rest api en php para la conexión del frontend con el backend

Frontend (Entorno Cliente)

El **frontend** es la parte de la aplicación con la que interactúan los usuarios. Para su desarrollo se ha utilizado **React**, una biblioteca moderna y eficiente de JavaScript para la construcción de interfaces de usuario. React permite ofrecer una experiencia dinámica e intuitiva al usuario.

- **Librería principal:** React (framework de JavaScript).
- **Herramienta de compilación:** Vite.
- **Estilado de la app:** Bootstrap 5 y CSS utilizando Flexbox y Grid.

Backend (Entorno Servidor)

El **backend** se encarga de procesar las solicitudes de los clientes, proporcionando los datos requeridos y gestionando la lógica del sistema.

- **Lenguaje:** PHP nativo, con el cual se ha desarrollado una API que suministra los recursos a la aplicación y gestiona las peticiones de los usuarios.
- **Gestor de base de datos:** MySQL, utilizado para almacenar y administrar los datos de forma eficiente y segura.
- **HeidiSQL:** empleada para conectarse, crear, modificar y consultar bases de datos de forma visual, sin necesidad de escribir todas las consultas SQL manualmente.

React (Framework de Javascript)

React es un framework de JavaScript que está diseñado para construir la interfaz de usuario a partir de pequeñas piezas reutilizables llamadas componentes que gestionan su propio estado interno. También react utiliza un enfoque donde los datos sólo fluyen en la misma dirección que es de componentes padre a componentes hijos, esto se hace pasando los datos como propiedades del componente también conocidas como "props".

- **Componentes:** Piezas pequeñas de código que forman la interfaz de usuario.
- **Props:** Propiedades que se pasan de un componente padre a un componente hijo.

DESARROLLO DEL PROYECTO

Diagrama de entidades

En esta imagen se muestra las **relaciones** con sus **cardinalidades** respectivas entre entidades. (Un administrador crea productos, una categoría contiene productos, un administrador crea usuarios, un usuario tiene un perfil, un usuario crea citas, un usuario crea clientes, un cliente puede tener más de una cita, muchos productos pueden estar recomendados a muchos clientes.)

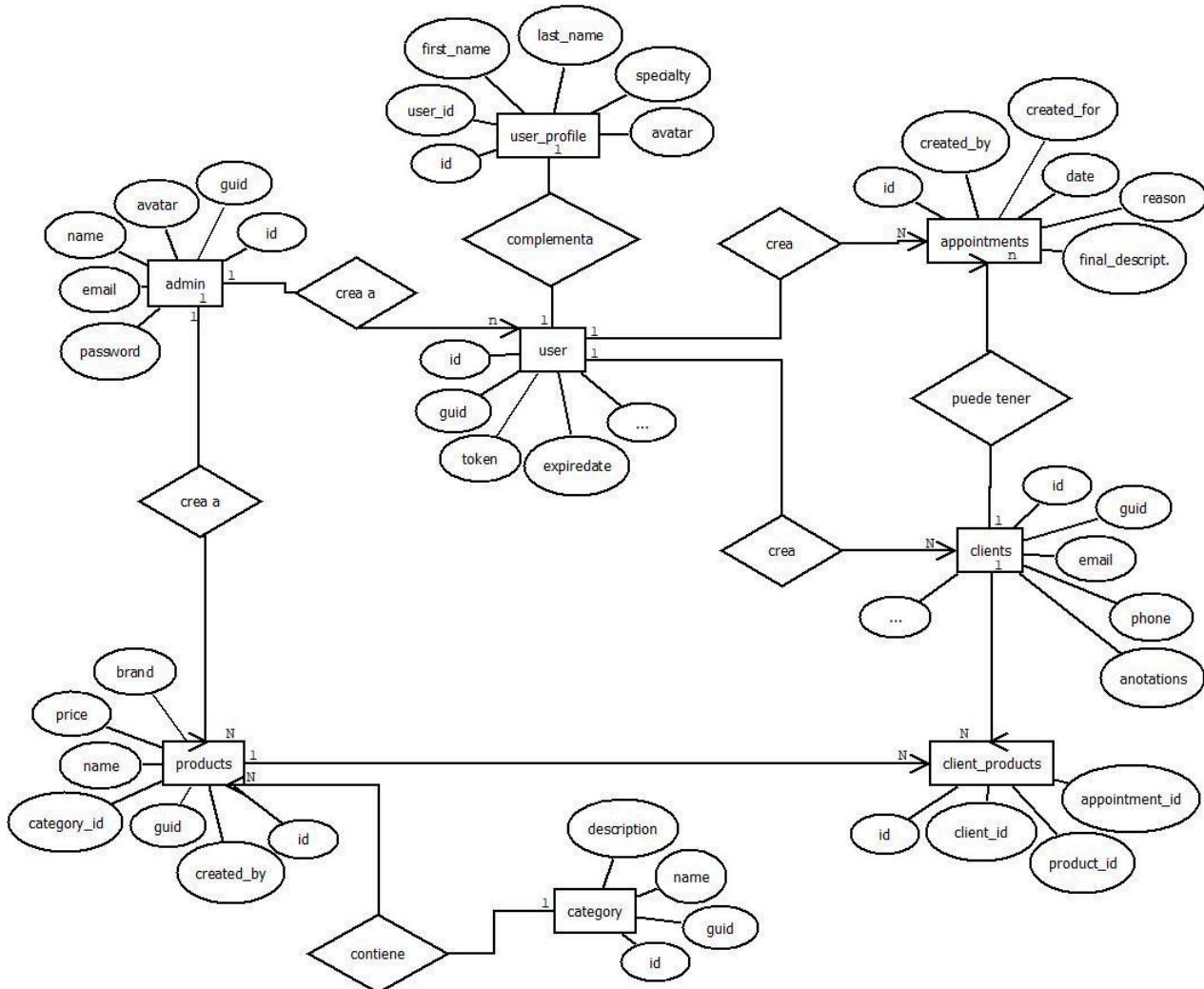
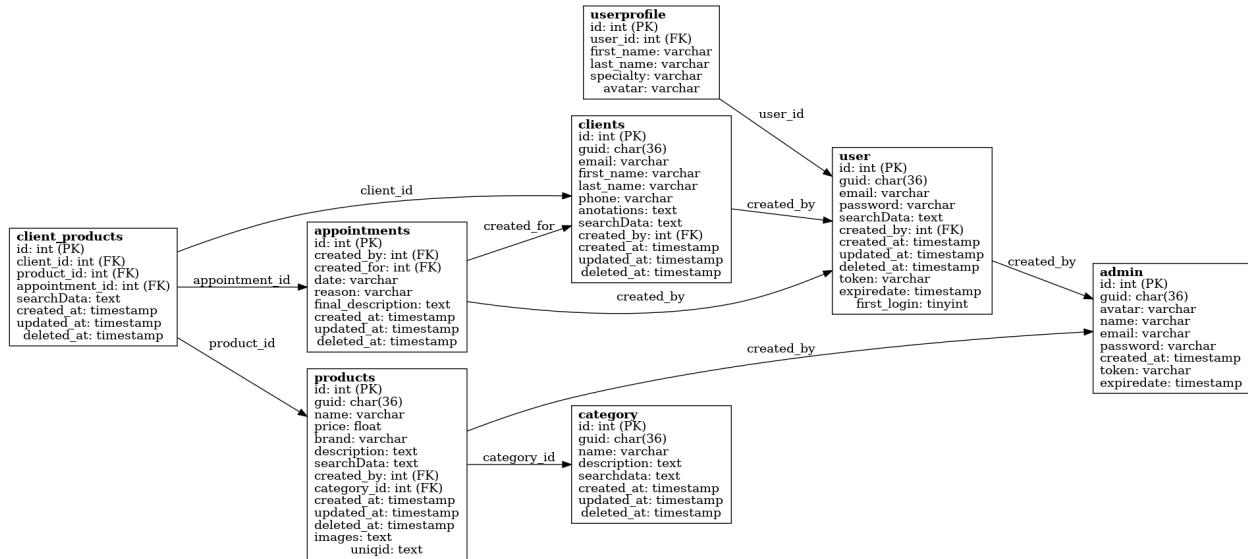
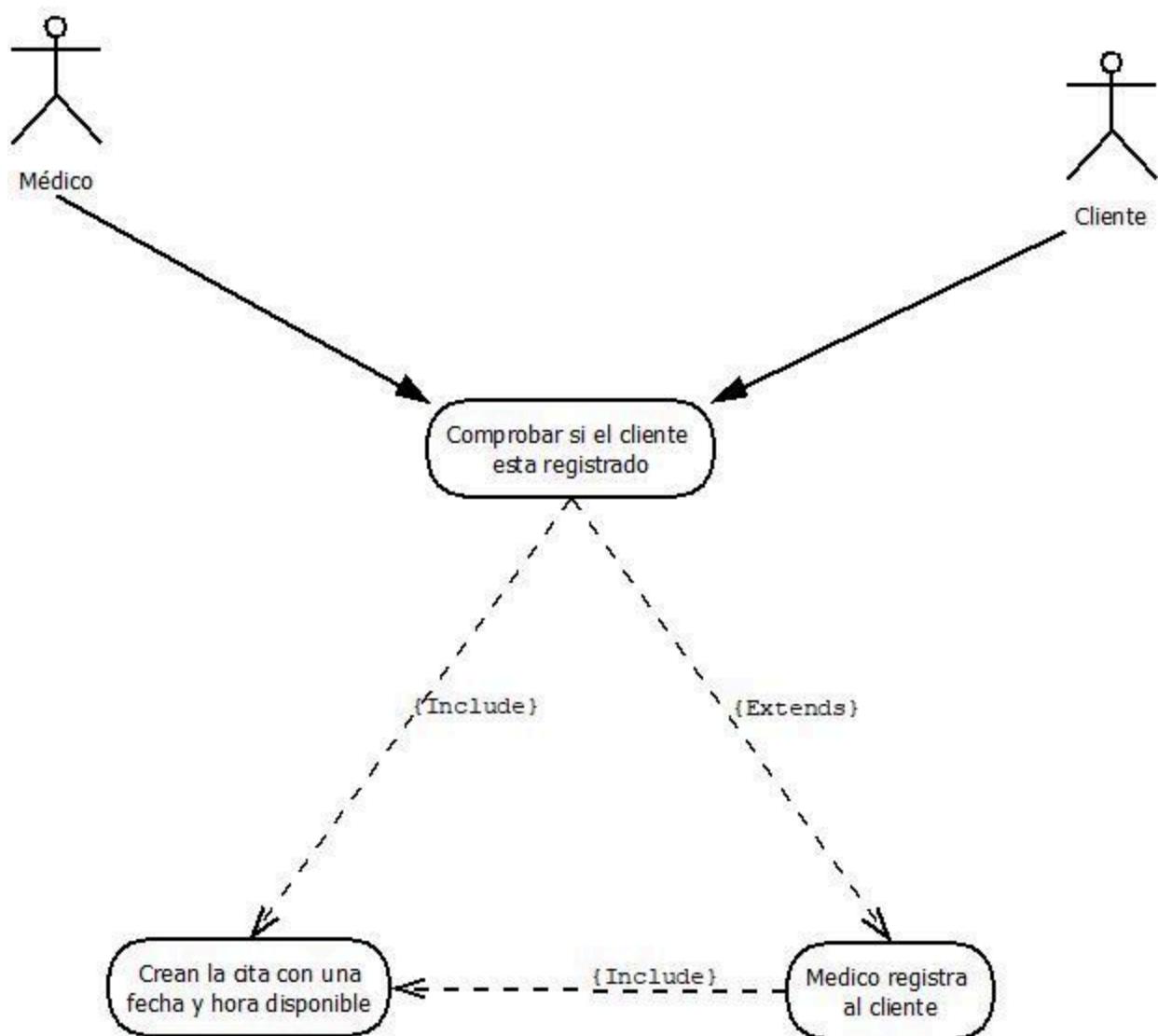


Diagrama de clases



Casos de uso

-> Crear una cita entre médico y cliente



Nombre de la acción: Comprobar si el usuario está registrado

Actores: Médico y cliente.

Condiciones anteriores a la acción: El médico debe estar anteriormente registrado por el administrador de la clínica.

Flujo de la acción:

1. El cliente entrega sus datos a su médico.
2. El médico comprueba que el cliente está registrado. (Haciendo uso de los filtros en el apartado “Clientes”).

Condiciones posteriores a la acción: Una vez comprobado seguir con el caso de uso.

Alternativas: Sin alternativas.

Nombre de la acción: Registrar al cliente

Actores: Médico y cliente.

Condiciones anteriores a la acción: El médico debe haber iniciado sesión anteriormente.

Flujo de la acción:

1. El médico pregunta los datos necesarios para el registro al cliente y el cliente le entrega sus datos.
2. El médico le realiza un examen genérico para guardar en la base de datos el estado del cliente.

Condiciones posteriores a la acción: Una vez registrado ya se pueden crear citas.

Alternativas: Sin alternativas.

Nombre de la acción: Crear cita para cliente

Actores: Médico y cliente

Condiciones anteriores a la acción: El cliente debe estar registrado anteriormente por su médico.

