NutriGestión: Aplicación de gestión de pacientes en el área de la nutrición

Trabajo de fin de grado en ingeniería Informática, facultad de informática
Universidad Complutense de Madrid

Autor: Iván Canas Ramos

Director: Antonio Sarasa Cabezuelo

Madrid, 22 de abril de 2019

Índice

Índice de Figuras

Resumen

En este trabajo se ha desarrollado una aplicación que ayuda a optimizar el desempeño de la actividad laboral de los nutricionistas. En este sentido, permite centralizar toda información que se obtiene en una consulta en un mismo lugar, gestionar la información de los pacientes como las diferentes patologías o datos antropométricos, realizar cálculos sobre los datos de los pacientes para generar informes con los resultados obtenidos o elaborar dietas a partir de plantillas donde aparecen los alimentos y sus cantidades distribuidos en diferentes franjas horarias de la semana. Por su parte, los pacientes pueden visualizar su progreso en forma de gráficas, su dieta asignada, su próxima cita o redactar un correo electrónico a su nutricionista.

Palabras Clave: Nutricionista, paciente, datos antropométricos, patología, dieta, cita.

Abstract

In this work, an application has been developed that helps to optimize the performance of the work activity of the nutritionists. The application allows the nutritionist to centralize all the information that it obtains from its nutrition clinic in one place, manage patients information such as different pathologies or anthropometric data, perform calculations on patient data to generate reports with the results obtained, or create diets from templates and assign them to their patients where food and their quantities can be added and distributed in different days slots. By the other hand, patients can visualize their progress in the form of graphs, their assigned diet, their next appointment or write an email to their nutritionist.

Keywords: Nutritionist, patient, anthropometric data, pathologies, diet, appointment.

1.A. Introducción

En este capítulo se van a describir la motivación y objetivos del trabajo desarrollado, así como la estructura que presenta la memoria.

1.1. Motivación

El nutricionista es el profesional sanitario encargado de la nutrición de los seres humanos y para llevar a cabo su labor profesional debe conocer, almacenar e interpretar los datos de un paciente. Los datos más importantes de un paciente son sus medidas antropométricas puesto que a partir de ellas se pueden obtener métricas que proporciona información acerca del estado de salud de un paciente. A partir de las medidas, métricas y patologías de un paciente, se elaboran las dietas. En muchos casos, esta información se encuentra dispersa en diferentes medios tales como folios, hojas de Excel o documentos de Word.

En este sentido, para agilizar la labor de un nutricionista es interesante disponer de una herramienta informática que le permitiera gestionar de manera integral toda la información de los pacientes, realizar los cálculos sobre los datos antropométricos, elaborar informes o preparar tratamientos para los pacientes en base a los datos y calculados realizados.

1.2. Objetivos

El desarrollo de la aplicación tiene como objetivo crear una herramienta informática que facilite el trabajo que se realiza en una consulta de nutrición. Este objetivo general se puede especificar mediante los siguientes objetivos más específicos:

- Facilitar la extracción y almacenamiento en un historial todos los datos relevantes de un paciente (datos antropométricos, patologías, cálculos de métricas…)
- Capacidad para realizar cálculos sobre los datos de un paciente.
- Facilitar la visualización de los datos y cálculos realizados a través de gráficas.
- Capacidad de realizar, editar y visualizar dietas de una forma sencilla para un paciente.
- Gestión de citas de pacientes.
- Facilitar a los pacientes el acceso a su información personal.

1.3. Estructura de la memoria

A continuación, se explicará la estructura de la memoria y los diferentes capítulos que contiene junto a una pequeña descripción de los mismos.

- **Introducción**
  En este capítulo se describe las razones para llevar a cabo el desarrollo de una herramienta como la desarrolla en este trabajo, los objetivos planteados en el trabajo, y la estructura de la memoria.
- **Estado del arte**
  En este capítulo se revisan algunas herramientas que disponen de funcionalidades parecidas a las implementadas en esta aplicación.

- **Tecnología empleada**
  En este capítulo se describen las herramientas tecnológicas empleadas para el desarrollo de la aplicación.
- **Casos de uso**
  En este capítulo se describen los casos de uso definidos para la aplicación.
- **Modelo de datos**
  En este capítulo se describe el modelo de datos definido para realizar la persistencia de la información, así como la base de datos implementada.
- **Arquitectura**
  En este capítulo se describe la arquitectura de software que se ha implementado en la aplicación desarrollada.
- **Diseño de la aplicación**
  En este capítulo se explica las funcionalidades implementadas, así como todos los detalles acerca del diseño de la aplicación.
- **Evaluación**
  En este capítulo se describe la evaluación realizada, así como los resultados de la misma.
- **Conclusiones y trabajo futuro**
  En este capítulo se propone un conjunto de conclusiones acerca del trabajo realizado y se plantean algunas líneas de trabajo futuro.
- **Bibliografía**
- **Anexos**
  En este capítulo se describen tres anexos. Estos anexos incluyen una guía de uso de la aplicación donde se muestra cómo utilizar la herramienta de una forma guiada, una guía de instalación de la aplicación donde se explica cómo llevar a cabo la instalación de la aplicación, y un anexo donde se describen las fórmulas matemáticas utilizadas para el cálculo de las métricas.

1.B. Introduction

This chapter will describe the objectives and motivation of the project developed, as well as the structure of the present document.

1.1. Motivation

The nutritionist is the health care professional in charge of the human being's nutrition, and to carry out its professional tasks, the nutritionist must know, store and interpret the patient's data. The patient most important data are its anthropometric measures as through them, nutritionists can obtain metrics that provide information about the health status of a patient. Diets are made from the measurements, metrics and pathologies of a patient. In many cases, this information is scattered in different media such as sheets of papers, Excel, or Word documents.

In this way, to expedite the nutritionist work, it is interesting to have a computer tool that allows the professional to manage all patient's information, make calculations on the anthropometric data, elaborate reports, or prepare patients treatments based on collected and calculated data.

1.2. Objectives

The development of this application aims to create a computer tool that facilitates the work that is done in a nutrition office. The development of the application can be specified by the following objectives:

- Make easier the extraction and storage in a historic all the relevant data of a patient (anthropometric data, pathologies, metrics...)
- Ability to perform calculations on a patient's data.
- Make easier the visualization of data and calculations through graphs.
- Ability to create, edit and visualize diets in a simple way.
- Patient appointment management.
- Provide patients access to personal information.

1.3. Structure of the document

It will explain the structure of the document and all the different chapters that it contains including a small description of them.

- **Introduction**
  This chapter describes the reasons to develop a tool like the one developed in this project, the objectives set in the development of the application, and the structure of the document.
- **State of the art**
  In this chapter are going to be review some tools that have similar functionality to those implemented in this application.
- **Used technology**
  This chapter describes the tools used for the development of the application.

- **Use cases**
  This chapter describes the use cases defined for the application.
- **Data structure**
  This chapter describes the data structure defined to perform the persistence of information, as well as the database implementation.
- **Architecture**
  This chapter describes the software architecture that has been deployed in the developed application.
- **Application Design**
  This chapter explains the implemented features as well as all the details about the application design.
- **Evaluation**
  This chapter describes the assessment document and the results of these assessment.
- **Conclusions and future work**
  This chapter proposes a set of conclusions about the project and some future work lines are proposed.
- **Bibliography**
- **Annex**
  Three annexes are described in this chapter. These three annexes include a user's guide which shows how to use the application in a guided way, a installation guide explaining how to install the application, and an annex that describes the mathematical formulas used for calculating metrics.

2. Estado del Arte

En este capítulo se revisan algunas herramientas que disponen de una funcionalidad parecida a la implementada en el trabajo desarrollado. Se describirán sus principales características y aplicaciones.

2.1. nutrium.io

nutrium.io[1] es una aplicación web encargada de gestionar pacientes, citas, dietas, recetas, alimentos, tablas de equivalencias y comunicación mediante chat en línea entre el paciente y el profesional.

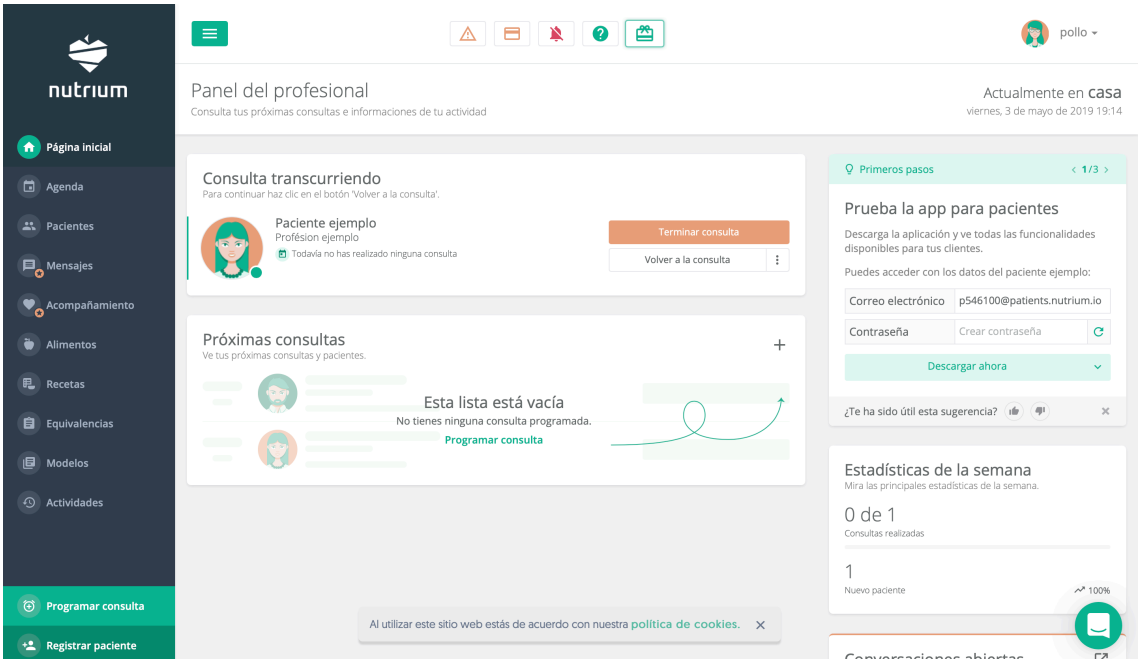


Figura 1 - nutrium.io principal

Como se observa en la *Figura 1* nutrium.io proporciona una interfaz amigable que permite encontrar las funcionalidades de una forma sencilla. Además, posee una base de datos con una gran cantidad de alimentos ya incluidos donde todas las propiedades de cada alimento se encuentran desglosadas en vitaminas, grasas, proteínas, calorías, colesterol, etc. Por otro lado, tal y como se muestra en la *Figura 2*, la aplicación permite añadir diferentes tipos de datos a la ficha del paciente tal como la calidad del sueño, su función intestinal, medicación, ingesta de alcohol y otros hábitos, así como antecedentes familiares, raza y estado civil.