Flujo de la acción:

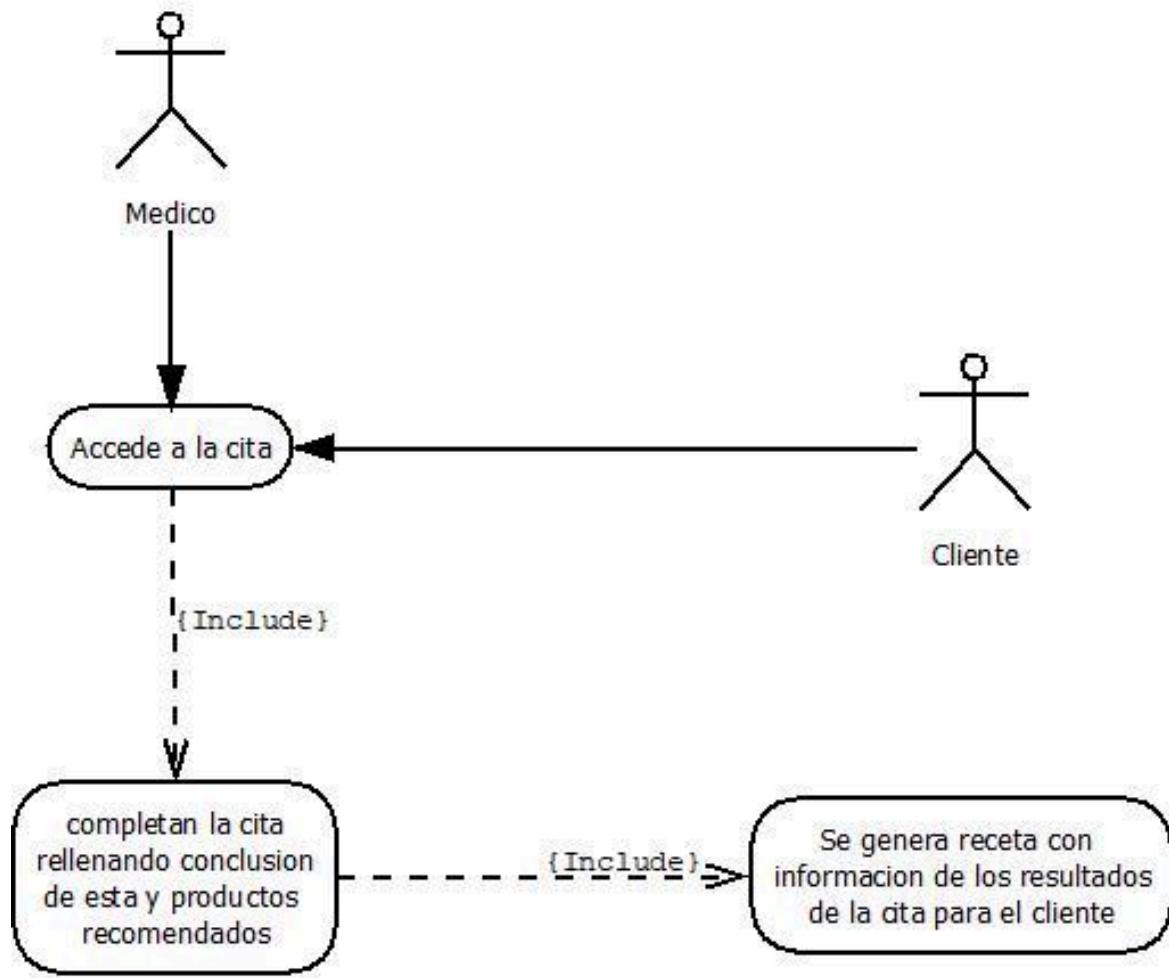
1. El médico entra al apartado de calendario para poder visualizar días y horas que tiene libres y así comunicárselo al cliente.
2. El cliente elige un día y hora para la cita.
3. El médico crea la cita para que se guarde en la base de datos.

Condiciones posteriores a la acción: Una vez creada la cita termina el caso de uso.

Alternativas:

1. Entre el tiempo que se creó la cita y el día de la cita podría ser cancelada esta y se terminaría el caso de uso.

-> Completar una cita y crear una receta médica



Nombre de la acción: Acceder a la cita

Actores: Médico y cliente
Condiciones anteriores a la acción: El médico debe estar anteriormente registrado por el administrador de la clínica y el cliente debe asistir y estar registrado.
Flujo de la acción: <ol style="list-style-type: none">1. El cliente llega a la clínica por el motivo de su visita.2. El médico busca la cita en su apartado de trabajo diario y accede a ella.
Condiciones posteriores a la acción: Una vez dentro de la cita seguir con el caso de uso.
Alternativas: El cliente no se presenta a la cita.

Flujo de la acción:

1. El cliente llega a la clínica por el motivo de su visita.
2. El médico busca la cita en su apartado de trabajo diario y accede a ella.

Nombre de la acción: Completar cita

Actores: Médico y cliente
Condiciones anteriores a la acción: El médico debe haber accedido a la cita y el cliente estar presente.
Flujo de la acción: <ol style="list-style-type: none">1. El médico examina la razón de la cita del cliente.2. El médico examina al cliente y realiza los exámenes médicos necesarios.3. El médico rellena la receta con la conclusión y los productos recomendados para el tratamiento del cliente.
Condiciones posteriores a la acción: Una vez completada la cita sigue el caso de uso.
Alternativas: Sin alternativas.

Flujo de la acción:

1. El médico examina la razón de la cita del cliente.
2. El médico examina al cliente y realiza los exámenes médicos necesarios.
3. El médico rellena la receta con la conclusión y los productos recomendados para el tratamiento del cliente.

Nombre de la acción: Crear receta médica

Actores: Médico y cliente

Condiciones anteriores a la acción: El médico debe haber completado la cita propuesta por el cliente.

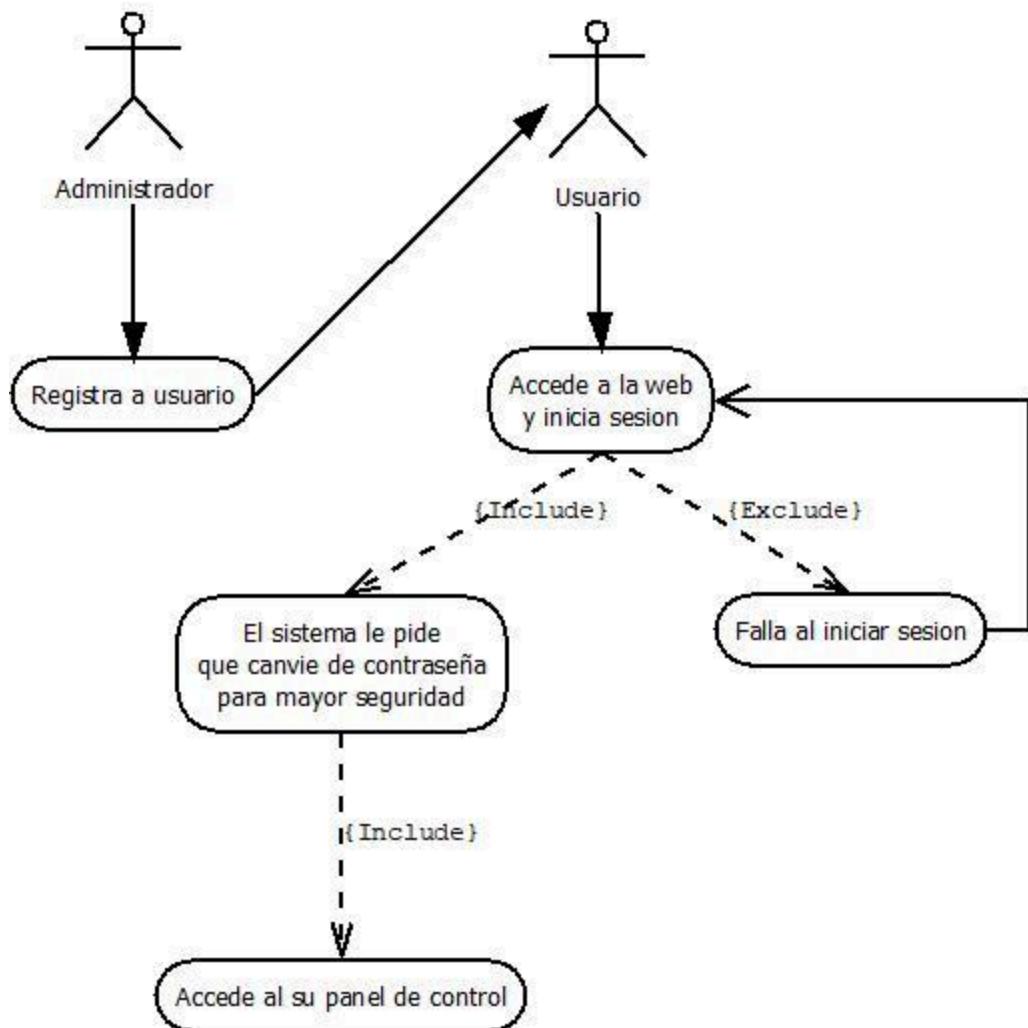
Flujo de la acción:

1. El médico descarga el pdf con la receta.
2. El cliente recibe la receta para poder dirigirse al mostrador y comprar los productos.

Condiciones posteriores a la acción: Una vez entregada la receta se termina el caso de uso.

Alternativas: El cliente puede no comprar ningún producto.

-> Registrar un usuario y iniciar sesion de usuario



Nombre de la acción: Registrar usuario

Actores: Administrador y usuario
Condiciones anteriores a la acción: Sólo un administrador puede hacer la acción de registrar a un usuario para que este pueda acceder a su panel de control.
Flujo de la acción: <ol style="list-style-type: none">1. El administrador contrata a un médico.2. El administrador se dirige al apartado “usuario” para llenar el cuestionario con la información del usuario.
Condiciones posteriores a la acción: Una vez creado el usuario las próximas acciones son por parte del usuario.

1. El administrador contrata a un médico.
2. El administrador se dirige al apartado “usuario” para llenar el cuestionario con la información del usuario.

Nombre de la acción: Acceder al sitio web y iniciar sesión

Actores: usuario y administrador
Condiciones anteriores a la acción: El administrador de la clínica debe haber creado al usuario desde su panel de control.
Flujo de la acción: <ol style="list-style-type: none">1. El usuario accede a la app y se sitúa en el apartado de inicio de sesión.2. Introduce su email y contraseña genérica que le proporciona el administrador.3. El usuario completa el formulario de cambio de contraseña.4. El usuario accede a su panel de control.
Condiciones posteriores a la acción: Una vez el usuario acceda a su panel de control y se complete el formulario de cambio de contraseña termina el caso de uso.

1. El usuario accede a la app y se sitúa en el apartado de inicio de sesión.
2. Introduce su email y contraseña genérica que le proporciona el administrador.
3. El usuario completa el formulario de cambio de contraseña.
4. El usuario accede a su panel de control.

-> Otros casos de uso...

1. Iniciar Sesión

- a. Actores: Administrador y usuario
- b. Descripción: Tanto el administrador como el usuario ingresan sus credenciales en sus respectivos inicios de sesión para acceder a la aplicación.

2. Editar Perfil

- a. Actores: Administrador y usuario
- b. Descripción: Tanto el administrador como el usuario pueden modificar sus datos personales.

3. Editar cuenta

- a. Actores: Administrador y usuario
- b. Descripción: Tanto el administrador como el usuario pueden modificar sus datos de cuenta como email y contraseña.

4. Elegir Idioma

- a. Actores: Todo tipo de usuario
- b. Descripción: Todo tipo de usuario puede elegir el idioma de la interfaz de la aplicación..

5. Cerrar sesión

- a. Actores: Administrador y usuario
- b. Descripción: Tanto el administrador como el usuario cierra su sesión en la aplicación.

6. Casos de uso del Administrador

a. Gestionar Categorías

- i. Crear categoría
 1. Actor: Administrador
 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
- ii. Editar categoría
 1. Actor: Administrador
 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.

-
- iii. Borrar categoría
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
 - iv. Visualizar todas las categorías
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
- b. Gestionar Productos
- i. Crear producto
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede crear nuevos productos en la base de datos.
 - ii. Editar producto
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede editar productos en la base de datos.
 - iii. Borrar producto
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede borrar productos de la base de datos.
 - iv. Visualizar todos los productos
 - 1. Actor: Administrador
 - 2. Descripción: El administrador visualiza todos los productos de la base de datos.
- c. Gestionar Usuarios
- i. Crear usuario
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede crear nuevos usuarios en la base de datos.
 - ii. Editar usuario
 - 1. Actor: Administrador

-
- 2. Descripción: El administrador puede editar información del usuario en la base de datos.
 - iii. Borrar usuario
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede borrar usuarios de la base de datos.
 - iv. Visualizar todos los usuarios
 - 1. Actor: Administrador
 - 2. Descripción: El administrador visualizará todos los usuarios registrados en la clínica.

7. Casos de uso del Usuario

- a. Gestionar Clientes
 - i. Crear Cliente
 - 1. Actor: Usuario
 - 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
 - ii. Editar Cliente
 - 1. Actor: Usuario
 - 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
 - iii. Borrar Cliente
 - 1. Actor: Usuario
 - 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
 - iv. Visualizar todos sus Clientes
 - 1. Actor: Usuario
 - 2. Descripción: El administrador puede crear nuevas categorías en la base de datos.
- b. Gestionar Citas
 - i. Crear producto
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede crear nuevos productos en la base de datos.

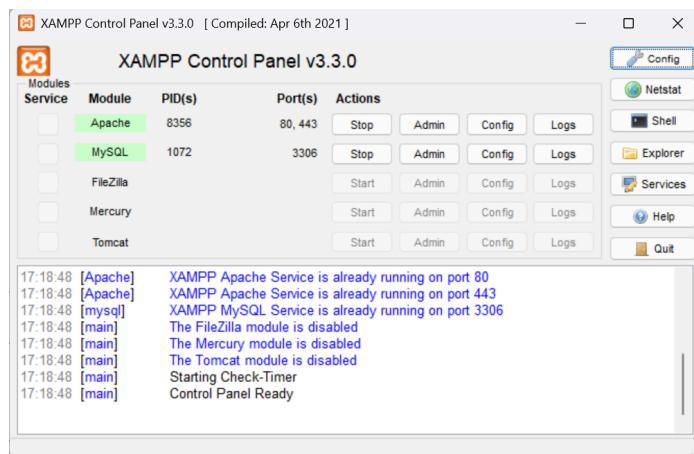
-
- ii. Editar producto
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede editar productos en la base de datos.
 - iii. Borrar producto
 - 1. Actor: Administrador
 - 2. Descripción: El administrador puede borrar productos de la base de datos.
 - iv. Visualizar todos los productos
 - 1. Actor: Administrador
 - 2. Descripción: El administrador visualiza todos los productos de la base de datos.

IMPLEMENTACIÓN DEL PROYECTO

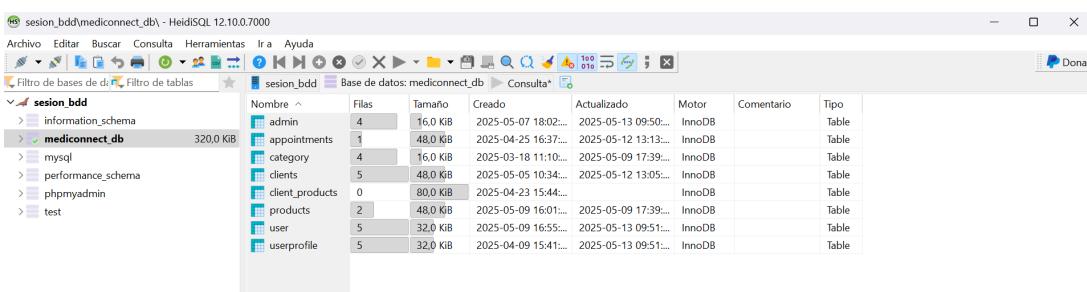
En este apartado explicaremos como trabajamos desde un principio corriendo la aplicación en local hasta desplegar esta en un servidor y dominio propio.

Primera fase del proyecto

Durante el inicio del proyecto se trabajó completamente en local por lo tanto con la herramienta XAMPP podíamos correr tanto apache como MYSQL.



También con la herramienta de diseño HeidiSQL se fueron diseñando las diferentes versiones de la base de datos ya que con el progreso del proyecto podían hacer falta más columnas o tipos de datos en esta, desde heidisQL se exportó finalmente el sql de creación de la base de datos en MYSQL.



Dominio y subDominios

Desde la plataforma hostinger que es un proveedor de dominios y hostings se buscó un dominio para la app.

Una vez ya nos pertenecía el dominio creamos dos subdominios uno para la app y otro para la api apuntando a la misma ip.

“A” apuntando a la ip de la máquina virtual de azure y dos “CNAME” para cada uno de los dos subdominios.

Tipo	Nombre	Prioridad	Contenido	TTL	Borrar	Editar
A	www	0	9.223.1.159	60	Borrar	Editar
CNAME	app	0	blesadev.es	60	Borrar	Editar
CNAME	api	0	blesadev.es	60	Borrar	Editar
CAA	@	0	0 issuewild "digicert.com"	14400	Borrar	Editar
CAA	@	0	0 issuewild "sectigo.com"	14400	Borrar	Editar
CAA	@	0	0 issuewild "pki.goog"	14400	Borrar	Editar
CAA	@	0	0 issuewild "letsencrypt.org"	14400	Borrar	Editar
CAA	@	0	0 issuewild "globalsign.com"	14400	Borrar	Editar
CAA	@	0	0 issuewild "comodoca.com"	14400	Borrar	Editar
CAA	@	0	0 issue "sectigo.com"	14400	Borrar	Editar
CAA	@	0	0 issue "pki.goog"	14400	Borrar	Editar
CAA	@	0	0 issue "letsencrypt.org"	14400	Borrar	Editar
CAA	@	0	0 issue "globalsign.com"	14400	Borrar	Editar
CAA	@	0	0 issue "digicert.com"	14400	Borrar	Editar
CAA	@	0	0 issue "comodoca.com"	14400	Borrar	Editar
A	@	0	9.223.1.159	60	Borrar	Editar

Servidor en azure

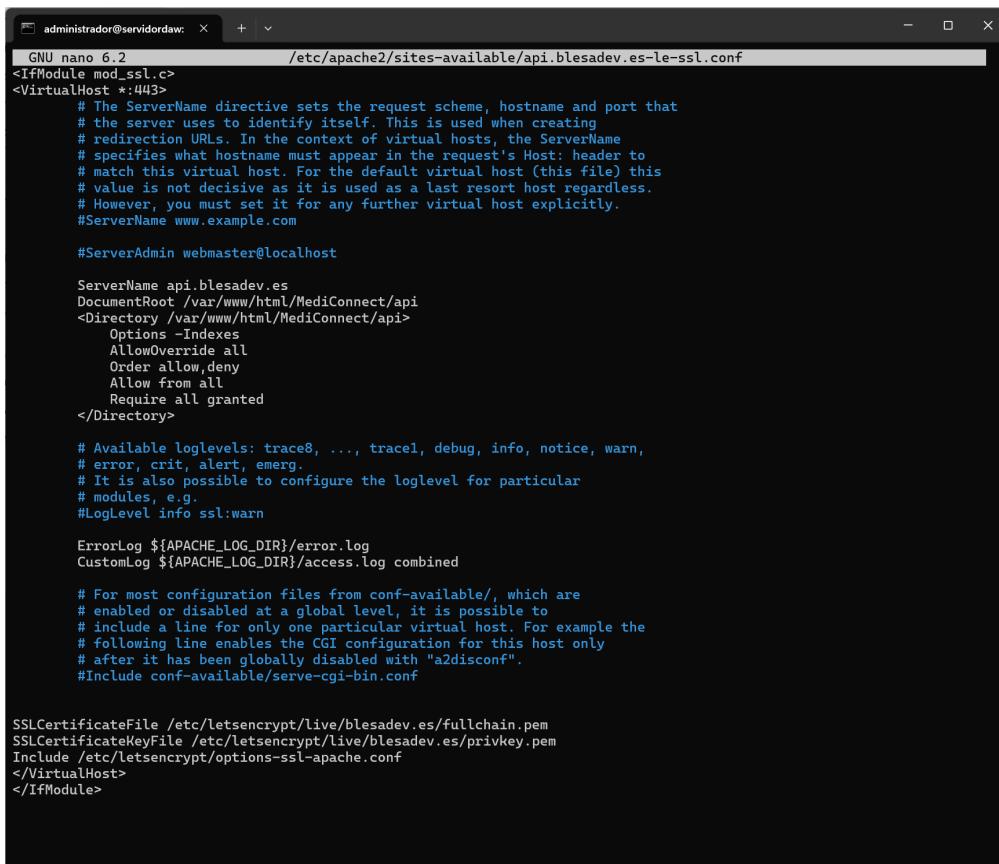
Una vez ya el proyecto terminado y sin errores en local se creó una máquina virtual en azure con el sistema operativo de ubuntu Server 22.24, a esta máquina se le instaló mySQL para poder más tarde crear la base de datos dentro del servidor.

También se le instaló apache que es un servidor web de código abierto utilizado para alojar aplicaciones en internet.

Se configuraron dos virtualHost para que no hubiera conflictos entre la api y la app.

1. VirtualHost para la api

Como se puede observar apunta a la carpeta /var/www/html/MediConnect/api.



```
GNU nano 6.2          /etc/apache2/sites-available/api.blesadev.es-le-ssl.conf
<IfModule mod_ssl.c>
<VirtualHost *:443>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    #ServerAdmin webmaster@localhost

    ServerName api.blesadev.es
    DocumentRoot /var/www/html/MediConnect/api
    <Directory /var/www/html/MediConnect/api>
        Options -Indexes
        AllowOverride all
        Order allow,deny
        Allow from all
        Require all granted
    </Directory>

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

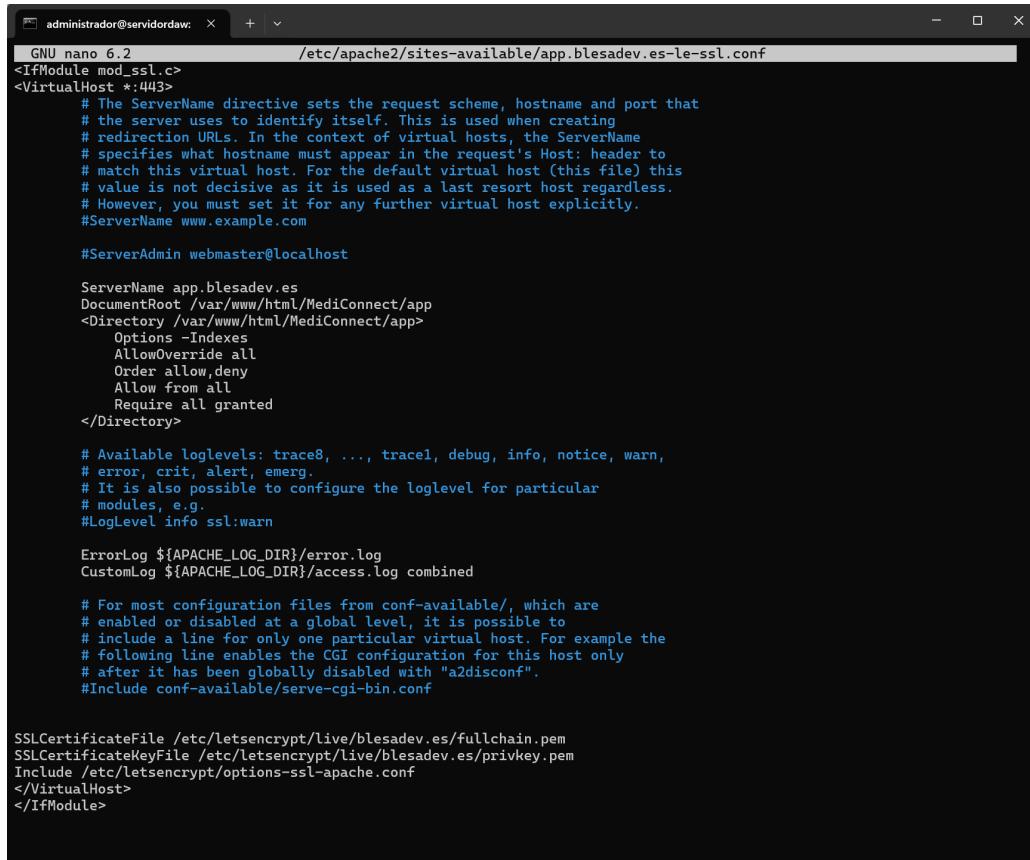
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf

    SSLCertificateFile /etc/letsencrypt/live/blesadev.es/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/blesadev.es/privkey.pem
    Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
</IfModule>
```

2. VirtualHost para la app

Como se puede observar apunta a la carpeta /var/www/html/MediConnect/app.



```
GNU nano 6.2          /etc/apache2/sites-available/app.blesadev.es-le-ssl.conf
<IfModule mod_ssl.c>
<VirtualHost *:443>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    #ServerAdmin webmaster@localhost

    ServerName app.blesadev.es
    DocumentRoot /var/www/html/MediConnect/app
    <Directory /var/www/html/MediConnect/app>
        Options -Indexes
        AllowOverride all
        Order allow,deny
        Allow from all
        Require all granted
    </Directory>

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf

    SSLCertificateFile /etc/letsencrypt/live/blesadev.es/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/blesadev.es/privkey.pem
    Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
</IfModule>
```

Subida de código al servidor

->Base de Datos

Para alojar la base de datos de mi proyecto primero instalé MySQL en el servidor y cree un usuario con todos los permisos sobre la base de datos mediconnect_db para evitar usar al usuario root para operaciones habituales sobre la base de datos.

```
administrador@servidordaw:~$ sudo mysql -u user -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.42-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
| mediconnect_db   |
| performance_schema |
+-----+
3 rows in set (0.01 sec)

mysql> |
```

Una vez creada la base de datos y con un usuario con permisos para poder trabajar con ella es el momento de transferir el sql de creación generado por HeidiSQL hasta el servidor para ejecutarlo y poder ver las tablas de la base de datos.

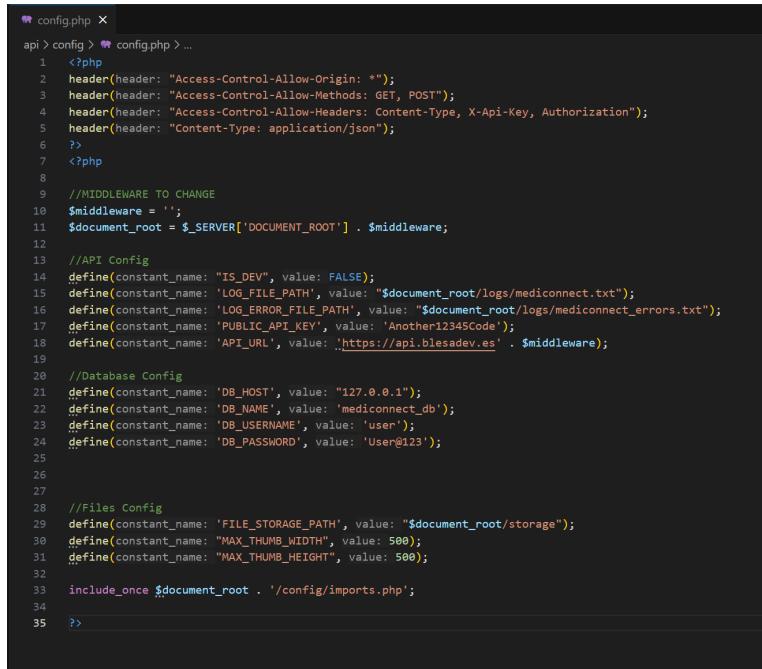
```
mysql> USE mediconnect_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_mediconnect_db |
+-----+
| admin
| appointments
| category
| client_products
| clients
| products
| user
| userprofile
+-----+
8 rows in set (0.00 sec)

mysql>
```

>Api del proyecto

Como la base de datos no se encontraba ya en el mismo lugar ni con el mismo nombre tuve que adaptar la configuración de la api para que esta apuntar correctamente al lugar de la base de datos que en este caso al estar en el mismo servidor apuntaba a localhost con el usuario creado anteriormente, también la ruta para llamarla se modificó.

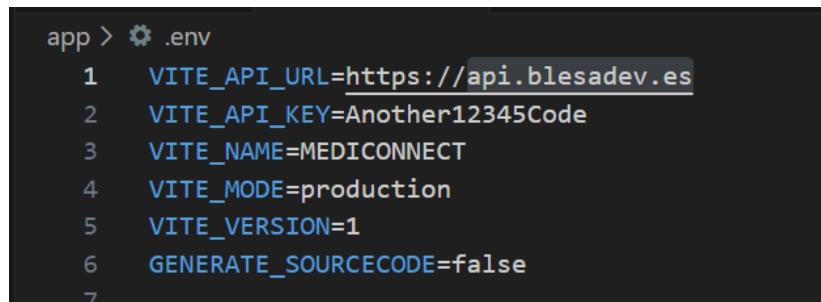


```
config.php x
api > config > config.php > ...
1  <?php
2  header(header: "Access-Control-Allow-Origin: *");
3  header(header: "Access-Control-Allow-Methods: GET, POST");
4  header(header: "Access-Control-Allow-Headers: Content-Type, X-Api-Key, Authorization");
5  header(header: "Content-type: application/json");
6  ?>
7  </php
8
9 //MIDDLEWARE TO CHANGE
10 $middleware = '';
11 $document_root = $_SERVER['DOCUMENT_ROOT'] . $middleware;
12
13 //API Config
14 define(constant_name: "IS_DEV", value: FALSE);
15 define(constant_name: 'LOG_FILE_PATH', value: "$document_root/logs/mediconnect.txt");
16 define(constant_name: 'LOG_ERROR_FILE_PATH', value: "$document_root/logs/mediconnect_errors.txt");
17 define(constant_name: 'PUBLIC_API_KEY', value: 'Another12345Code');
18 define(constant_name: 'API_URL', value: ..https://api.blesadev.es' . $middleware);
19
20 //Database Config
21 define(constant_name: 'DB_HOST', value: "127.0.0.1");
22 define(constant_name: 'DB_NAME', value: 'mediconnect_db');
23 define(constant_name: 'DB_USERNAME', value: 'user');
24 define(constant_name: 'DB_PASSWORD', value: 'User@123');
25
26
27
28 //Files Config
29 define(constant_name: 'FILE_STORAGE_PATH', value: "$document_root/storage");
30 define(constant_name: "MAX_THUMB_WIDTH", value: 500);
31 define(constant_name: "MAX_THUMB_HEIGHT", value: 500);
32
33 include_once $document_root . '/config/imports.php';
34
35 ?>
```

La api a la hora se transferirla al servidor simplemente se comprime en .Zip para que al transferir no tarde tanto.