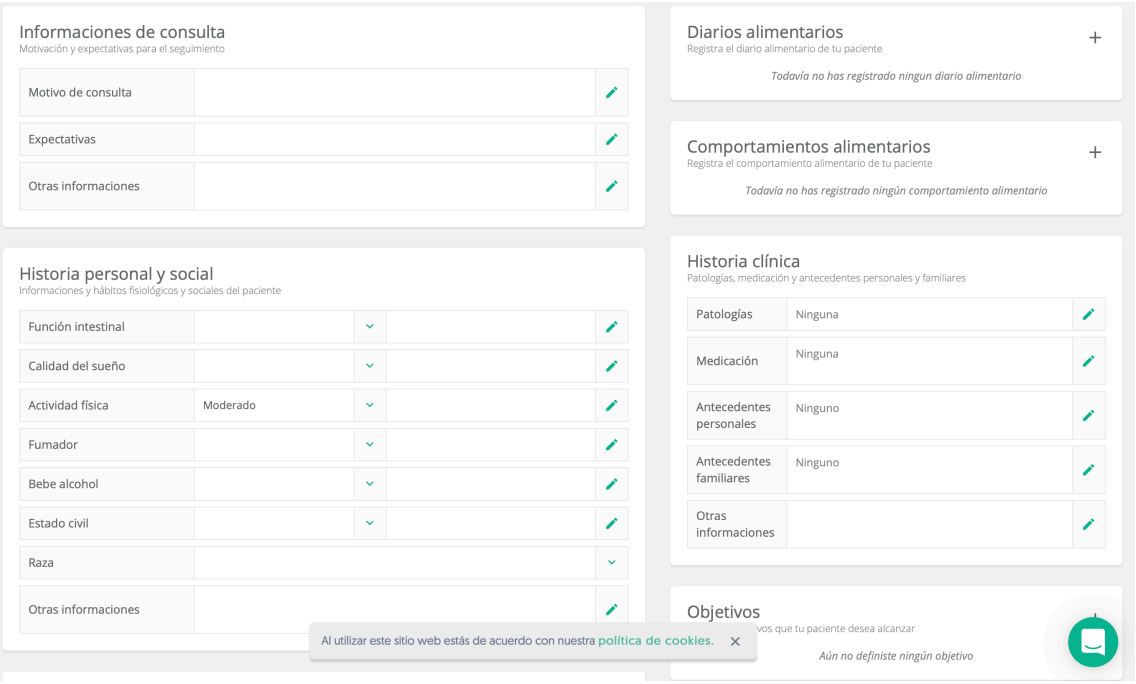


Figura 2- Representación ficha de paciente

En la *Figura 3*, se observa la sección de recetas de nutrium.io. Esta funcionalidad permite añadir alimentos a la base de datos así como los valores nutricionales de los mismos. Una vez añadidos los alimentos, se pueden combinar para crear recetas. La funcionalidad tiene el inconveniente de tener que introducir todos los alimentos de cada receta y sus valores nutricionales manualmente. En la base de datos ya se encuentran incluidos algunos alimentos completos con sus valores nutricionales. Su principal desventaja es la necesaria comprobación de la veracidad de los datos que contienen los alimentos almacenados en la base de datos.

El precio de nutrium.io es de 50€ mensuales para un máximo de 10 pacientes, 68€ para un máximo de 25 pacientes, y 104€ mensuales para pacientes ilimitados.

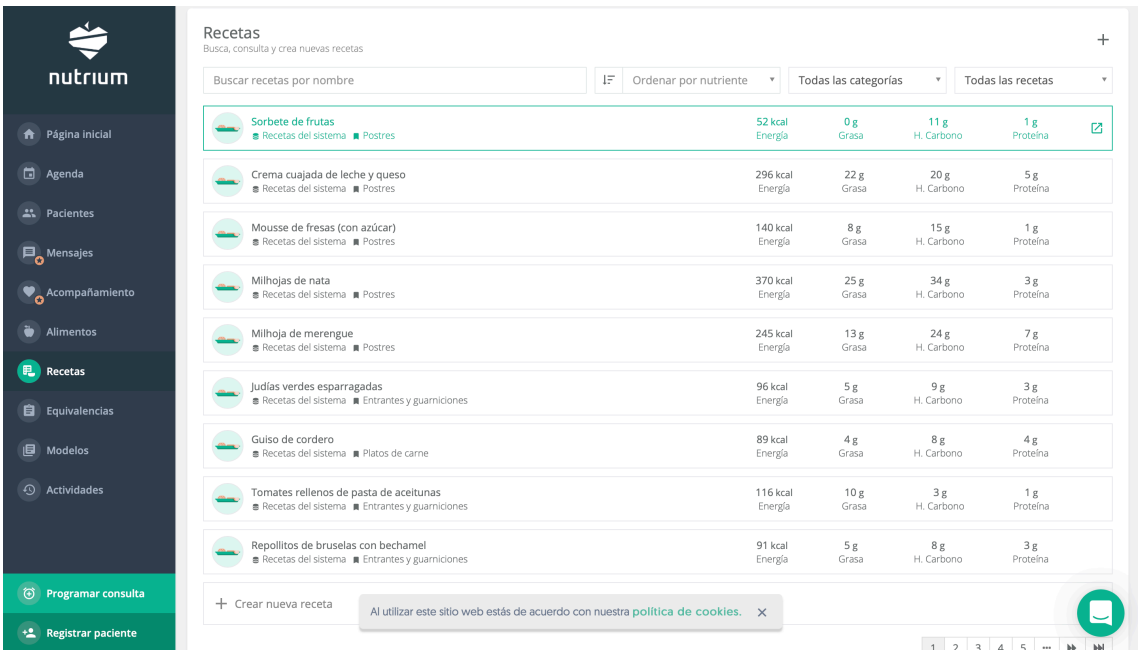


Figura 3 - Recetas nutrium.io

2.2. nutritioapp

nutritioapp[2] cuenta con una interfaz sencilla e intuitiva (*Figura 4)*. En este sentido, presenta unas funcionalidades muy específicas tales como crear múltiples pacientes y asignarles una cita, añadirle datos médicos, medidas y dietas.

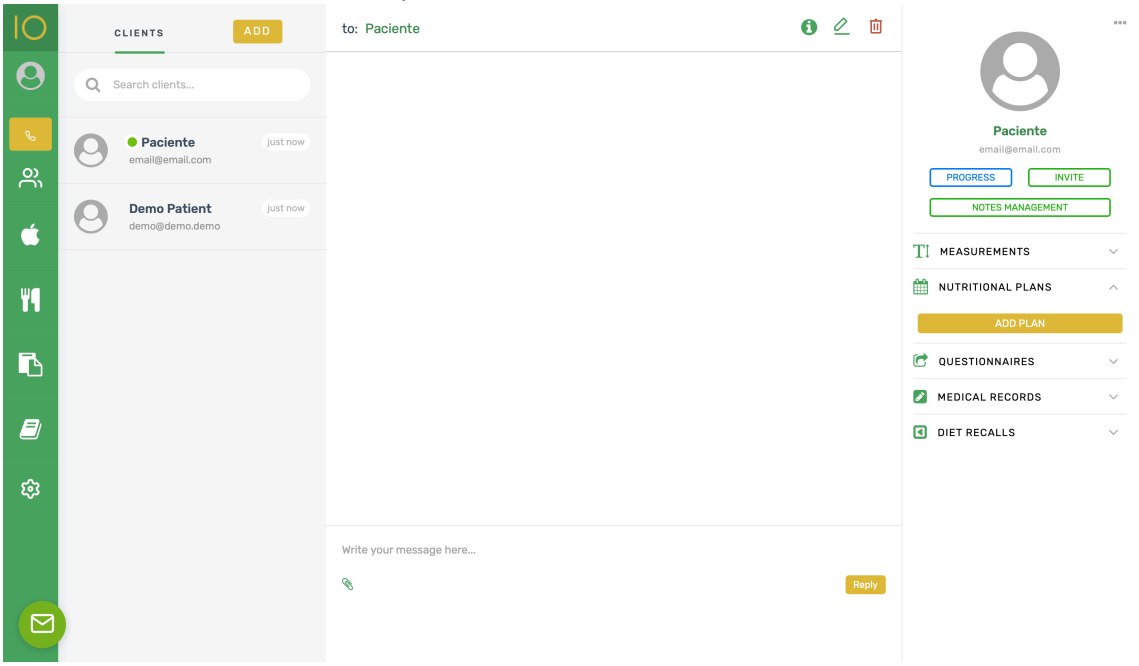


Figura 4 - Principal nutritioapp

Entre sus funciones se encuentra la posibilidad de introducir alguna de las medidas del paciente. Sin embargo, realizar un seguimiento gráfico de la evolución de un paciente, puesto que sólo dispone de un histórico de las medidas numéricas que se han introducido (*Figura 5)*. Con respecto a la explotación de los datos, la aplicación facilita el índice de masa corporal, su índice metabólico y su ingesta de calorías diarias recomendadas.

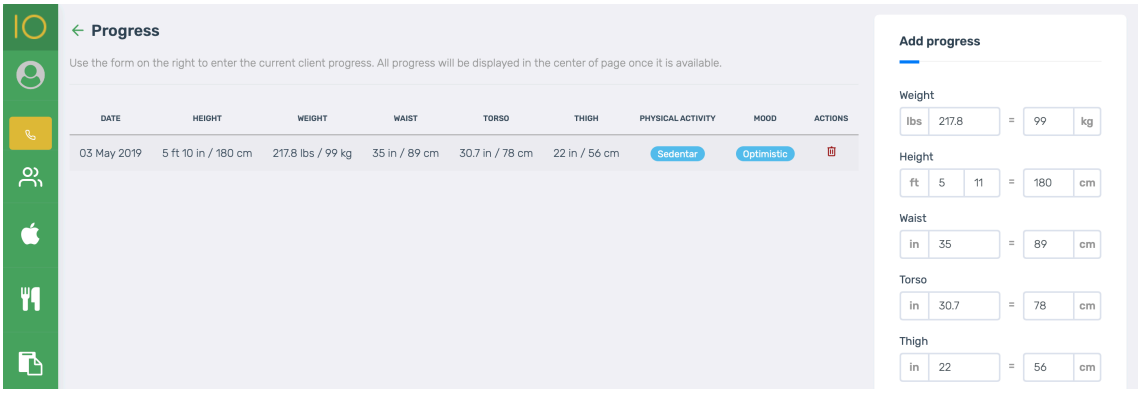


Figura 5 - nutritioapp medidas e histórico

La aplicación, no dispone de una base de datos de alimentos inicial, pero permite introducir alimentos a su base de datos y rellenar todos los valores nutricionales. Los alimentos introducidos, pueden ser utilizados posteriormente para realizar una dieta a cada paciente. Otra de las funcionalidades que incluye esta aplicación es un chat en directo entre el nutricionista y el paciente.

El precio de nutritioapp es de 44€ al mes con máximo de 25 pacientes, 74€ al mes con máximo de 75 pacientes, no siendo posible añadir ni un paciente más. Su página web, no indica precio para más pacientes, lo incluye en su plan de clínica del cual no ofrecen información de precios.

2.3. Hojas de Microsoft Excel

En la actualidad, muchos profesionales de la nutrición emplean hojas de cálculo para llevar un seguimiento de sus pacientes utilizándose también a modo de base de datos para almacenar los pacientes y sus medidas. Normalmente contienen información antropométrica y datos básicos del paciente pero no almacenan dietas, ni recetas, ni próximas citas (*Figura 6*). Así con los datos de los pacientes se pueden realizar cálculos, representarlos gráficamente, y añadir o modificar diferentes métricas sobre los datos introducidos.

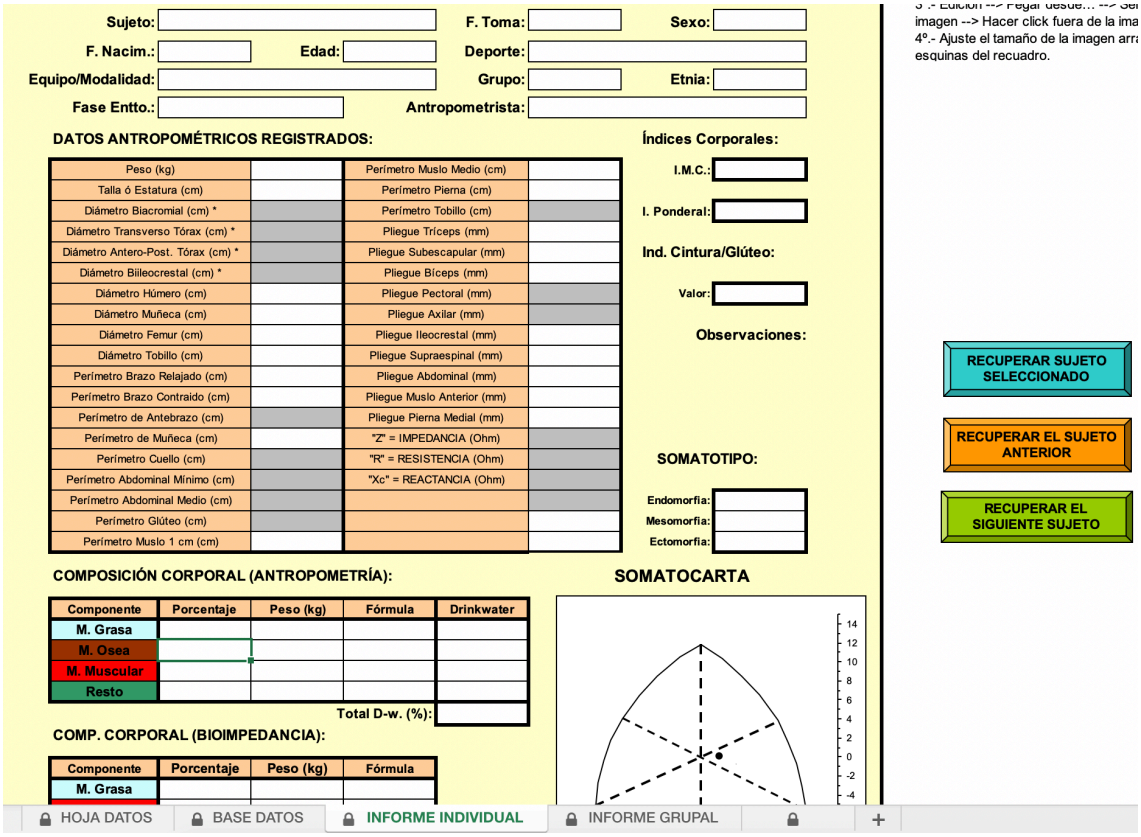


Figura 6 - Hoja Excel I

3. Tecnología Empleada

En este capítulo se describen las herramientas tecnológicas empleadas para el desarrollo de la aplicación. Se trata de una aplicación web, y por ello se va a agrupar la presentación de las mismas en herramientas de la parte del cliente (Front-end) y herramientas de la parte del servidor (Back-end).

3.1. Herramientas de la parte del cliente (Front-end)

3.1.1. Angular

Angular[1] es un framework de desarrollo de aplicaciones web, desarrollado y mantenido por Google, cuya finalidad es el desarrollo de aplicaciones web SPA[1] (son aplicaciones web que constan de una sola página, donde la navegación y las interacciones se realizan sobre dicha página, no siendo necesario recargar la página en cada cambio de sección puesto que las acciones, son dinámicas, asíncronas, reactivas e instantáneas).
El framework sigue una filosofía de desarrollo por componentes. Cada componente es independiente de los demás y tienen asignada una funcionalidad concreta con el objetivo de construir las aplicaciones de manera modular. Un componente tiene tres archivos diferenciados.

- Un archivo html que representa la vista y los elementos de un componente que aparecen en el cliente.
- Un archivo Typescript que representa el modelo de datos y el controlador del componente.
- Un archivo css que define los estilos de un componente.

Por último mencionar que la framework dispone de componentes denominados “servicios” que son los encargados de comunicarse con los demás componentes y librerías para facilitar el intercambio de datos.

3.1.2. Typescript

Es un lenguaje de programación de código abierto, desarrollado y mantenido por Microsoft[11], similar a JavaScript. Ambos lenguajes, son compatibles debido a que todos los archivos con extensión Typescript (.ts) se compilan y se convierten en archivos de JavaScript (.js).

3.1.3. Bootstrap

Se trata de un framework CSS [5] desarrollado inicialmente por Twitter cuyo principal objetivo es el de facilitar el desarrollo de una página web a través de las hojas de estilo CSS. En este sentido ofrece los componentes necesarios para implementar la vista de una aplicación web de manera adaptativa. Esto quiere decir que el desarrollador se despreocupa de conocer las características del navegador web del cliente, dado que Bootstrap se encargará de garantizar un diseño similar independientemente del navegador. Para ello utiliza un diseño de filas con doce columnas cada fila, de forma que sea posible especificar para cada componente cuantas columnas va a ocupar y dónde debe mostrarse.

[1] Single page application

Así mismo, la framework, proporciona estilos y efectos para muchos de los componentes de HTML, siendo fácilmente modificables para adaptarlos a cualquier diseño.

3.1.4. Node.js

Node.js[12] es un entorno de ejecución para JavaScript en el lado del servidor que permite ejecuciones asíncronas y orientadas a eventos en tiempo real fuera de un navegador web. Está construido sobre el motor JavaScript V8 de Chrome. En este sentido Angular está construido en torno a Node.js. Es por ello por lo que para poder instalar el framework de Angular, primero se debe tener instalado Node.js, y obtener, a través del gestor de paquetes de Node.js (npm), el módulo de Angular. Observar que a pesar de que Node.js está orientado hacia el lado del servidor, Angular utiliza muchas de las funcionalidades de Node.js para poder implementar la parte front-end.

3.2. Herramientas de la parte del servidor (Back-end)

3.2.1. Apache HTTP Server

Apache HTTP Server[4] es un servidor web desarrollado y mantenido por *Apache Foundation* encargado de recibir, procesar información en la parte del servidor y de renderizar y enviar la información a un cliente que ha realizado una petición.

3.2.2. PHP

PHP[14] es un lenguaje de scripts que permite ejecutar funciones en el servidor con el objetivo de proporcionar una salida en forma de texto plano o un documento HTML.

3.2.3. MariaDB

MariaDB[10] es un sistema de base de datos procedente de MySQL y compatible con este último.

3.2.4. Xampp

Xampp[3] es una recopilación de herramientas de software libre multiplataforma (Linux, Windows y macOS) que proporciona un entorno de desarrollo local para aplicaciones web. Contiene numerosas herramientas tales como un Servidor Apache, una Base de datos MariaDB, o los lenguajes PHP y Perl. En particular dispone de un gestor web para la base de datos MariaDB llamado *phpMyAdmin* que permite realizar operaciones de gestión, administración y mantenimiento de las bases de datos.

3.3. Librerías

3.3.1. ngx-bootstrap

Se trata de una librería desarrollada y mantenida por *Valor Software*[17] para trabajar con Angular. Esta herramienta, otorga la posibilidad de utilizar distintos componentes con múltiples funcionalidades con el estilo CSS ya aplicado de las hojas de estilo de Bootstrap. Como ejemplo de estos componentes están incluidos: botones, alertas, selectores de fecha, modales, globos

de consejo y barras de progreso. Esta librería se encuentra en el repositorio de Node.js a través del gestor de paquetes npm, puede ser descargada e instalada en el proyecto de Angular. Contiene una documentación muy extensa, bien presentada y sencilla de entender, con ejemplos para facilitar el uso de todos los componentes que ofrece. Se encuentra en su versión 4.2 bajo licencia MIT.