>App del proyecto

En el archivo “.env” de la app donde se encuentran variables estáticas que se utilizan en la app como para los endpoints... se tuvo que adaptar la ruta donde se apuntaba a la api y el modo de producción.



```
app > .env
1  VITE_API_URL=https://api.blesadev.es
2  VITE_API_KEY=Another12345Code
3  VITE_NAME=MEDICONNECT
4  VITE_MODE=production
5  VITE_VERSION=1
6  GENERATE_SOURCECODE=false
7
```

>Método de transferencia y estructura final del proyecto en el servidor

Para transferir los datos desde local una vez ya preparados se utilizó el comando scp que es una herramienta de línea de comandos que permite copiar archivos entre un ordenador local y un servidor remoto a través de SSH, este método trabaja con transferencia cifrada por lo tanto es más seguro. El usuario que se usa para la transferencia debe tener permisos suficientes. El comando sería así.

```
scp archivo/ruta al archivo usuario@ip_host:/ruta_remota
```

Estructura final en /var/www/html:

```
administrador@servidor:~$ tree -L 2 /var/www/html/MediConnect/
/var/www/html/MediConnect/
├── api
│   ├── composer.json
│   ├── composer.lock
│   ├── config
│   ├── controller
│   ├── endpoints
│   ├── helpers
│   ├── logs
│   ├── objects
│   ├── resources
│   ├── storage
│   └── vendors
└── app
    ├── Translations
    ├── assets
    └── index.html
        └── vite.svg
```

De esta manera ya estaría en completa producción nuestra app:
app.blesadev.es

->Flujo de datos entre frontend y backend

La comunicación entre el frontend y el backend se realiza mediante solicitudes **HTTP** a través de la api construida en **PHP**. Principalmente las solicitudes son de tipo **POST/GET** aunque no siempre son iguales ya que cuando se pasan imágenes o diferentes tipos de archivos cambia la forma de transmitir los datos.

->solicitud desde react

Desde el front construido en react enviamos de esta manera la solicitud al backend, la función request es un **custom hook** que hace uso de **axios** para enviar la solicitud **Post** al servidor.

```
const handleSubmit = () => {
  if (checkForm()) {
    setSubmitting(true);
    request("post", getEndpoint(EndpointsUser.Clients.create), { ...data })
      .then(() => {
        push(Paths[Views.clients].path);
        successNotification(ViewStrings.messageSuccess);
      })
      .catch((err) => errorNotification(err.message))
      .finally(() => setSubmitting(false))
  }
}
```

```
switch (type.toLowerCase()) {
  case "post":
    return await axios
      .post(url, payload, {
        headers: {
          ...headers,
          // "Content-type": "app
        },
      })
      .then(checkResponse)
      .catch(checkError);
```

->Recepción desde php

Desde el back en php se recibe la información con la función **postInput()**, a continuación se crea la transición de la conexión con la base de datos y se comprueba si necesitas autenticación con las funciones **checkUser()** o **checkAdmin()**.

Se valida la información recibida por el **postInput()** con la función validate, si la validación es correcta se realizan las acciones necesarias y se devuelven los datos al cliente usando **Response::sendResponse()**.

```
<?php

include_once "../../config/config.php";

$database = new Database();
$db = $database->getConnection();

$data = postInput();

try {
    $db->beginTransaction();
    $userId = checkAuthUser();

    $input = validate(data: $data, rules: [
        "firstName" => "required|string",
        "lastName" => "required|string",
        "email" => "required|string",
        "phone" => "required|string",
        "anotations" => "required|string",
    ]);
}

$client_exist = Client::getByEmail(db: $db, email: $input->email);
if (!$client_exist) {
    $client = new Client(db: $db);
    $client->first_name = $input->firstName;
    $client->last_name = $input->lastName;
    $client->email = $input->email;
    $client->phone = $input->phone;
    $client->anotations = $input->anotations;
    $client->created_by = $userId;

    $client->store();
}

else {
    createException(message: "Email already exist", code: 409);
}

$db->commit();
Response::sendResponse();
} catch (\Exception $th) {
    $db->rollBack();
    print_r(value: json_encode(value: array("status" => false, "message" => $th->getMessage())));
}
```

Interesante sobre react

->routes

Las rutas de react funcionan con la librería react-router-dom estas se renderizan en el archivo main.jsx y todas las rutas se encuentran definidas en el archivo routes.constants.jsx, aquí se encuentran diferentes arrays que alberga la ruta dependiendo del tipo de autorización por ejemplo:

```
34
35 const getRoute = (path, component, exact = true) => ({
36   path,
37   component,
38   exact,
39 });
40
41 // Region User
42
43 export const AuthRoutes = [
44   getRoute(Paths[Views.login].path, LoginUser),
45   getRoute(Paths[Views.completeUser].path, CompleteUser),
46 ];
47
48
49 export const AppRoutes = [
50   getRoute(Paths[Views.homeUser].path, HomeUser),
51
52   getRoute(Paths[Views.clients].path, Clients),
53   getRoute(Paths[Views.new_client].path, NewClient),
54   getRoute(Paths[Views.edit_client].path, EditClient),
55
56   getRoute(Paths[Views.schedule].path, Schedule),
57   getRoute(Paths[Views.new_appointment].path, NewAppointment),
58
59   getRoute(Paths[Views.accountViewUser].path, AccountUser),
60   getRoute(Paths[Views.profileViewUser].path, ProfileUser),
61
62   getRoute(Paths[Views.privacyPolicyUser].path, PrivacyPolicy),
63 ];
```

Aquí se muestran las rutas del usuario, el primer array define las rutas que no se debe estar autenticado para acceder y el segundo array debe estar autenticado para acceder.

```
[Views.clients]: getPath(`/my-user/clients`, "SideBarClients", "e9ed"),
[Views.new_client]: getPath(`/my-user/client/new`),
[Views.edit_client]: getPath(`/my-user/client/edit/:client_guid`),
```

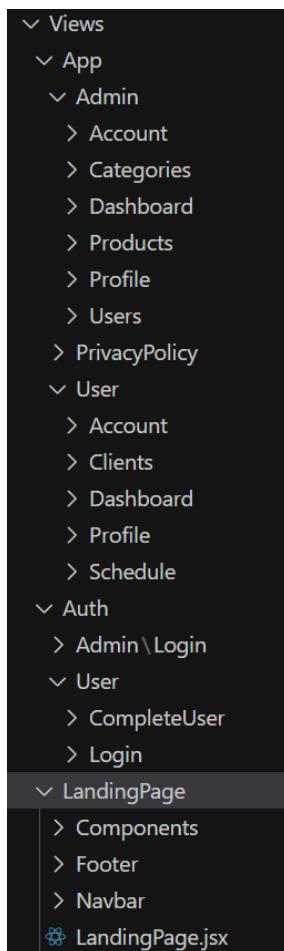
La estructura de las rutas consiste en optimizar por lo tanto las rutas se llaman mediante Paths[Views.nombre].path y a continuación se importaría y se coloca el componente.

->Estructura de carpetas y componentes de la app

La estructura de carpetas del programa tanto de componentes es algo muy importante ya que el programa no exige ningún tipo de estructuración al ser tan anárquico puede todo llegar a ser una bola de código muy difícil de leer, además si no se tiene un orden en proyectos grandes con mucha gente trabajando en un mismo código puede ser inentendible el programa por eso yo he estructurado de esta forma la app:

- **Carpetas**

La estructura de carpeta de la app está organizada de una forma muy visible ya que las vistas o componentes de la app que caen sobre /views se separan primero en tres carpetas dependiendo de la autenticación que necesite el usuario, dentro de estas se separan en el tipo de usuario si es administrador, un usuario médico o un usuario normal sin ningún tipo de autenticación que solo podría acceder a sitios como la landing page.



- Componentes

La estructura interna de los componentes es importante tenerla en cuenta ya que react implementa un formato muy anárquico, es decir, permite realizar la estructura como se quiera por lo tanto si no estas pendiente puede ser un problema.

Los componentes de mediconnect está estructurado de forma que las variables se sitúan al principio del componente, los métodos o funciones debajo de las variables y el renderizado de html dentro del return al final del componente.

```
17  const NewCategory = () => {
18
19    const { strings } = useContext(StringsContext);
20    const ViewStrings = strings.Categories.add;
21
22    const request = useRequest();
23    const { push } = useHistory();
24
25    const [submiting, setSubmiting] = useState(false);
26
27    const { showNotification: successNotification } = useNotification("success");
28    const { showNotification: errorNotification } = useNotification();
29
30    const [data, setData] = useState({});
```

```
31
32    const handleSubmit = () => {
33      if (checkForm()) {
34        setSubmiting(true);
35        request("post", getEndpoint(EndpointsAdmin.Categories.create), { ...data })
36        .then(() => {
37          push(Paths[Views.categories].path);
38          successNotification(ViewStrings.messageSuccess);
39        })
40        .catch(() => errorNotification(ViewStrings.messageError))
41        .finally(() => setSubmiting(false))
42      }
43    }
44
45    const handleInput = (e) => {
46      const { id, value } = e.target;
47      setData({ ...data, [id]: value });
48    }
49
50    const checkForm = () => {
51      const { name, description } = data;
52      return validateData([name, description]);
53    }
54
55    return (
56      <GeneralLayout showBackButton title={ViewStrings.newCategory}>
57        <PanellLayout>
58          <SectionLayout>
59            <FormControl
60              required
61              controlId="name"
62              maxLength={50}
63              showMaxLength={true}
64              vertical={false}
65              value={data.name}
66              title={ViewStrings.name}
67              onChange={handleInput}
```

->Rutas a los endpoints

A la hora de llamar a los endpoints no llamamos directamente a la ruta de la api sino que la app lo tiene centralizado en un archivo llamado **endpoints.constants.js** el cual contiene objetos con los strings que enfocan a los endpoints de la api en el backend, todo esto siempre lo máximo separado posible para que el mantenimiento y la facilidad de entender el código sea mayor.

```
export const EndpointsAdmin = {
  Auth: {
    loginAdmin: `${BASE_URL_ADMIN}/auth/login`,
    get: `${BASE_URL_ADMIN}/auth/get`,
    changeImage: `${BASE_URL_ADMIN}/auth/changeImageProfile`,
    logout: `${BASE_URL_ADMIN}/auth/logout`,
    checkAdmin: `${BASE_URL_ADMIN}/auth/checkAdmin`,
    updateAdminProfile: `${BASE_URL_ADMIN}/auth/updateAdminProfile`,
    updateAdminEmail: `${BASE_URL_ADMIN}/auth/updateAdminEmail`,
    updateAdminPassword: `${BASE_URL_ADMIN}/auth/updateAdminPassword`,
  },

  Categories: {
    getAll: `${BASE_URL_ADMIN}/category/getAll`,
    getList: `${BASE_URL_ADMIN}/category/getList`,
    get: `${BASE_URL_ADMIN}/category/get`,
    create: `${BASE_URL_ADMIN}/category/create`,
    update: `${BASE_URL_ADMIN}/category/update`,
    delete: `${BASE_URL_ADMIN}/category/delete`,
  },
}
```

->Hooks

Los Hooks te permiten utilizar diferentes funciones de react desde los componentes, se podría decir que actúan como si fueran funciones globales. Estos son unos custom Hooks utilizados:

- **useLoaded()**
- **seModalManager()**
- **useNotification()**
- **useQuery()**
- **useRequest()**
- **useSideBar()**
- **useWindowSize()**

->Validacion de formulario

Para validar los formularios realizamos una función **checkForm()** la cual es la encargada con ayuda de otra función, en este caso **validateData()** validar que la información que se va a enviar es correcta y así habilitar el botón de envío o no devolviendo un booleano.

Ejemplo de CheckForm del componente **EditUser.jsx**:

```
const checkForm = () => {
  const { firstName, lastName, specialty, email } = data;
  return validateData([firstName, lastName, specialty, email])
    && EmailRegex.test(email)
    && JSON.stringify(data) !== JSON.stringify(initialData);
}
```

El botón se habilita dependiendo del checkForm:

```
<div className="d-flex justify-content-end align-items-center">
  <Button disabled={!checkForm() || submiting} onClick={handleSubmit}>
    {submiting ? <Spinner size="sm" /> : ViewStrings.buttonUpdate}
  </Button>
</div>
```

Interesante sobre PHP

PHP es uno de los lenguajes más utilizado para programar backend, en mediconnect se ha utilizado este para realizar una rest api y poder conectar el frontend con la base de datos.

Tiene la particularidad de que no existen controladores sino que se llama directamente al fichero y este actúa de endpoint, también se hacen querys a la base de datos directamente con funciones.

->Estructura de carpetas

La estructura de carpeta cuenta con:

-Config: Es la carpeta donde se encuentran los archivos de configuración de la api y archivos que importan todos los objetos y ressources de la api.

-Controller: En esta carpeta se encuentra el archivo que genera el pdf de las recetas, contiene una clase GeneratePDF la cual usa la librería Dompdf.

-Endpoints: Carpeta que contiene todos los endpoints creados para mediconnect organizados por tipo de usuario:

-my-admin: Endpoints que actúan para el administrador.

-my-user: Endpoints que actúan para el usuario médico.

-landing-page: Endpoints que actúan para la landing page sin ningún tipo de autorización.

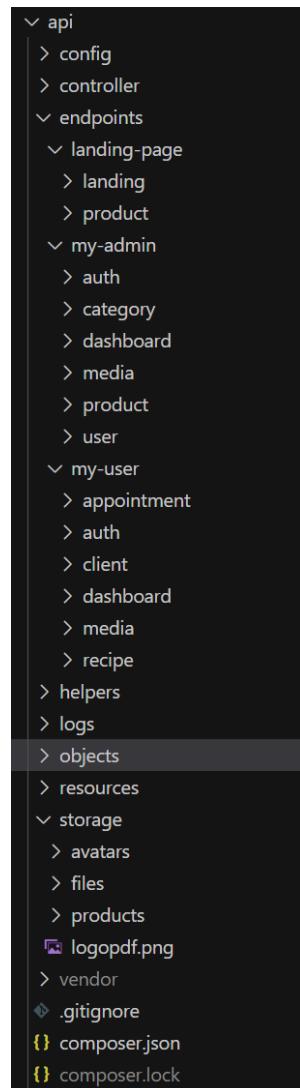
-Helpers: Carpeta que contiene fichero con funciones que realizan acciones como validar información pasada desde el frontend o funciones que interceptan el código de error para **personalizar** el mensaje de error genérico.

-Logs: Carpeta donde se encuentran los archivos .txt de los errores o los logs de la app para poder ver con la función **logApi()** el contenido de variables.

-Objects: Carpeta donde se encuentran los objetos que hacen referencia a la base de datos.

-Resources: Carpeta donde se encuentran los ressources de cada objeto para retornar con estas funciones solo la información necesaria.

-storage: Carpeta donde se guardan la imágenes de la aplicación como los avatares de los usuarios, imágenes de los productos o el logo que se estampa en la cabecera del pdf de las recetas.



->Objects

Todos los objetos de la carpeta objects hacen referencia a una tabla de la base de datos, así se puede conseguir hacer querys a esta y retornar la información como objetos, aquí se muestra el objeto user y como se hace la función que contiene la query a la base de datos para poder retornar un de objetos de tipo user paginado y filtrado gracias a funciones que se encuentran en archivos de la carpeta utils.

```
1 reference | 0 overrides
public static function getAll(PDO $db, int $page, int $offset, string $search = "", array $filters = []): array
{
    $query = "SELECT * FROM `". self::$table_name . "` WHERE deleted_at IS NULL";

    foreach ($filters as $index => $object) {
        $query .= " AND $object->id = :val{$index}";
    }

    applySearchOnQuery(query: &$query);
    doPagination(valuesAmount: $offset, page: $page, query: &$query);

    $stmt = $db->prepare(query: $query);

    applySearchOnBindedValue(search: $search, stmt: &$stmt);

    foreach ($filters as $index => $object) {
        $value = $object->value;
        $stmt->bindValue(param: ":val{$index}", value: $value, type: PDO::PARAM_INT);
    }

    if ($stmt->execute()) {
        $arrayToReturn = [];

        while ($row = $stmt->fetch(mode: PDO::FETCH_ASSOC)) {
            $arrayToReturn[] = self::getMainObject(db: $db, row: $row);
        }
        return $arrayToReturn;
    }
    createException(message: $stmt->errorInfo());
}
```

->Endpoints

Como anteriormente se había mencionado la api no tiene controladores, por lo tanto en el archivo de configuración de la api se define la ruta a la carpeta endpoints y a partir de ahí la ruta hasta el archivo sería el endpoint. Por ejemplo para llamar al endpoint que retorna todos los usuarios paginados y filtrados seria de esta manera:

`https://api.blesadev.es/MediConnect/api/endpoints/my-admin/user/getAll.php`

(con los filtros y paginado necesario al final de la llamada)

-En rojo tenemos la referencia al servidor

-En amarillo la ruta hasta el archivo donde actua el endpoint

```
|?php
include_once '../../../../../config/config.php';

$database = new Database();
$db = $database->getConnection();

$data = getInput();

try {
    $db->beginTransaction();
    checkAuthAdmin();

    $input = validate(data: $data, rules: [
        'page' => 'required|numeric',
        'offset' => 'required|numeric',
        'search' => 'sometimes|string',
        'filter' => 'sometimes|string'
    ]);

    $search = isset($input->search) ? $input->search : '';
    $filter = $input->filter ? json_decode(json: $input->filter) : [];

    $users = User::getAll(db: $db, page: $input->page, offset: $input->offset, search: $input->search, filters: $filter);
    $usersCount = User::getAllCount(db: $db, search: $input->search, filters: $filter);

    foreach ($users as $user) {
        $userProfile = UserProfile::getById(db: $db, userId: $user->id);
        $admin = Admin::get(db: $db, id: $user->created_by);

        $user->creator = $admin->name;
        $user->firstName = $userProfile->first_name;
        $user->secondName = $userProfile->last_name;
        $user->specialty = $userProfile->specialty;
        $user->avatar = $userProfile->avatar;
    }
}

$usersResource = UserResource::getUsersArray(users: $users);
$totalPages = ceil(num: $usersCount / $input->offset);

$db->commit();
Response::sendResponse(data: [
    "users" => $usersResource,
    "totalPages" => $totalPages
]);

} catch (\Exception $th) {
    $db->rollBack();
    print_r(value: json_encode(value: array("status" => false, "message" => $th->getMessage(), 'code' => $th->getCode())));
}
```

Vistas y acciones siguiendo el flujo de la app

Ahora entenderemos como funciona la app, los beneficios que tiene hacia una empresa tanto por la facilidad de gestión para el administrador de esta tanto como el panel de control para que los médicos puedan saber con facilidad citas, clientes, estadísticas...

1. Landing Page

Lo primero que podemos visualizar al entrar a la app es una landing page que da información sobre la clínica, en este caso la clínica que tenemos en producción es la clínica audiológica Mico, esto es editable dependiendo el tipo de clínica que desea usar la app.



2. Admin Portal

El admin portal es el sitio de trabajo del administrador donde puede gestionar categorías para organizar así productos, gestionar productos y también los usuarios médicos que trabajan en la clínica.

a. Login Administrador

Para acceder al login del administrador es tan fácil como escribir en el navegador app.blesadev.es/my-admin esto nos llevaría a la zona de logueo del administrador de la clínica.



b. Inicio Administrador

Una vez se loguea el administrador ya puede acceder a el panel de control donde puedes encontrar gráficas e información sobre la clínica como un apartado de recuperación de productos los cuales se han quedado sin categoría.

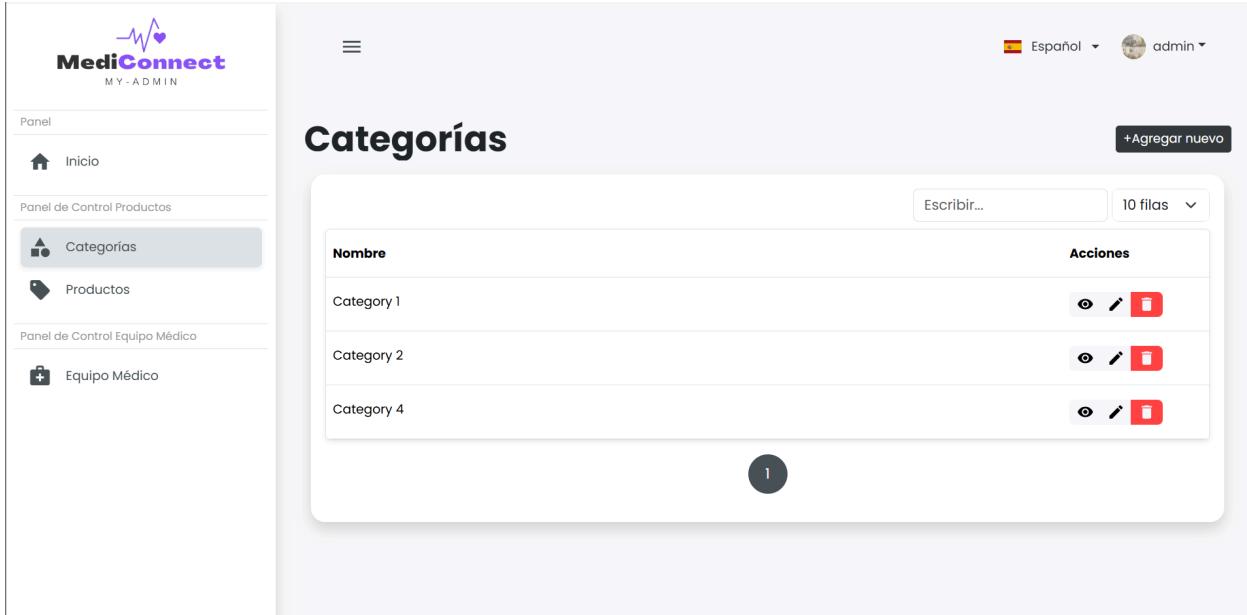
- Total de productos
- Total de categorías
- Total de personal de la clínica
- Productos sin categoría asignada
- porcentaje de productos por categorías

The screenshot shows the MediConnect MY-ADMIN dashboard. On the left, there's a sidebar with navigation links: 'Panel' (selected), 'Inicio', 'Panel de Control Productos' (with 'Categorías' and 'Productos' options), and 'Panel de Control Equipo Médico' (with 'Equipo Médico' option). At the top right, there are language ('Español') and user ('admin') dropdowns. The main area has a title 'Panel'. It features three summary boxes: 'Productos' (5), 'Categorías' (3), and 'Equipo Medico' (3). Below these is a section titled 'Productos sin categorías' with a table showing one item: P3, Marca Audifonex, Precio 2000€. There are 'Escribir...' and '10 filas' input fields. At the bottom is a pie chart titled 'Productos por categoría' showing 80% in Category 2 (red) and 20% in Category 1 (light blue). A legend at the bottom indicates 'Category 1' (light blue) and 'Category 2' (red).

Nombre	Marca	Precio	Acciones
P3	Audifonex	2000€	+ -

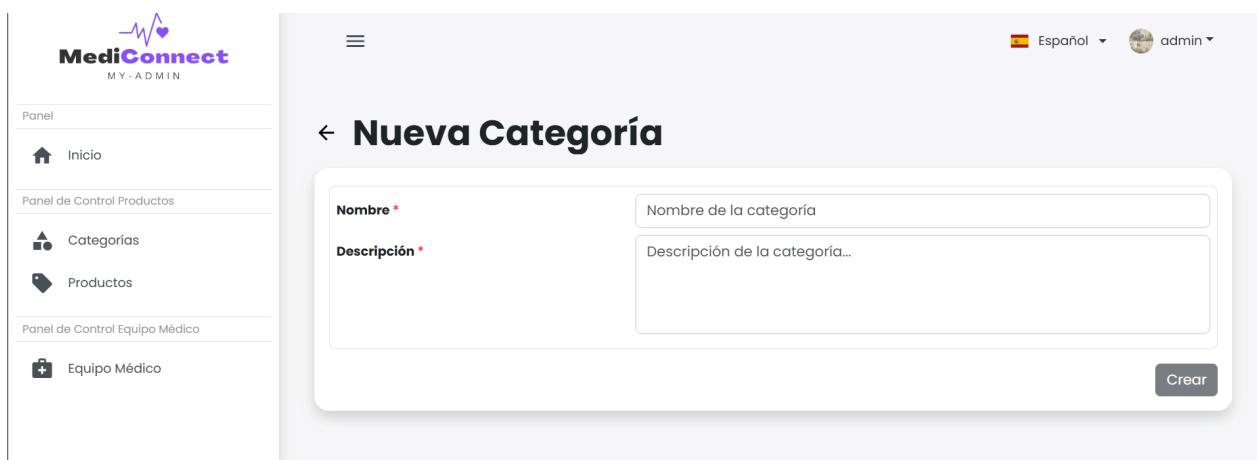
c. Categorías

Siguiendo el flujo de la aplicación nos iremos al apartado de categorías en la cual podemos visualizar todas las categorías creadas además de más acciones.



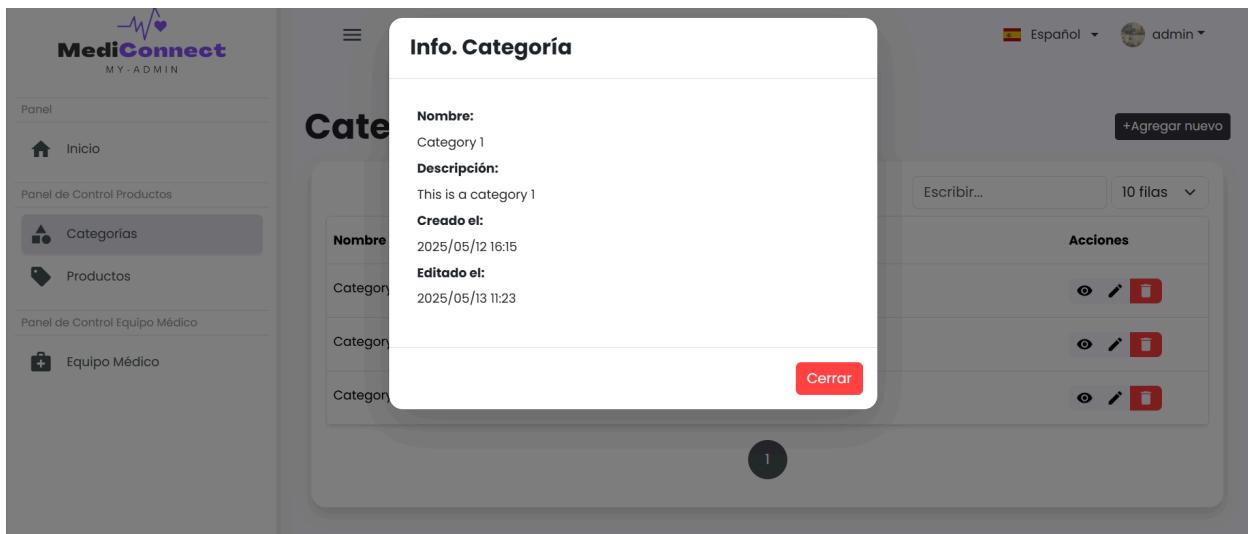
The screenshot shows the MediConnect admin panel. On the left, there's a sidebar with links: Panel (selected), Inicio, Panel de Control Productos (Categories selected), Productos, Panel de Control Equipo Médico, and Equipo Médico. The main area is titled "Categorías". It has a search bar with "Escribir..." and a dropdown for "10 filas". A button "+Agregar nuevo" is at the top right. Below is a table with columns "Nombre" and "Acciones". The table contains four rows: "Category 1", "Category 2", "Category 3", and "Category 4". Each row has three actions: a magnifying glass icon, a pencil icon, and a trash bin icon. A small number "1" is in a circle at the bottom center of the table.