3.3.2. Chart.js

Chart.js[6] contiene una serie de herramientas de código libre, destinadas a la creación de gráficas utilizando tecnología JavaScript, así como de elementos gráficos donde los parámetros de representación de los datos pueden ser fácilmente modificados. En su página web, existe una documentación muy cuidada, fácil de entender y con multitud de ejemplos. Todas estas herramientas se encuentran en el repositorio de Node.js a través del gestor de paquetes npm y puede ser descargada e instalada en el proyecto de Angular. Esta librería se encuentra bajo licencia MIT.

3.3.3. Font Awesome

Font Awesome[7] es una galería de iconos vectorizados y fuentes de texto basada en hojas de estilo CSS utilizados por desarrolladores y diseñadores web. Permiten una gran flexibilidad a la hora de implementarlos gracias a su hoja de estilos en donde se encuentran todas las propiedades que puede tener un icono, permitiendo al desarrollador modificarlas. Algunas de estas propiedades son: size, color, shadow, etc.

3.4. Otras Herramientas

3.4.1. Visual Studio Code

Se trata de un editor de código fuente capaz de integrar multitud de plug-ins entre los cuales se encuentran Git y pistas, anotaciones y consejos para: Typescript, PHP, SQL, CSS. Entre sus características se encuentra la capacidad de reconocer automáticamente los proyectos de Angular, y organizar su estructura, iconos y sugerencias de código para los diferentes elementos de los componentes que forman un proyecto. También permite abrir varios archivos y colocarlos en diferentes localizaciones, consiguiendo de esta manera que el proceso de codificación sea sencillo, y rápido. Además, avisa de los errores sintácticos o de programación que se producen y posee unas normas de estilo que fuerzan a que el código sea limpio y uniforme.

3.4.2. Git

Git es un sistema de control de versiones que permite almacenar un historial de versiones con los cambios realizados en código fuente para que, en caso necesario, poder volver a una versión anterior. Sirve también para desarrollar varias versiones. Para este proyecto se ha utilizado la plataforma git de Github[8].

3.4.3. Postman

Postman[15] es una aplicación que permite enviar peticiones a un servidor web utilizando alguno de los protocolos disponibles para tal fin. Para ello encapsula una petición en un formato tal como JSON, XML, texto plano y otros. Permite modificar los parámetros, cabeceras y cuerpo de una petición para ser enviada a un servidor y esperar su respuesta. Una vez se ha obtenido

una respuesta por parte del servidor, esta herramienta es capaz de mostrar de una forma ordenada toda la información recibida como las cookies, el cuerpo del paquete y las cabeceras del mismo.

4. Casos de uso

En este capítulo se describen los casos de uso definidos para desarrollar la aplicación. Para ello en la sección 4.1 se definirán los principales términos utilizados en el capítulo, en la sección 4.2 se describirá el diagrama de casos de uso y los actores que intervienen y por último se mostrarán los diferentes casos de uso definidos.

4.1. Definición de términos

Espacio – Se considera espacio a la zona restringida a un profesional. Es absolutamente privado y únicamente accesible a cada profesional. Este espacio contiene todos los datos, y herramientas para permitir al profesional trabajar con la aplicación.

Profesional – Se considera profesional al usuario encargado de utilizar la aplicación con el rol "*profesional"*. El profesional es el administrador único de su espacio en la aplicación y será el usuario principal de la aplicación.

Paciente - Se considera paciente a la entidad que representa a un paciente. Cada paciente es único y los datos del mismo son generados por el profesional.

Código de registro – Es un código único aleatorio generado por el administrador del sitio que permite dar de alta a un nuevo profesional.

Dieta - Se considera dieta a la entidad que define lo que debe ingerir un paciente. La dieta es generada por un profesional, va asociada a un paciente, y se genera a partir de los datos que contiene un paciente. Si la dieta se asocia con un profesional, se considerará una *"dieta predefinida".*

Datos - Son todas aquellas características que un profesional ha asignado a cada paciente o que ha generado la aplicación.

Restricciones de paciente – Se trata de características del paciente que deben ser consideradas para crear una dieta por parte del profesional.

Cita – Es un evento que define los detalles de una reunión entre un paciente y su profesional asociado.

Medidas – Hace referencia a los datos antropométricos de cada paciente.

Métricas – Hace referencia al resultado de los cálculos obtenidos a partir de las medidas y datos asociados a un paciente.

4.2. Diagrama de Casos de Uso

A continuación, en la Figura 7 se muestra el diagrama de casos de uso perteneciente a la aplicación.

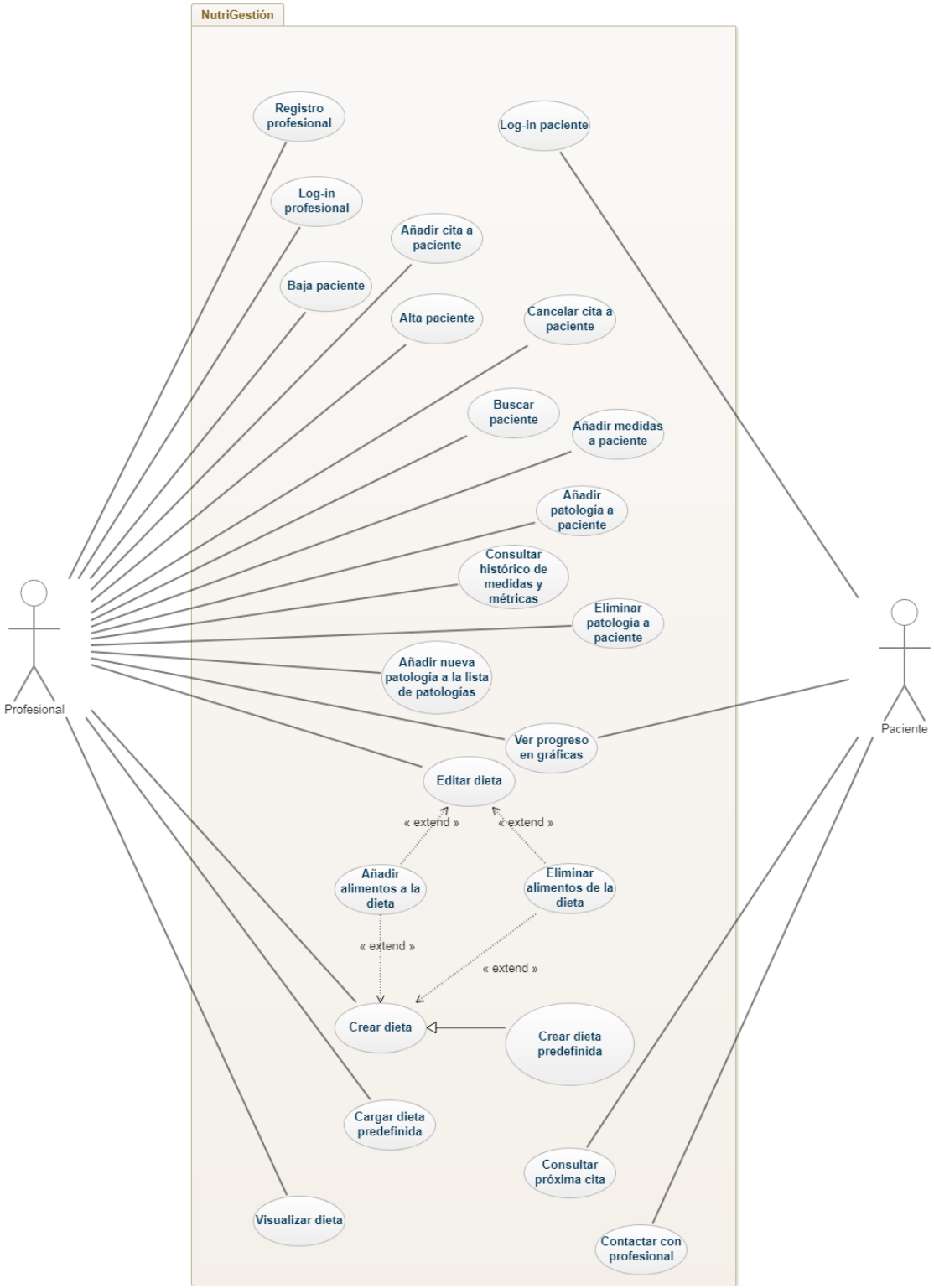


Figura 7 - Diagrama casos de uso

En la aplicación existen dos actores diferenciados: el profesional y el paciente. El profesional es el usuario principal encargado de generar y visualizar todo el contenido que gestiona la aplicación. En este sentido tiene la capacidad de crear pacientes, citas, dietas, y patologías, así como asociar la información creada a sus pacientes (medidas, métricas, patologías, dietas y citas). Por su parte, el paciente únicamente tiene la capacidad de consumir el contenido que tiene asociado de forma visual.

4.3. Casos de uso

En esta sección se muestran los casos de uso de la aplicación separados en casos de uso relacionados con el profesional, y casos de uso relacionados con el paciente. Estos casos de uso se representan en tablas y capturan los requisitos para ayudar a definir la arquitectura y el diseño de la aplicación.

4.3.1. Casos de uso relacionados con el paciente

A continuación, se muestran los casos de uso relacionados con el paciente en la aplicación. Estos casos de uso son aquellos que puede realizar el paciente en el sistema.