-Crear nuevas categorías pulsando el botón de agregar nuevo.

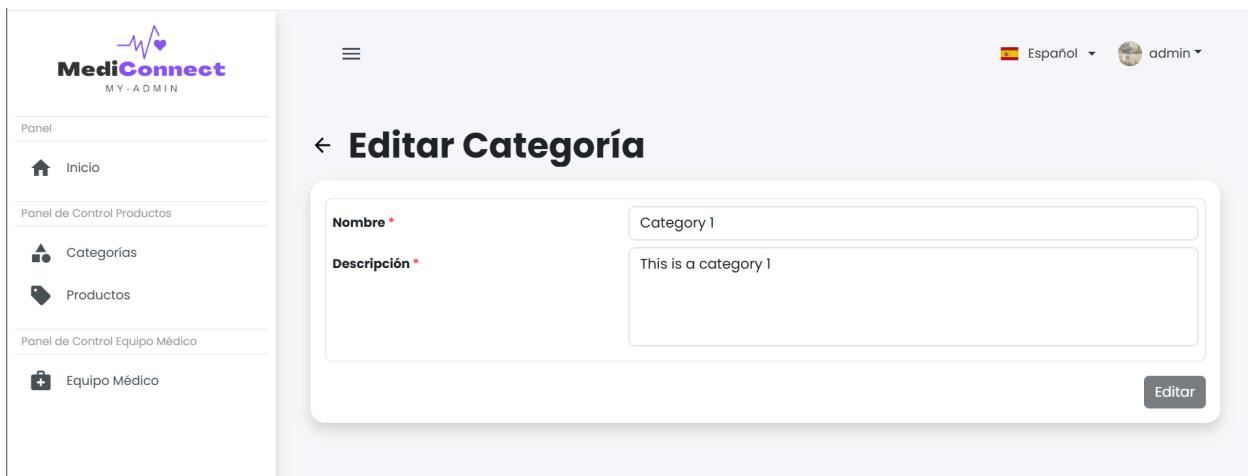


The screenshot shows the MediConnect admin panel. The sidebar is identical to the previous one. The main area is titled "← Nueva Categoría". It has two input fields: "Nombre *" with placeholder "Nombre de la categoría" and "Descripción *" with placeholder "Descripción de la categoría...". At the bottom right is a "Crear" button.

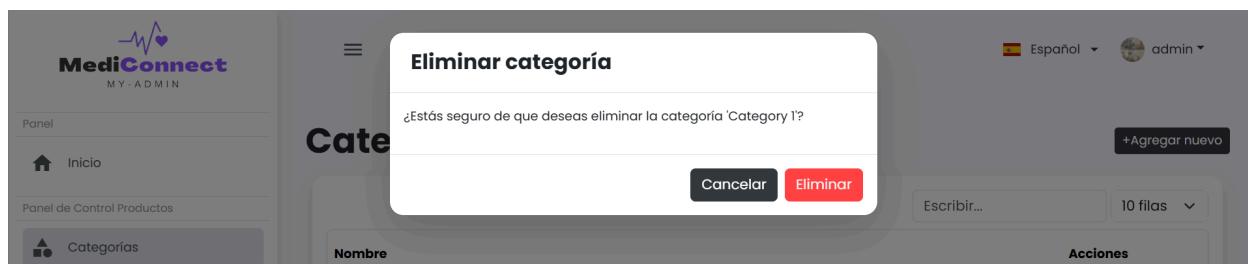
-Visualizar la categoría pulsando el botón con el símbolo del ojo.



-Editar categoría pulsando el símbolo del lápiz.

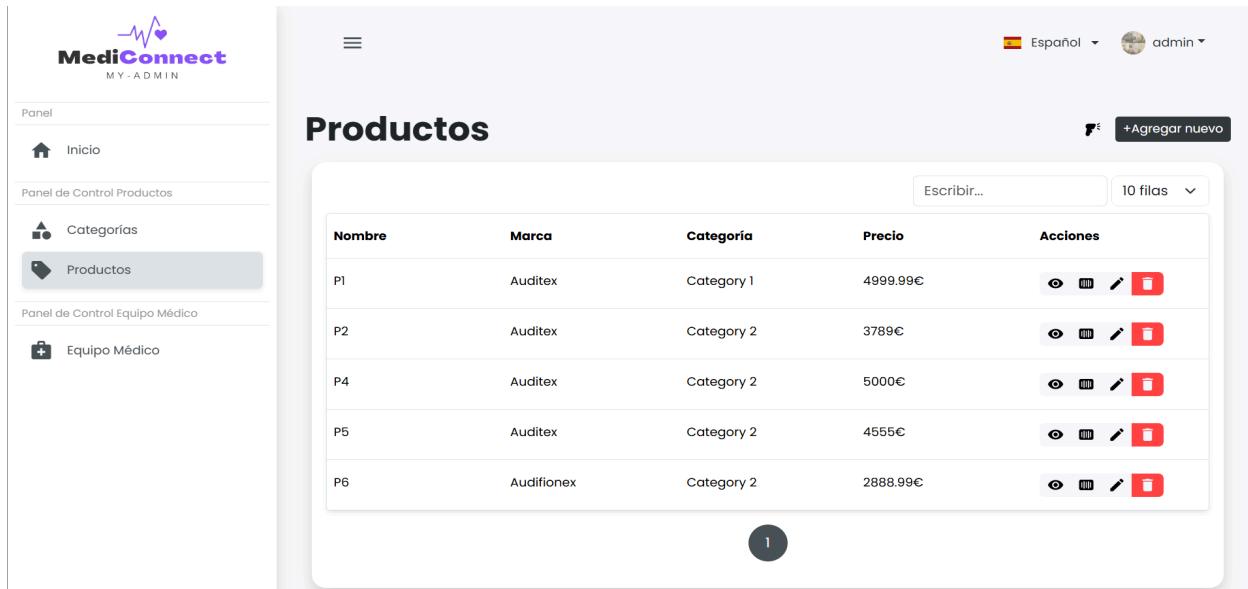


-Eliminar categoría pulsando el símbolo de la papelera.



d. Productos

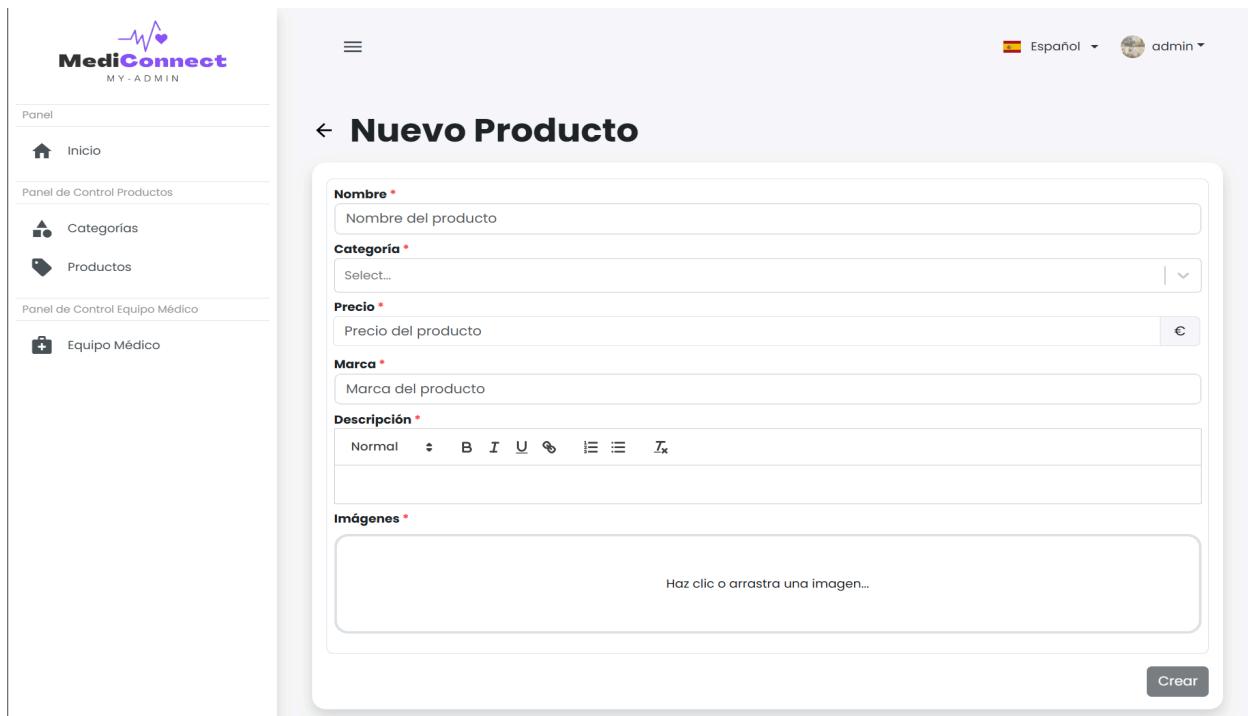
Una vez creadas las categorías podemos dirigirnos al apartado de productos que muestra todos los productos que tiene la clínica, además también se pueden realizar más acciones.



The screenshot shows the MediConnect administrative interface. On the left, there's a sidebar with the MediConnect logo and navigation links: Panel, Inicio, Panel de Control Productos (with Categorías and Products selected), and Panel de Control Equipo Médico (with Equipo Médico). The main area is titled "Productos". It features a search bar with "Escribir..." and a dropdown for "10 filas". A button "+Agregar nuevo" is at the top right. Below is a table with columns: Nombre, Marca, Categoría, Precio, and Acciones. The table contains six rows of data:

Nombre	Marca	Categoría	Precio	Acciones
P1	Auditex	Category 1	4999.99€	[Edit]
P2	Auditex	Category 2	3789€	[Edit]
P4	Auditex	Category 2	5000€	[Edit]
P5	Auditex	Category 2	4555€	[Edit]
P6	Audifonex	Category 2	2888.99€	[Edit]

-Crear nuevos productos pulsando el botón de agregar nuevo.

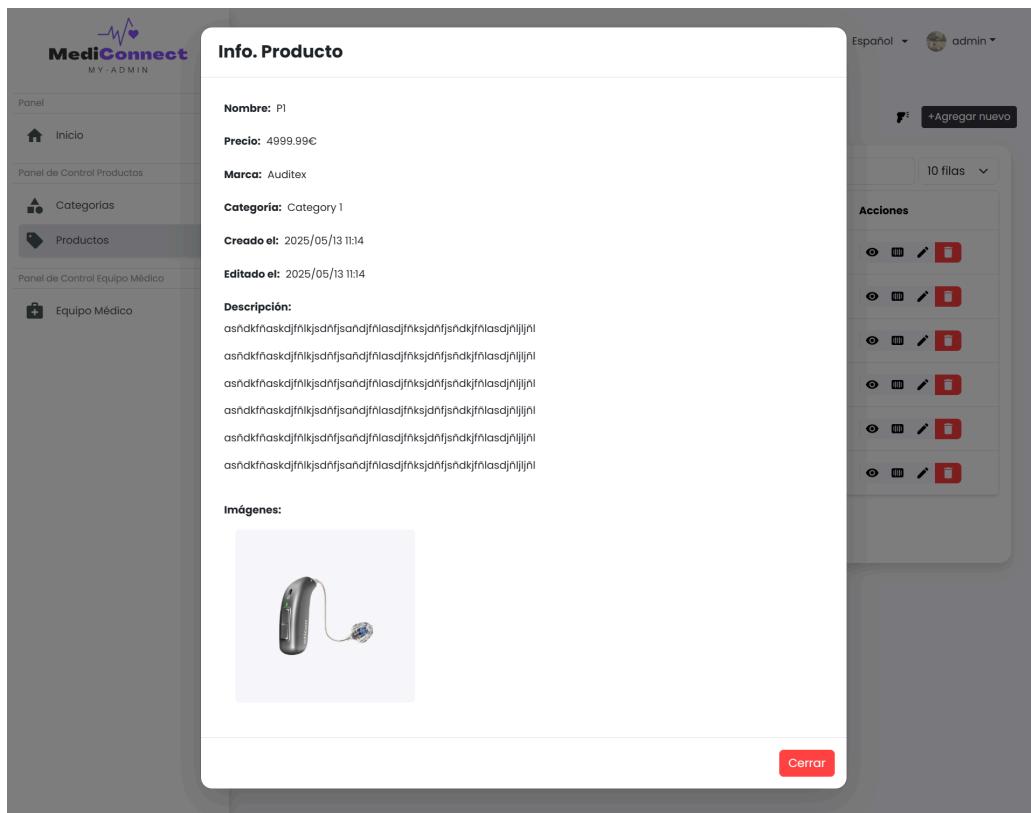


The screenshot shows the MediConnect administrative interface for creating a new product. The sidebar is identical to the previous screenshot. The main area is titled "Nuevo Producto" with a back arrow. It contains several input fields with validation stars (*):

- Nombre ***: A text input field labeled "Nombre del producto".
- Categoría ***: A dropdown menu labeled "Select...".
- Precio ***: A text input field with a currency symbol "€" and a placeholder "Precio del producto".
- Marca ***: A text input field labeled "Marca del producto".
- Descripción ***: A rich text editor toolbar with icons for Normal, Bold, Italic, Underline, etc., followed by a large empty text area.
- Imágenes ***: A file upload area with the placeholder "Haz clic o arrastra una imagen...".

A "Crear" button is located at the bottom right of the form.

-Visualizar el producto pulsando el botón con el símbolo del ojo.



-visualizar el código de barras del producto único para el.



-Editar el producto pulsando el símbolo del lápiz.

The screenshot shows the MediConnect admin interface. On the left is a sidebar with 'Dashboard', 'Home', 'Control Panel Products' (Categories and Products selected), and 'Control Panel Medical Team'. The main area is titled 'Update Product' with a back arrow. It has fields for Name (PI), Category (Category 1), Price (4999,99), Brand (Auditex), and Description (with rich text editor icons). Below is an 'Images' section with a placeholder 'Click or drag an image...', an uploaded image of a hearing aid, and a 'Delete' button. At the bottom right is an 'Update' button.