CU - 01	**Log-in**	
Objetivos asociados	Acceder al sistema con el rol de paciente	
Entradas	• Correo electrónico • Contraseña	
Salidas	Vista principal paciente	
Precondición	• El profesional ha debido registrar en el sistema al paciente	
Secuencia Normal	**Paso**	**Acción**
	1	Acceder a la pagina de log-in
	2	Introducir credenciales
	3	Validar
Postcondición	Usuario accede al sistema con rol de paciente teniendo disponibles todas las funcionalidades asociadas a este rol	
Excepciones	**Paso**	**Acción**
	S-1	Credenciales introducidas incorrectas
	S-2	Paciente no registrado en el sistema
Comentarios	Se cargará la vista del paciente en la aplicación	

CU – 02	**Consultar progreso**	
Objetivos asociados	Visualizar gráficamente el progreso de sus medidas y métricas asociadas a su histórico	
Entradas	Ninguna	
Salidas	Gráficas del histórico de su progreso	
Precondición	Estar registrado como paciente Paciente activo Existencia de medidas y métricas asociadas al paciente en la base de datos	
Secuencia Normal	**Paso**	**Acción**
	1	Log-in como paciente
	2	Navegar por las diferentes pestañas
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	No existen medidas y métricas asociadas al paciente
Comentarios		

CU – 03	**Consultar dieta**	
Objetivos asociados	Visualizar en un formato limpio la dieta asociada al paciente	
Entradas	Pulsar sobre “Mi Dieta”	
Salidas	Dieta en bien formateada asociada al paciente	
Precondición	Estar registrado como paciente Paciente activo Existencia de dieta asociada al paciente en la base de datos	
Secuencia Normal	**Paso**	**Acción**
	1	Log-in como paciente
	2	Pulsar en “Mi Dieta”
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	No existe dieta asociada al paciente
Comentarios		

CU – 04	**Consulta próxima cita**	
Objetivos asociados	Visualizar la próxima cita activa del paciente	
Entradas	Pulsar en “Mi Próxima Cita”	
Salidas	Visualización de los datos de la próxima cita asociada al paciente	
Precondición	Estar registrado como paciente Paciente activo	
Secuencia Normal	**Paso**	**Acción**
	1	Log-in como paciente
	2	Pulsar en “Mi Próxima Cita”
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	No existe cita asociada al paciente
Comentarios	Si no existe una cita asociada al paciente, se mostrará un mensaje, anunciando que no tiene próximas citas	

CU – 05	**Contactar con el profesional**	
Objetivos asociados	Escribir un correo electrónico al profesional asociado al paciente	
Entradas	Pulsar en “Contactar Nutricionista”	
Salidas	Se abrirá el administrador de correo electrónico disponible en la máquina del paciente con la dirección de destino correspondiente a su profesional	
Precondición	Estar registrado como profesional Paciente activo Existencia de un gestor de correo electrónico por defecto	
Secuencia Normal	**Paso**	**Acción**
	1	Log-in como paciente
	2	Pulsar en “Contactar Nutricionista”
	3	Escribir correo electrónico deseado
	4	Enviar correo electrónico
Postcondición	Correo electrónico enviado	
Excepciones	**Paso**	**Acción**
	S-1	Gestor de correo electrónico no configurado correctamente
Comentarios		

4.3.2. Casos de uso relacionados con el profesional

A continuación, se muestran los casos de uso relacionados con el profesional en la aplicación. Estos casos de uso son aquellos que puede realizar el profesional en el sistema.

CU – 06	**Registro**	
Objetivos asociados	Registrar en el sistema a un usuario con el rol de *profesional*	
Entradas	• Nombre • Correo electrónico • Contraseña	
Salidas	Mensaje de éxito	
Precondición	• Poseer código de registro • Código de registro registrado en la base de datos • Correo electrónico no registrado en base de datos	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar “Tengo un Código”
	2	Introducir el código de registro único
	3	Rellenar los datos
	4	Validar
Postcondición	Nuevo profesional registrado en el sistema	
Excepciones	**Paso**	**Acción**
	S-1	Correo electrónico ya registrado en el sistema
Comentarios		

CU – 07	**Log-in**	
Objetivos asociados	Acceder al sistema con el rol de profesional	
Entradas	• Correo electrónico • Contraseña	
Salidas	Vista principal de profesional	
Precondición	• Haber creado previamente una cuenta con rol profesional	
Secuencia Normal	**Paso**	**Acción**
	1	Acceder a la pagina de log-in
	2	Introducir credenciales
	3	Validar
Postcondición	Usuario accede al sistema con rol de profesional teniendo disponibles todas las funcionalidades asociadas a este rol	
Excepciones	**Paso**	**Acción**
	S-1	Credenciales introducidas incorrectas
	S-2	Profesional no registrado en el sistema
Comentarios	Se cargará la vista correspondiente al profesional en la aplicación	

CU - 08	**Alta paciente**	
Objetivos asociados	Crea un nuevo paciente en la aplicación y se asociará al profesional que lo ha creado Esta acción creará una cuenta con la que posteriormente el paciente podrá realizar log-in en la aplicación y acceder a su espacio. Si el email del paciente ya existía, se volverá a activar su cuenta	
Entradas	• Nombre • Apellido 1 • Apellido 2 • Edad • Sexo • Email • Contraseña	
Salidas	Aparecerá la página del paciente creado	
Precondición	• Estar registrado como profesional	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar nuevo paciente
	2	Rellenar campos disponibles de paciente
	3	Validar
Postcondición	Paciente registrado en el sistema	
Excepciones	**Paso**	**Acción**
	S-1	Paciente ya existe
	S-2	Datos necesarios no proporcionados
Comentarios		

CU – 09	**Baja paciente**	
Objetivos asociados	Desactiva un paciente	
Entradas	• Confirmación explicita de la eliminación	
Salidas	Confirmación de la desactivación del paciente	
Precondición	• Estar registrado como profesional • Acceder a ficha de paciente	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente y acceder a su ficha
	2	Seleccionar “Desactivar paciente”
	3	Confirmar escribiendo “si” en el cuadro de texto proporcionado
	4	Validar
Postcondición	Paciente marcado como baja de la base de datos	
Excepciones	**Paso**	**Acción**
	S-1	Profesional, introduce en cuadro de confirmación texto distinto de “si”
Comentarios		

CU - 10	**Añadir cita a paciente**	
Objetivos asociados	Asocia una cita a un paciente	
Entradas	• Fecha • Hora	
Salidas	Mostrará su cita frente a su nombre en la lista de pacientes	
Precondición	• Estar registrado como profesional • Tener al menos un paciente registrado • El paciente está activo	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente
	2	Seleccionar “añadir cita”
	3	Rellenar los campos de la cita
	4	Validar
Postcondición	Nueva cita asociada a un paciente	
Excepciones	**Paso**	**Acción**
	S-1	No existe ningún paciente asociado
	S-2	Fecha y día no validos
Comentarios	Si se el paciente ya tiene asignado una cita, y se le asigna otra en otra fecha u hora diferente, la cita anterior se sobrescribirá	

CU - 11	**Cancelar cita a paciente**	
Objetivos asociados	Elimina la cita de un paciente	
Entradas	Ninguna	
Salidas	Desaparecerá la fecha de cita asociada de la visualización	
Precondición	• Estar registrado como profesional • Tener al menos un paciente registrado • El paciente registrado está activo • El paciente registrado tiene una cita activa y asociada	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente
	2.1	Pinchar botón “Atendido”
	2.2	Pinchar cruz roja al lado de la cita
Postcondición	Cita marcada como no activa en la base de datos	
Excepciones	**Paso**	**Acción**
	S-1	No existe ninguna cita asociada a un paciente
Comentarios		

CU - 12	**Buscar paciente**	
Objetivos asociados	Buscar a un paciente por nombre o apellido en la lista de pacientes de un profesional	
Entradas	Nombre o apellido en el cuadro de búsqueda	
Salidas	Aparecerá en la lista de pacientes los nombres y apellidos que coincidan con los términos de búsqueda.	
Precondición	• Estar registrado como profesional • Tener al menos un paciente registrado	
Secuencia Normal	**Paso**	**Acción**
	1	Escribir en el cuadro de búsqueda
Postcondición	La lista de pacientes mostrará los coincidentes	
Excepciones	**Paso**	**Acción**
	S-1	No existe ningún paciente registrado
Comentarios		

<table>
<tr><td>CU - 13</td><td colspan="2">Añadir medidas a paciente</td></tr>
<tr><td>Objetivos asociados</td><td colspan="2">Permite rellenar los datos de medidas pertenecientes a un paciente</td></tr>
<tr><td>Entradas</td><td colspan="2">• Peso
• Altura
• Pliegue tríceps
• Pliegue cresta iliaca
• Pliegue subescapular
• Pliegue bíceps
• Pliegue supraspina
• Pliegue abdominal
• Pliegue muslo
• Pliegue pierna
• Perímetro brazo relajado
• Perímetro brazo flexionado
• Perímetro cintura
• Perímetro cadera
• Perímetro pierna
• Diámetro muñeca
• Diámetro humero
• Diámetro biepicondilar fémur</td></tr>
<tr><td>Salidas</td><td colspan="2">Métricas actualizadas en tiempo real</td></tr>
<tr><td>Precondición</td><td colspan="2">• Estar registrado como profesional
• Tener un paciente registrado y asignado al profesional
• Acceder a la ficha del paciente al cual se quieren modificar sus medidas</td></tr>
<tr><td rowspan="3">Secuencia Normal</td><td>Paso</td><td>Paso</td></tr>
<tr><td>1</td><td>Seleccionar al paciente</td></tr>
<tr><td>2</td><td>Modificar los diferentes campos</td></tr>
<tr><td>Postcondición</td><td colspan="2">Nuevas medidas asociadas al paciente y añadidas a su histórico de medidas</td></tr>
<tr><td>Comentarios</td><td colspan="2"></td></tr>
</table>

CU - 14	**Consultar histórico de medidas y métricas**	
Objetivos asociados	Consulta el histórico en forma de datos numéricos de medidas y métricas de un paciente	
Entradas	Seleccionar en la ficha del paciente la fecha del histórico que se desea consultar	
Salidas	Se rellenarán las medidas y métricas de acuerdo con los datos consultados	
Precondición	Estar registrado como profesional Paciente activo Tener un paciente asociado con al menos dos tomas de medidas en la base de datos asociadas a ese paciente	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente a consultar
	2	Seleccionar del histórico unas medidas pertenecientes a una fecha
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	No existen medidas asociadas al paciente
Comentarios	Las métricas se recalcularán en función de las medidas seleccionadas y se mostrará el cálculo en tiempo real al mismo tiempo que aparecen las medidas selecionadas	

CU - 15	**Añadir patología a paciente**	
Objetivos asociados	Añadir una o más patologías a un paciente seleccionado	
Entradas	Selección de patologías deseadas	
Salidas	Se mostrará en la ficha del paciente las patologías asociadas	
Precondición	Estar registrado como profesional Paciente activo Existencia de patologías registradas en la base de datos	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente
	2	Pulsar en patologías
	3	Asociar tantas patologías como se desee
	4	Pulsar en “Guardar” para validar
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	Lista de patologías vacía
Comentarios		

CU - 16	**Eliminar patología a paciente**	
Objetivos asociados	Eliminar una o más patologías a un paciente seleccionado	
Entradas	Selección de patologías deseadas	
Salidas	Se mostrará en la ficha del paciente las patologías asociadas	
Precondición	Estar registrado como profesional Paciente activo Existencia de patologías registradas en la base de datos Existencia de patologías asociadas al paciente	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente
	2	Pulsar en patologías
	3	Desmarcar tantas patologías como se desee
	4	Pulsar en “Guardar” para validar
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	Lista de patologías vacía
Comentarios		

CU – 17	**Añadir nueva patología a la lista de patologías**	
Objetivos asociados	Añadir una o más patologías a la base de datos	
Entradas	Nombre de la patología	
Salidas	Aparecerá en la lista de patologías	
Precondición	Estar registrado como profesional Paciente activo	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente
	2	Pulsar en patologías
	3	Escribir el nombre de la nueva patología en el cadro de texto
	4	Pulsar en “+” para validar
Postcondición	Nueva patología en la base de datos Nueva patología asociada al profesional que la ha creado	
Excepciones	**Paso**	**Acción**
	S-1	La patología ya existe
Comentarios	La nueva patología se asociará al profesional que la ha creado y únicamente será visible para el profesional en concreto. Existen también patologías comunes a todos los profesionales que podrán ser introducidas por un técnico.	

CU – 18	**Ver progreso en graficas del paciente**	
Objetivos asociados	Visualizar mediante gráficas el histórico de medidas y métricas del paciente	
Entradas	Ninguna	
Salidas	Se mostrará las gráficas del histórico	
Precondición	Estar registrado como profesional Paciente activo Medidas y métricas asociadas al paciente selecionado	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar paciente
	2	Pulsar en progreso
	3	Moverse por las diferentes pestañas
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	S-1	Paciente no tiene medidas asociadas
Comentarios		

CU – 19	**Añadir alimentos a la dieta**	
Objetivos asociados	Añadir múltiples alimentos a una dieta	
Entradas	• Nombre del alimento • Cantidad • Unidades de la cantidad	
Salidas	Aparecerá el alimento añadido	
Precondición	Estar registrado como profesional	
Secuencia Normal	**Paso**	**Acción**
	1	Pulsar sobre el botón “+”
	2	Rellenar los datos de entrada
Postcondición	Alimento añadido a la dieta	
Comentarios	Si anteriormente se ha introducido un alimento en la base de datos, al escribir su nombre, si existe, se autocompletarán las unidades con las asociadas a ese alimento en la base de datos	

CU – 20	**Eliminar alimentos a la dieta**	
Objetivos asociados	Eliminar un alimento que se ha añadido a nuestra dieta	
Entradas	Pulsar botón “-“	
Salidas	Desaparecerá el alimento	
Precondición	Estar registrado como profesional Alimento existente en la dieta	
Secuencia Normal	**Paso**	**Acción**
	1	Pulsar sobre el botón “-”
Postcondición	Alimento eliminado de la dieta	
Comentarios		

CU – 21	**Crear dieta para paciente**	
Objetivos asociados	Asociar una dieta a un paciente	
Entradas	• Dieta con al menos un alimento en una franja horaria • Nombre de la dieta (Opcional)	
Salidas	Visualización de la dieta en la lista de dietas del paciente	
Precondición	Estar registrado como profesional Paciente seleccionado activo	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar el paciente deseado
	2	Pulsar en dietas
	3	Añadir/Eliminar alimentos en las franjas deseadas
	4	Añadir un nombre (Opcional)
	5	Pulsar en guardar
Postcondición	Nueva dieta registrada en la base de datos y asociada al paciente seleccionado	
Comentarios		

CU – 22	**Crear dieta predefinida de profesional**	
Objetivos asociados	Asociar una dieta a un profesional	
Entradas	• Dieta con al menos un alimento en una franja horaria • Nombre de la dieta (Opcional)	
Salidas	Visualización de la dieta en la lista de dietas predefinas en la vista “Dietas de paciente”	
Precondición	Estar registrado como profesional	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar “Dietas predefinas” en la vista principal del profesional
	3	Añadir/Eliminar alimentos en las franjas deseadas
	4	Añadir un nombre (Opcional)
	5	Pulsar en guardar
Postcondición	Nueva dieta registrada en la base de datos y asociada al profesional	
Comentarios	Esta dieta podrá ser cargada posteriormente en la acción “Crear dieta para paciente” con la ventaja de que ya posee multitud de alimentos precargados en franjas horarias	

CU – 23	**Cargar dieta predefinida a paciente**	
Objetivos asociados	Rellenar los alimentos de una nueva dieta en sus franjas horarias correspondientes, los cuales pertenecen a una dieta predefinida por el profesional.	
Entradas	Seleccionar dieta de un histórico	
Salidas	Se rellenará la nueva dieta con los mismos datos que contiene la dieta predefinida	
Precondición	Estar registrado como profesional Existencia de dietas predefinidas	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar de la lista de dietas predeterminadas la dieta
Postcondición	Nueva dieta poblada con datos de la dieta predefinida	
Comentarios	Se pueden añadir o eliminar nuevos alimentos a esta nueva dieta, así como darle un nuevo nombre	

CU – 24	**Editar dieta**	
Objetivos asociados	Editar cualquier dieta creada anteriormente	
Entradas	Seleccionar la dieta deseada a editar	
Salidas	Nueva dieta poblada con datos de la dieta a editar seleccionada	
Precondición	Estar registrado como profesional Existencia de dietas creadas asociadas al profesional o al paciente	
Secuencia Normal	**Paso**	**Acción**
	1	Seleccionar dietas paciente o dietas predefinidas
	2	Pulsar “editar” sobre la dieta deseada en la lista de dietas
	3	Añadir o eliminar alimentos deseados
	4	Pulsar en “Guardar” para validar y guardar la dieta editada
Postcondición	Ninguna	
Comentarios	La edición de una dieta, supone la creación de una nueva dieta con el objetivo de mantener un histórico de dietas	

CU – 25	**Visualizar dieta**	
Objetivos asociados	Visualizar en un formato limpio cualquier dieta	
Entradas	• Dieta predefinida o dieta asociada a paciente desde la lista de dietas	
Salidas	Visualización de la dieta en un formato adecuado	
Precondición	Estar registrado como profesional o paciente	
Secuencia Normal	**Paso**	**Acción**
	N1-1	Como profesional, seleccionar el paciente deseado
	N1-2	Pulsar en dietas
	N1-3	Pulsar sobre cualquier dieta deseada
	N2-1	Estar registrado como paciente
	N2-2	Pulsar sobre mi dieta
Excepciones	**Paso**	**Acción**
	S-1	No existen dietas asociadas a profesional o a paciente
Postcondición	Nueva dieta registrada en la base de datos y asociada al paciente seleccionado	
Comentarios		

5. Modelo de datos

En este capítulo se describe el modelo de datos definido para realizar la persistencia de la información, así como la implementación de las bases de datos utilizada. En primer lugar, se mostrará un modelo entidad-relación y a continuación se presentará la estructura y contenido de la base de datos implementada.