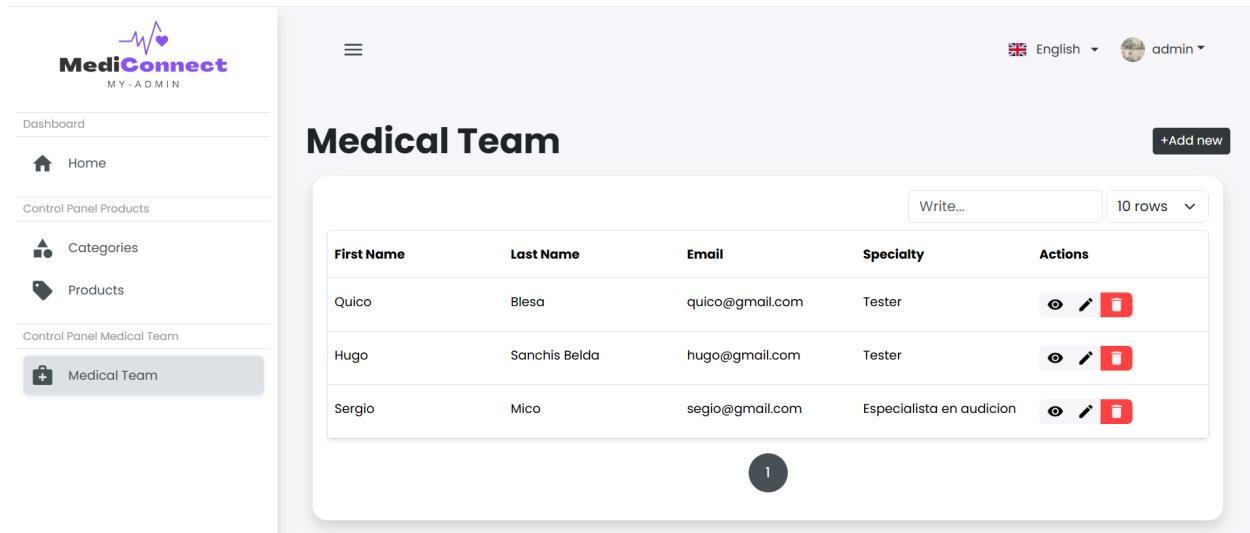
-Eliminar producto pulsando el símbolo de la papelera.

The screenshot shows the MediConnect admin interface. The sidebar includes 'Dashboard', 'Home', 'Control Panel Products' (Categories and Products selected), and 'Control Panel Medical Team'. A modal dialog titled 'Delete Product' asks 'Are you sure you want to delete this product?'. In the background, a table lists products with columns: Name, Brand, Category, Price, and Actions. One row for 'PI' (Brand: Auditex, Category: Category 1, Price: 4999,99) is selected. The 'Actions' column for this row contains icons for view, edit, and delete. The 'Products' tab in the sidebar is highlighted.

Estos dos apartados afectan tanto al backend de la aplicación a la hora de crear recetas y asignar productos a usuarios como a la landing page ya que los productos que se muestran en esta son los creados por el administrador.

e. Usuarios Médicos

Este apartado se centra en poder crear usuarios ya que los trabajadores de la clínica podrán acceder a su portal personalizado son sus clientes/pacientes para así gestionar citas más sencillamente. Inicialmente se muestra a todos los usuarios de la clínica.

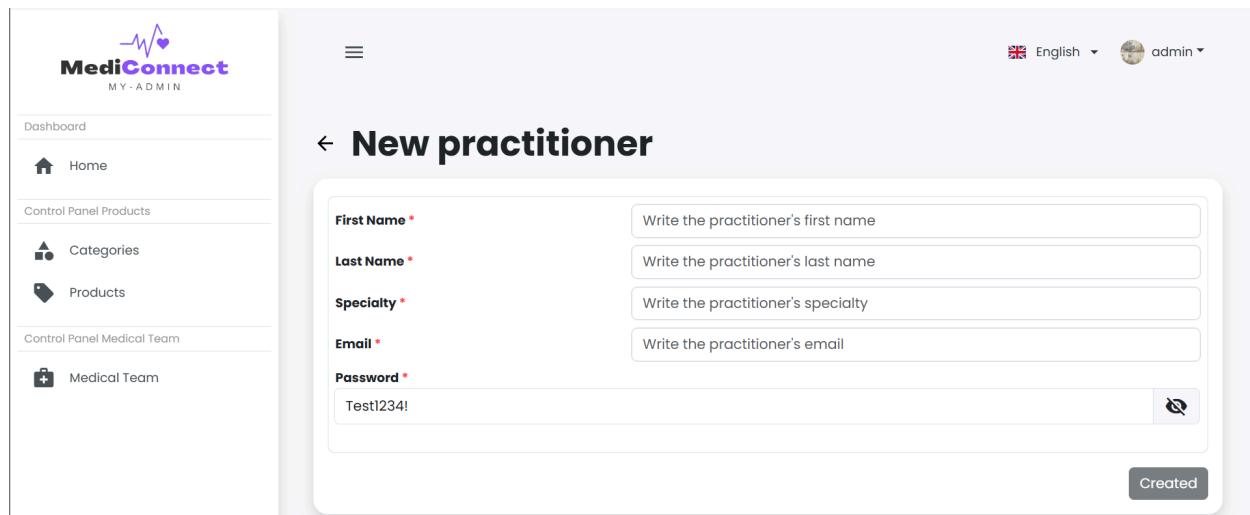


The screenshot shows the MediConnect Admin interface. On the left, there's a sidebar with the MediConnect logo and navigation links: Dashboard, Home, Control Panel Products (Categories and Products), and Control Panel Medical Team (Medical Team). The 'Medical Team' link is highlighted. The main area is titled 'Medical Team' and displays a table of users:

First Name	Last Name	Email	Specialty	Actions
Quico	Blesa	quico@gmail.com	Tester	
Hugo	Sanchis Belda	hugo@gmail.com	Tester	
Sergio	Mico	segio@gmail.com	Especialista en audición	

At the top right, there are language and user selection dropdowns (English, admin). A '+Add new' button is located at the top right of the table area. A page number '1' is at the bottom center.

-Crear nuevos usuarios pulsando el botón de agregar nuevo.



The screenshot shows the MediConnect Admin interface. On the left, there's a sidebar with the MediConnect logo and navigation links: Dashboard, Home, Control Panel Products (Categories and Products), and Control Panel Medical Team (Medical Team). The 'Medical Team' link is highlighted. The main area is titled 'New practitioner' and contains a form with the following fields:

First Name *	Write the practitioner's first name
Last Name *	Write the practitioner's last name
Specialty *	Write the practitioner's specialty
Email *	Write the practitioner's email
Password *	Test1234!

At the bottom right of the form is a 'Created' button.

-Visualizar información sobre el usuario pulsando el botón con el símbolo del ojo.

Practitioner info

Email: quico@gmail.com

First Name: Quico

Last Name: Blesa

Specialty: Tester

Creator: admin

Created at: 2025-05-13 11:24:56

Updated at: 2025-05-14 11:57:25

Close

-Editar el usuario excepto su password por seguridad pulsando el símbolo del lápiz.

← Update Practitioner

First Name * Quico

Last Name * Blesa

Specialty * Tester

Email * quico@gmail.com

Update

-Eliminar un usuario con el botón de la papelera.

Delete Practitioner

Are you sure you want to delete this user?

Cancel Delete

3. User Portal

El User portal es el sitio de trabajo de los usuarios que trabajan en la clínica, estos son creados anteriormente por el administrador de esta y les crea una cuenta para que puedan acceder a su portal personalizado.

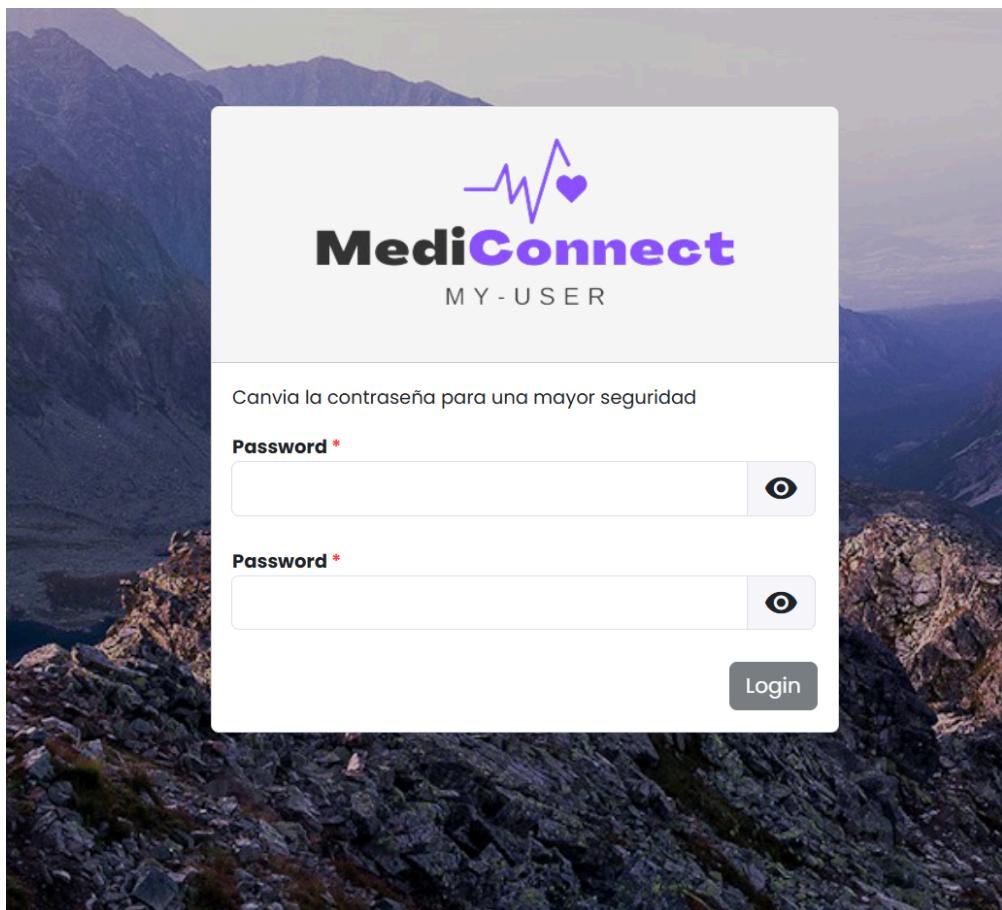
a. Login Usuario

Para acceder al login del usuario se debe pulsar el botón de sign in que se encuentra en el menú de navegación de la landing page, este te lleva a la ruta <https://app.blesadev.es/my-user/login>



b. Complete Login

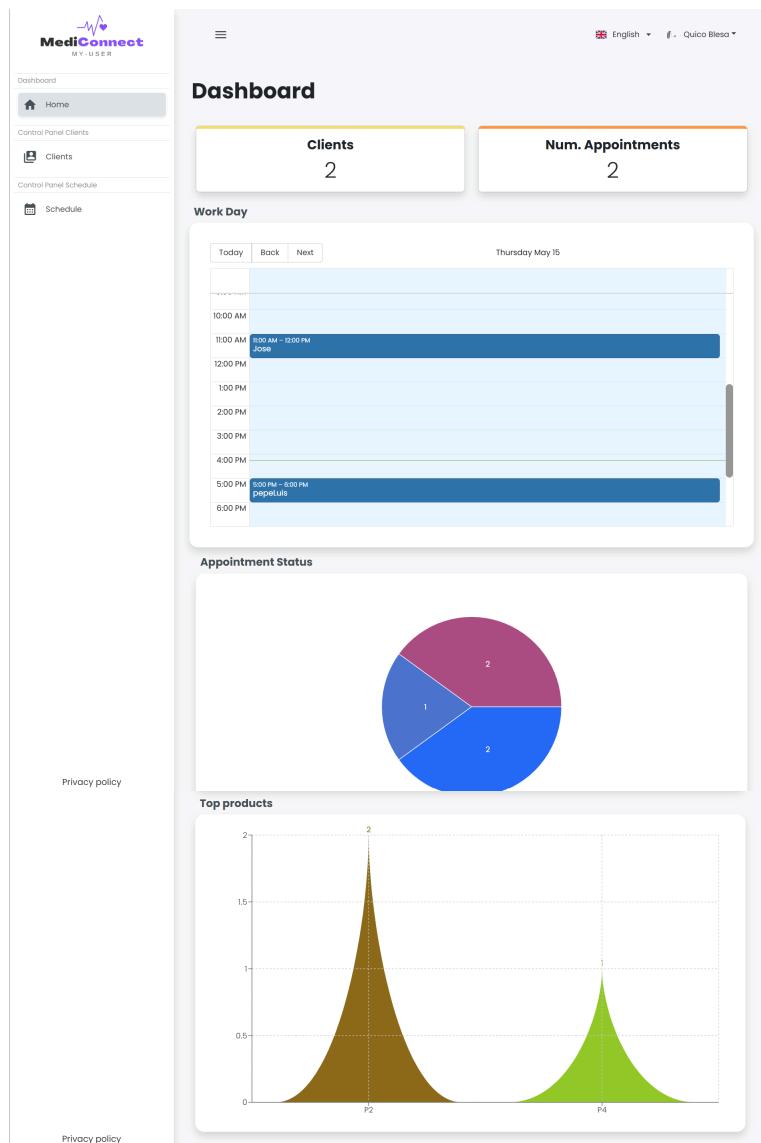
La primera vez que se registra un usuario por seguridad la app pide que se cambie la contraseña ya que cuando un administrador crea el usuario se crea con una contraseña genérica y el portal de un usuario es privado así que solo él debe saber la contraseña.



c. Inicio Usuario

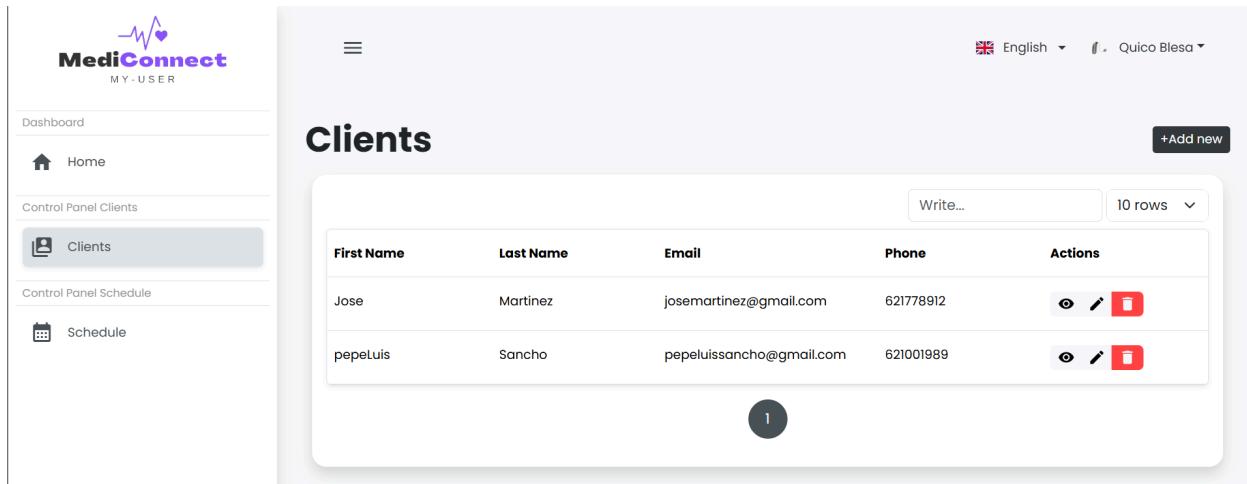
Una vez ya a terminado el cambio de contraseña el usuario puede acceder al panel de control donde se puede ver:

- Total de clientes propio
- Total de citas pendientes
- Espacio de trabajo diario que le indica la hora de la cita y la persona la cual está asignada
- Gráficos de citas pendientes, citas completadas y citas anuladas.
- Una gráfica que le informa de los productos más asignados por los médicos para así poder girarse de cual producto es más popular.



d. Clientes

Apartado en el que los usuarios medico podrán crear a sus clientes para así poder asignarles productos y citas. Inicialmente se muestran todos los clientes del usuario.