5.1. Modelo Entidad-Relación

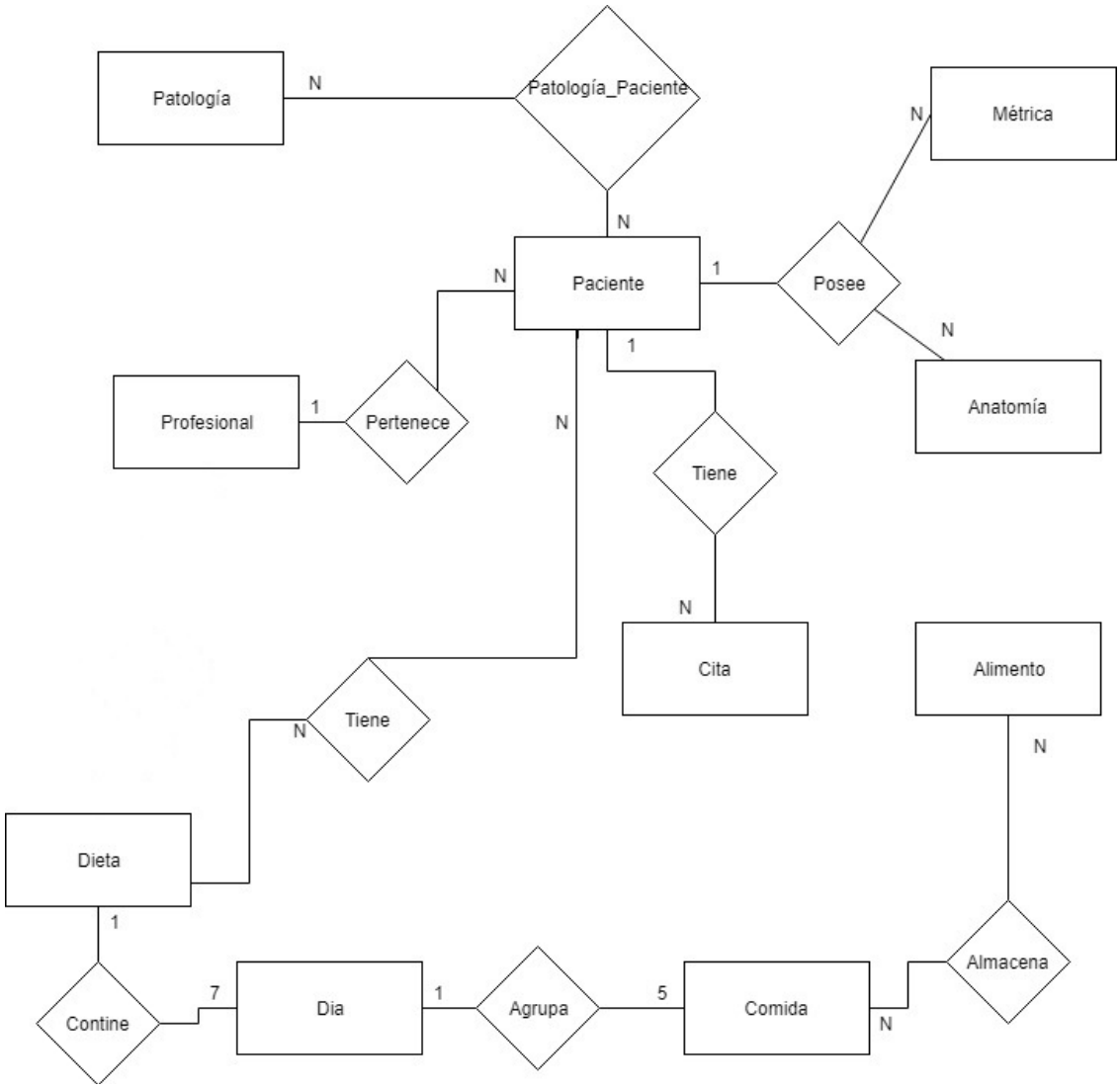


Figura 8 - Entidad-relación

Como se aprecia en la *Figura 8,* la entidad *"paciente*" es la principal en el modelo de datos ya que es el nexo mediante el cual se relacionan las demás entidades. La entidad "*Dieta*" se relaciona con las entidades "*Dia"*, "*Comida"* y "*Alimento"* con el objetivo obtener toda la información necesaria para crear una dieta, y un paciente puede tener asociadas muchas dietas. Por otro lado, es posible conocer todos los datos de un paciente a través de las relaciones que

establece con las demás entidades. De este modo, un profesional puede estar relacionado con muchos pacientes, mientras que un paciente pertenece únicamente a un profesional. Un paciente puede tener muchas citas, en cambio una cita pertenece a un único paciente. Las métricas y anatomía están relacionadas con un único paciente mientras que un paciente puede relacionarse con muchas de estas entidades. Los pacientes tienen asociadas muchas patologías. Estas patologías puedes estar asocias a muchos pacientes.

5.2. Estructura base de datos

El modelo entidad-relación descrito en el apartado anterior se ha implementado mediante una base de datos relacional denominada "*nutriGestion*". A continuación, se describirá la estructura de la base de datos, el propósito y significado de las tablas creadas; el significado y la utilidad de sus campos, y se explicará mediante qué elementos se relacionan entre sí las diferentes tablas. La *Figura 9* muestra gráficamente la relación entre las diferentes tablas y sus atributos.

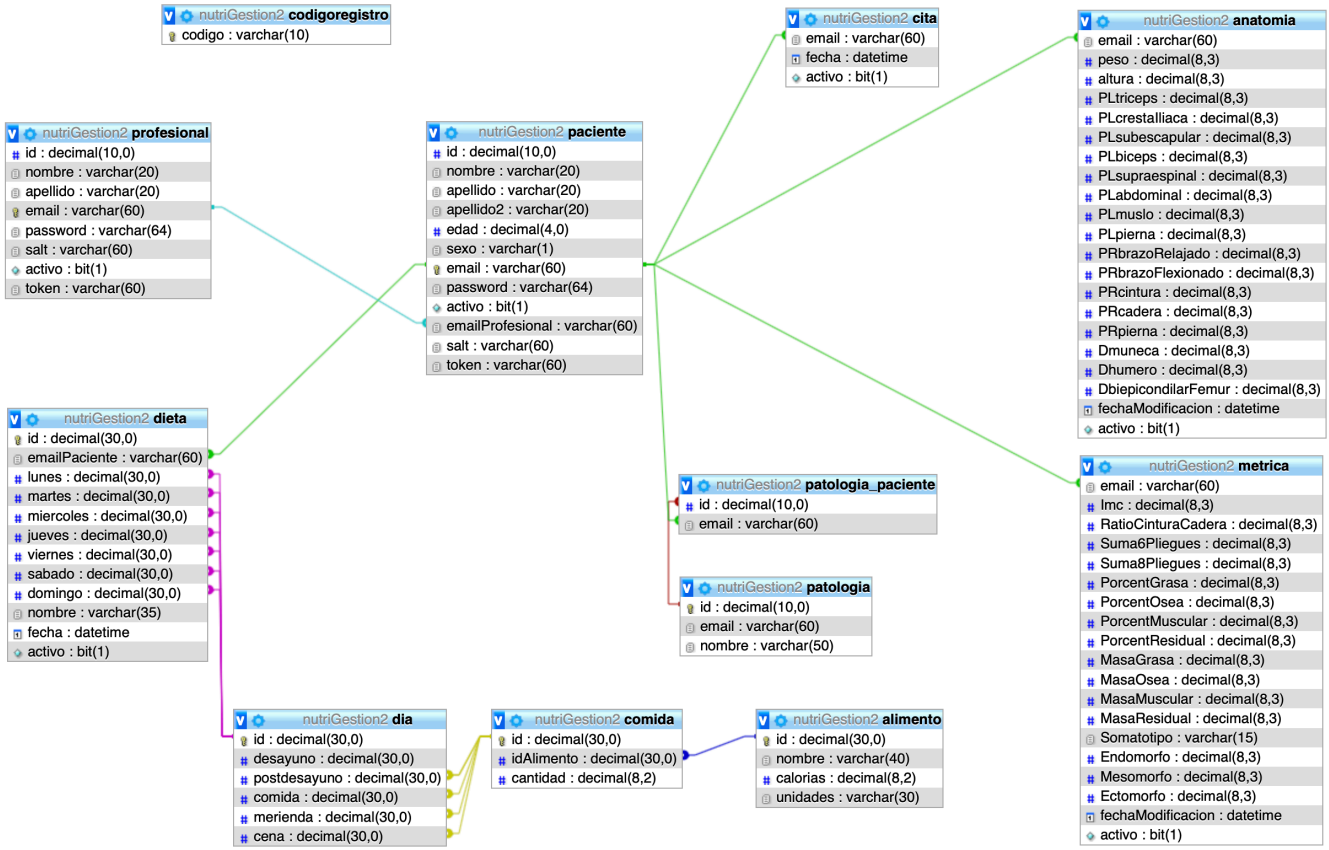


Figura 9 - Estructura de la base de datos

5.2.1. codigoregistro

Contiene una cadena de caracteres que representa un código de registro único generado aleatoriamente. Está formada por el siguiente campo de información:

Columna	Significado
Código (PK)	Código único

5.2.2. profesional

Contiene la información que representa al profesional en la aplicación. Está formado por los siguientes campos de información:

Columna	Significado
id	Código numérico del profesional
nombre	Nombre del profesional
apellido	Apellido del profesional
email (PK)	Email del profesional
password	Clave hash generada a partir de una contraseña introducida y concatenada con el campo *salt* y *pepper*
salt	Cadena hash generada aleatoriamente que permite el cifrado y descifrado de la contraseña
activo	Bit que indica si el profesional está activo o no
token	Clave hash de tipo sha1 generada al concatenar el email, contraseña e email. Tiene funciones a la hora de la autentificación

5.2.3. paciente

Contiene la información que define al paciente en la aplicación, así como atributos de estado y autenticación. La clave principal de esta tabla es el campo "*emailprofesional"*, encargada de relacionar a un paciente con su profesional. Está formado por los siguientes campos de información:

Columna	Significado
id	Código numérico del paciente
nombre	Nombre del paciente
apellido	Apellido del paciente
apellido2	Segundo apellido del paciente
edad	Edad del paciente
sexo	Sexo del paciente. h-> hombre m-> mujer
email (PK)	Email del paciente
password	Clave hash generada a partir de una contraseña introducida y concatenada con el campo *salt* y *pepper*
activo	Bit que indica si el profesional está activo o no
emailProfesional	Email del profesional que ha generado este paciente
salt	Cadena hash generada aleatoriamente que permite el cifrado y descifrado de la contraseña
token	Clave hash de tipo sha1 generada al concatenar el email, contraseña e email. Tiene funciones a la hora de la autentificación

5.2.4. cita

Contiene el histórico de citas del paciente. Se relaciona con la tabla "*5.2.3 paciente"* a través del campo *"email".* Está formado por los siguientes campos de información:

Columna	Significado
Email (PK)	Email del paciente
Fecha (PK)	Contiene la fecha de la cita en formato UTC
activo	Bit que indica si la cita está activa o no

5.2.5. patología_paciente

Contiene la relación de las tablas "*5.2.3 paciente"* y "*5.2.6 patología".* Está formado por los siguientes campos de información:

Columna	Significado
Id (PK)	Referencia al id de la tabla 5.2.6 patología
Email (PK)	Email del paciente

5.2.6. patología

Contiene el nombre de una patología y el email del profesional que la ha creado. Se relaciona con el email de la tabla paciente a través de una tabla intermedia denominada *"5.2.5 patologia_paciente".* Está formado por los siguientes campos de información:

Columna	Significado
Id(PK)	Identificador único de una patología
email	Email del paciente creador de la patología
nombre	Nombre de la patología

5.2.7. anatomía

Contiene las medidas de un paciente, la fecha de inserción y un bit de activo. Se relaciona con un paciente a través del campo *email.* Está formado por los siguientes campos de información:

Columna	Significado
Email (PK)	Email del paciente al que pertenecen los datos
peso	peso
altura	altura
PLtriceps	Pliegue de tríceps
PLcrestailiaca	Pliegue de cresta iliaca
PLsubescapular	Pliegue subescapular
PLbiceps	Pliegue de bíceps
PLsupraespinal	Pliegue supra espinal
PLabdominal	Pliegue abdominal
PLmuslo	Pliegue muslo
PLpierna	Pliegue de pierna
PRbrazoRelajado	Perímetro brazo relajado
PRbrazoFlexionado	Perímetro brazo flexionado
PRcintura	Perímetro de la cintura
PRcadera	Perímetro de la cadera
PRpierna	Perímetro de la pierna
Dmuneca	Diámetro de la muñeca
Dhumero	Diámetro del húmero
DbiepicondilarFemur	Diámetro biepicondilar del Fémur
fechaModificacion (PK)	Fecha de la introducción de la anatomía
activo	La medida es la más reciente para el paciente