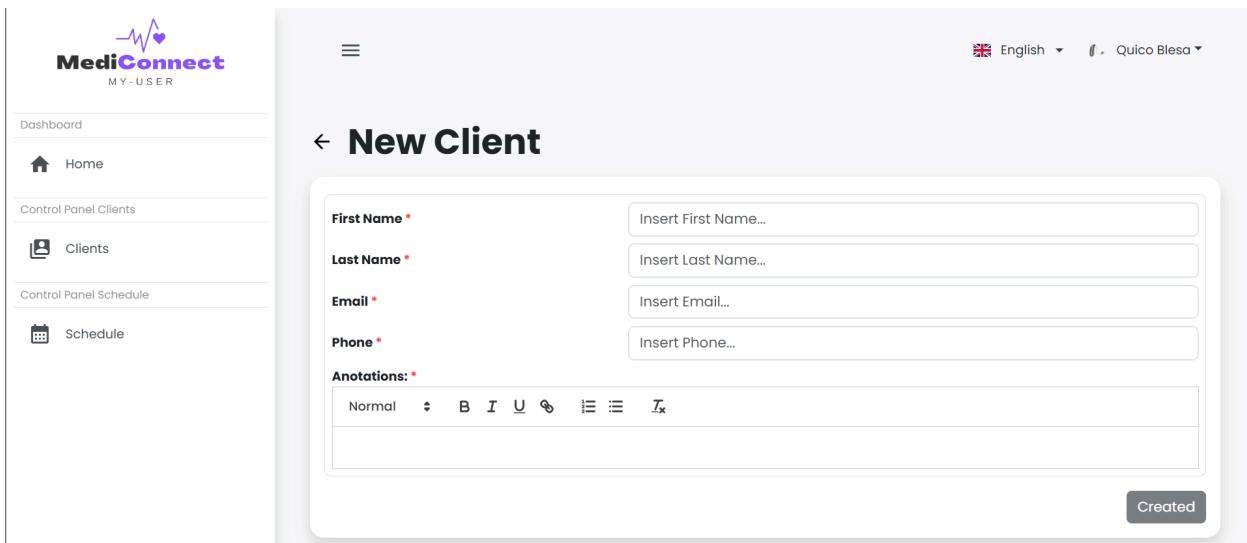


The screenshot shows the MediConnect application interface. On the left is a sidebar with the MediConnect logo and navigation links: Dashboard, Home, Control Panel Clients (Clients is selected), Control Panel Schedule, and Schedule. The main area is titled 'Clients' and contains a table with two rows of data:

First Name	Last Name	Email	Phone	Actions
Jose	Martinez	josemartinez@gmail.com	621778912	
pepeluis	Sancho	pepeluisancho@gmail.com	621001989	

At the top right of the main area are language and user dropdown menus. A button '+Add new' is located at the top right of the table header.

-Crear nuevos Clientes, al crearlos el usuario le hace un estudio inicial genérico y los resultados se apuntan en el campo Annotations, este campo sólo se modifica en el caso de que en futuros estudios el cliente cambiará su estado de salud médica.



The screenshot shows the MediConnect application interface for creating a new client. On the left is a sidebar with the MediConnect logo and navigation links: Dashboard, Home, Control Panel Clients (Clients is selected), Control Panel Schedule, and Schedule. The main area is titled ' New Client' and contains a form with fields for First Name*, Last Name*, Email*, and Phone*. Below these fields is a rich text editor toolbar with options like Normal, B, I, U, etc. An 'Annotations:' section is present with a text area and a toolbar below it. At the bottom right of the form is a 'Created' button.

-Visualizar la información de cada cliente con el botón del ojo.

Client info

First Name: Jose
Last Name: Martinez
Email: josemartinez@gmail.com
Phone: 621778912

Annotions:

- ñkasdñlkfjñasdjñflasjdñfkjasdñdfkjñsdjñfsdajfñskdjjñksdakdñkjsdañfjsadñj
- ñkasdñlkfjñasdjñflasjdñfkjasdñdfkjñsdjñfsdajfñskdjjñksdakdñkjsdañfjsadñj
- ñkasdñlkfjñasdjñflasjdñfkjasdñdfkjñsdjñfsdajfñskdjjñksdakdñkjsdañfjsadñj

Doctor Info:
First Name: Quico **Last Name:** Blesa

Actions

+Add new

10 rows

Close

-Editar la información del cliente en el botón con el icono del lápiz.

← Update Client

First Name * Jose
Last Name * Martinez
Email * josemartinez@gmail.com
Phone * 621778912

Annotions: *

- ñkasdñlkfjñasdjñflasjdñfkjasdñdfkjñsdjñfsdajfñskdjjñksdakdñkjsdañfjsadñj
- ñkasdñlkfjñasdjñflasjdñfkjasdñdfkjñsdjñfsdajfñskdjjñksdakdñkjsdañfjsadñj
- ñkasdñlkfjñasdjñflasjdñfkjasdñdfkjñsdjñfsdajfñskdjjñksdakdñkjsdañfjsadñj

Update

-Borrar un cliente en el icono de la papelera.

Delete Client

Are you sure you want to delete this client?

Actions

Cancel Delete

e. Calendario

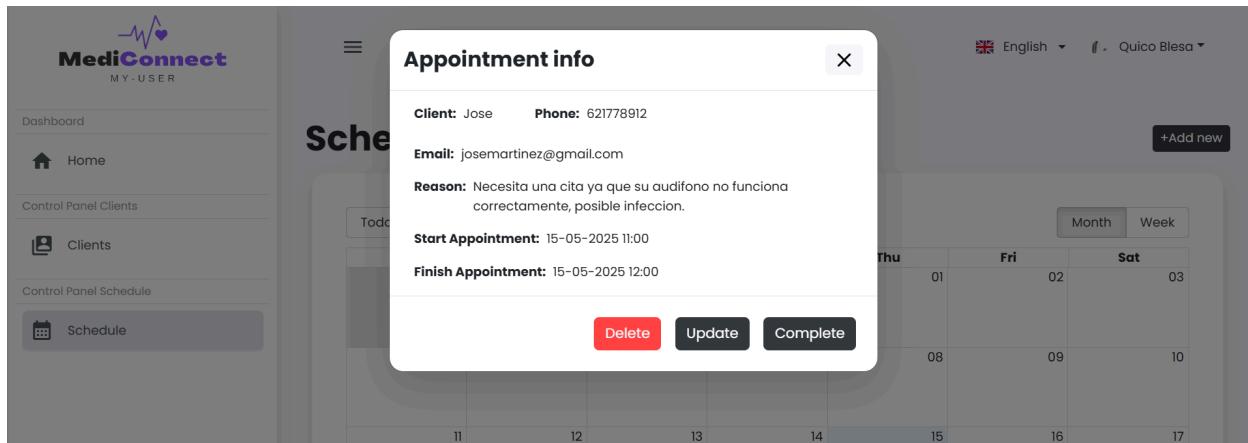
Este apartado inicialmente se visualiza un calendario con las citas por meses o semanas, también tiene más funcionalidad como crear citas, visualizar la información de la cita, editar la cita, cancelarla y también completar la cita, lo que genera una receta en formato pdf.

The screenshot shows the MediConnect application's interface. On the left, there's a sidebar with navigation links: Dashboard, Home, Control Panel Clients (Clients), and Control Panel Schedule (Schedule, which is currently selected). The main content area is titled "Schedule". At the top right of the main area, there are language and user selection buttons: English (with a dropdown arrow) and Quico Blesa (with a dropdown arrow). Below these are buttons for "+Add new" and "Month" (which is currently selected) and "Week". The main feature is a monthly calendar for May 2025. The days of the week are labeled from Sunday to Saturday. Specific dates are marked: May 15 is highlighted with a blue box containing the names "Jose" and "pepeluis". Other dates like 27, 28, 29, 30, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31 are also visible.

-Para crear una cita se pulsa en el botón de añadir nueva.

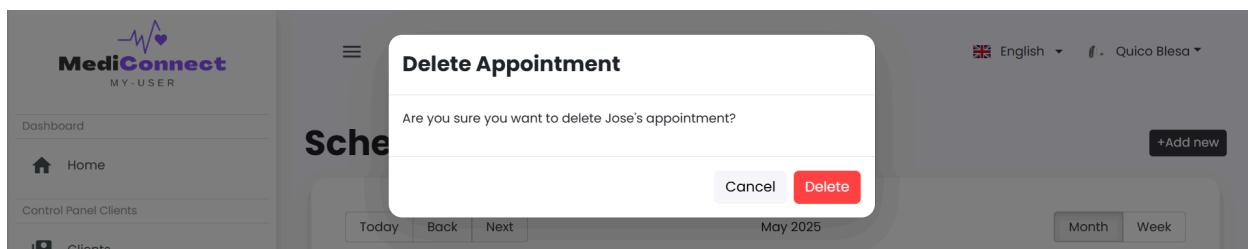
The screenshot shows the MediConnect application's interface for creating a new appointment. On the left, there's a sidebar with navigation links: Dashboard, Home, Clients, and Schedule. The main content area is titled "← New Appointment". It contains several input fields: "Client *" (a dropdown menu), "Reason *" (a text input field with placeholder "Insert Reason..."), "Date: *" (a date input field with a calendar icon), and "Hours *" (a dropdown menu with placeholder "Select..."). At the bottom right of the form is a "Create" button.

-Visualizar información de la cita y diferentes funcionalidades posibles para esta.

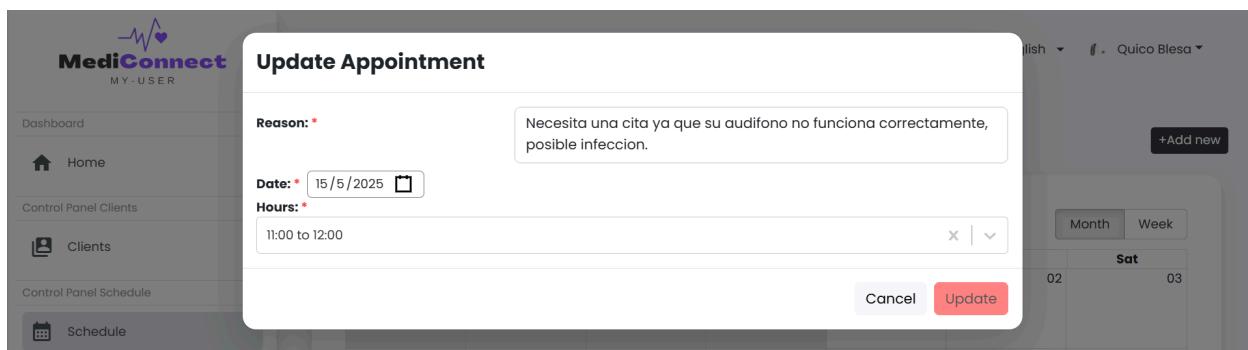


Dentro de este modo se pueden realizar estas funciones:

-borrar la cita que sería equivalente a cancelar.



-editar la cita tanto en razón como fecha como hora.



-Completar la cita significa que el cliente ya ha recibido su cita y se le ha dado una solución al problema que tenía.



-Esto generará un PDF con la descripción escrita por el médico y los productos recomendados por él.





Recomendados Products		
Name product	Images	BarCode
P2		
P4		

Mediconnect® - Documento generado el 15/05/2025 | Página 2

CONCLUSIÓN

RECURSOS

-Visual Studio Code:

Editor de código liviano y gratuito que permite escribir, depurar y administrar proyectos de programación con soporte para múltiples lenguajes.

-HeidiSQL:

Herramienta visual para gestionar bases de datos MySQL, MariaDB y SQL Server. Permite ejecutar consultas, ver tablas y editar datos fácilmente.

-Node.js:

Entorno que permite ejecutar JavaScript en el servidor. Se usa para crear aplicaciones web, APIs y herramientas de desarrollo.

-GitHub:

Plataforma para almacenar y compartir código usando control de versiones con Git. Facilita la colaboración entre desarrolladores.

-Azure:

Plataforma en la nube de Microsoft que permite alojar aplicaciones, bases de datos y servicios con alta disponibilidad y escalabilidad.

BIBLIOGRAFÍA

Esto son unos un poco de los sitios los cuales me han ayudado en el proceso de realización del proyecto tanto en información como en recursos:

- Documentación oficial de react.
<https://react.dev>
- Foro stack overflow
<https://es.stackoverflow.com>
- Documentación de react-bootstrap.
<https://react-bootstrap.netlify.app>
- iconos bootstrap
<https://icons.getbootstrap.com>
- Google icons
<https://fonts.google.com/icons>
- git hub
<https://github.com>