5.2.8. métrica

Contiene las métricas de un paciente, la fecha de inserción y un bit de activo. Se relaciona con un paciente a través del campo *email.* Está formado por los siguientes campos de información:

Columna	Significado
Email(PK)	Email del paciente al que pertenecen los datos
Imc	Índice de masa corporal
RatioCinturaCadera	Cociente entre la cintura y la cadera
Suma6Pliegues	Suma de los 6 pliegues
Suma8Pliegues	Suma de los 8 pliegues
PorcentGrasa	Porcentaje de grasa
PorcentOsea	Porcentaje óseo
PorcentMuscular	Porcentaje muscular
PorcentResidual	Porcentaje residual
MasaGrasa	Masa grasa
MasaOsea	Masa ósea
MasaMuscular	Masa muscular
MasaResidual	Masa residual
Somatotipo	Somatotipo corporal
Endomorfo	Índice de endomorfo
Mesomorfo	Índice de mesomorfo
Ectomorfo	Índice de ectomorfo
fechaModificacion(PK)	Fecha de la introducción de la métrica
activo	La métrica es la más reciente para el paciente

5.2.9. dieta

Contiene siete campos representado los días de la semana, un nombre para la dieta y el email del paciente al que va asociada. En los campos que representan los días de la semana, se almacena un "*id"* que referencia a la clave primaria "*id"* de la tabla "5*.2.10 día"*. Está formado por los siguientes campos de información:

Columna	Significado
Id(PK)	Identificador único de una dieta
emailPaciente	Email del paciente al que pertenecen la dieta
lunes	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
martes	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
miercoles	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
jueves	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
viernes	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
sabado	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
domingo	Contendrá un *id* referente al *id* de la tabla 5*.2.10 día*
nombre	Nombre perteneciente a dicha dieta
fecha	Fecha de creación de la dieta
activo	La métrica es la más reciente para el paciente

5.2.10. día

Contiene cinco campos los cuales representan las diferentes franjas horarias. Cada franja horaria, contiene un id correspondiente a una comida y es a su vez referenciado a través del *"id"* por la tabla *6.2.9 dieta.* Está formado por los siguientes campos de información:

Columna	**Significado**
Id(PK)	Identificador único de un día
desayuno	Contendrá un *id* referente al *id* de la tabla 5*.2.11 comida*
postdesayuno	Contendrá un *id* referente al *id* de la tabla 5*.2.11 comida*
comida	Contendrá un *id* referente al *id* de la tabla 5*.2.11 comida*
merienda	Contendrá un *id* referente al *id* de la tabla 5*.2.11 comida*
cena	Contendrá un *id* referente al *id* de la tabla 5*.2.11 comida*

5.2.11. comida

Contiene un id al que referencia una comida y se identifica en la tabla "*5.2.12 alimento"* a través de "*idAlimeto*". Está formado por los siguientes campos de información:

Columna	**Significado**
Id(PK)	Identificador único de una comida
idAlimento	Contendrá un *id* referente al *id* de la tabla 5*.2.12 alimento*

5.2.12. alimento

Contiene como clave primaria un "*id"* para ser referenciada por la tabla "*5.2.11 comida"*. Está formado por los siguientes campos de información:

Columna	**Significado**
Id(PK)	Identificador único de un alimento
nombre	Nombre del alimento
calorias	Calorías del alimento
unidades	Unidades del alimento

5.3. Modelo de datos del controlador

Observar que el modelo de datos elegido debe reflejarse en la aplicación. Por ello han sido creadas a través de Typescript las clases necesarias (*Figura 10)* con el objetivo de que exista una correspondencia entre la base de datos y las clases con las que se debe trabajar en Typescript.

a)

```
export class Alimento {

    id: number;
    nombre: string;
    calorias: number;
    franja: string;
    cantidad: number;
    unidades: string;

    constructor(franja: string) {
      this.franja = franja;
      this.cantidad = 0;
      this.unidades = 'gramos';
    }

  }
```

b)

```
export class Anatomia {
    email: string;
    peso: number;
    altura: number;
    PLtriceps: number;
    PLcrestaIliaca: number;
    PLsubescapular: number;
    PLbiceps: number;
    PLsupraespinal: number;
    PLabdominal: number;
    PLmuslo: number;
    PLpierna: number;
    PRbrazoRelajado: number;
    PRbrazoFlexionado: number;
    PRcintura: number;
    PRcadera: number;
    PRpierna: number;
    PRmuneca: number;
    Dmuneca: number;
    Dhumero: number;
    DbiepicondilarFemur: number;
    fechaModificacion: Date;
    activo: string;
```

c)

```
export class Cita {

    email: string;
    fecha: Date;
    activo: string;

  }
```

d)

```
export class Dieta {

    lunes: Alimento[] = [];
    martes: Alimento[] = [];
    miercoles: Alimento[] = [
    jueves: Alimento[] = [];
    viernes: Alimento[] = [];
    sabado: Alimento[] = [];
    domingo: Alimento[] = [];
    activo: boolean;
    nombre: string;
    fecha: string;

  }
```

e)

```
export class Metricas {
    paciente: Paciente;
    email: string;

    Imc: number;
    RatioCinturaCadera: numbe
    Suma6Pliegues: number;
    Suma8Pliegues: number;
    PorcentGrasa: number;
    PorcentOsea: number;
    PorcentMuscular: number;
    PorcentResidual: number;
    MasaGrasa: number;
    MasaOsea: number;
    MasaMuscular: number;
    MasaResidual: number;

    Somatotipo: string;
    Endomorfo: number;
    Mesomorfo: number;
    Ectomorfo: number;

    fechaModificacion: Date;
    activo: string;
```

f)

```
export class Paciente {
    id: number;
    nombre: string;
    apellido: string;
    apellido2: string;
    edad: number;
    sexo: string;
    email: string;
    password: string;
    emailProfesional: string;
    activo: string;
    citas: Cita[];
    anatomia: Anatomia;
    patologias: Patologia[];

  }
```

g)

```
export class Patologia

    id: number;
    nombre: string;
    checked: boolean;
```

h)

```
export class Profesional {
    id: number;
    nombre: string;
    apellido: string;
    email: string;
    pacientes: Paciente[];
    token: string;
  }
```

Figura 10 - a)Typescript alimento.ts, b)Typescript Anatomia.ts, c)Typescript cita.ts, d) typescript dieta.ts, e) métricas.ts, f) paciente.ts, g)patología.ts, h)profesional.ts

6. Arquitectura de la aplicación

En este capítulo se describe la arquitectura software que se ha empleado en la aplicación desarrollada, así como la propuesta de diseño implementada y cada una de las entidades que lo componen tanto en el lado del cliente como en el lado del servidor.

6.1. Arquitectura de la aplicación

La aplicación desarrollada es una aplicación web que sigue un modelo MVC[2], esto quiere decir que los datos de la aplicación, la interfaz gráfica de usuario y la lógica responsable del control de la aplicación se encuentran separados entre sí, siendo completamente independientes unos de otros. La comunicación entre el servidor y el cliente se produce de manera asíncrona. De este modo el front-end, se construye mediante bloques, de manera que cada bloque contiene tres archivos que representan en el modelo MVC la vista y el controlador.

- **HTML:** Es el responsable de la vista y es el archivo HTML que visualizará el cliente. En este fichero se pueden incluir todas las etiquetas correspondientes a un archivo HTML, además de algunas etiquetas propias que proporciona Angular para facilitar la tarea de desarrollo.
- **CSS:** Representa la hoja de estilo aplicada al bloque en concreto. Todos los estilos aplicados en este fichero serán visibles en la vista del bloque. En este archivo se definen únicamente los atributos de estilo que tendrá la vista.
- **Typescript:** Representa el controlador del bloque. En este archivo se encuentran todas las funcionalidades, métodos y funciones que permiten obtener, modificar y guardar los datos generados por el usuario, o por la propia aplicación. Es el responsable de la comunicación de doble vía entre el código HTML y el modelo de datos. Esto significa que cualquier dato representado en la vista puede tener una representación en el controlador. De esta manera, es posible realizar cualquier cambio en la vista en tiempo real a través de funciones y métodos del controlador.

La aplicación se compone de varios bloques que comparten información entre ellos. En el proyecto, se ha establecido el modelo de datos en un directorio denominado *"model"* el cual contiene la representación de la estructura de datos utilizada. El archivo HTML junto con el archivo CSS de un bloque, forman parte de la vista, encargada de recibir las peticiones del usuario. El controlador está representado por los archivos Typescript de cada bloque y responden a los cambios y peticiones de la vista. El controlador modifica el modelo de la forma solicitada por el usuario a través de la vista. Una vez actualizado el modelo, los datos se verán reflejados en la vista gracias a la comunicación de doble vía que proporciona Angular.

[2] Modelo Vista Controlador

6.2. Front-end y Back-end

En el front-end, a través de Angular, se ha creado una clase que implementa un patrón de diseño DAO[3] encargada de realizar llamadas a la API alojada en el servidor (back-end), y de esta manera devolver la información proporcionada por las bases de datos a las clases que lo soliciten, de forma correctamente formateada. Del mismo modo, se empleará la clase para guardar la información en la base de datos.
En el back-end y como parte de una aplicación del tipo cliente – servidor, se dispone de un servidor web Apache capaz de ejecutar código PHP y una base de datos MariaDB instalada en el mismo servidor, encargada de almacenar y proporcionar los datos necesarios para el correcto funcionamiento de la aplicación. Por este motivo y para facilitar el trabajo se ha creado una API[4] en el lenguaje PHP encargada de comunicarse con la base de datos mediante procedimientos y funciones codificadas en la propia API.
De esta manera, el servidor Apache recibirá peticiones de la clase que gestiona el DAO en los clientes desde el front-end hasta la API en el servidor. La API se encargará de obtener la información solicitada y almacenada en la base de datos, y la devolverá a la clase que gestiona el DAO en el cliente.

La *Figura* 11, muestra gráficamente la arquitectura de la aplicación implementada:

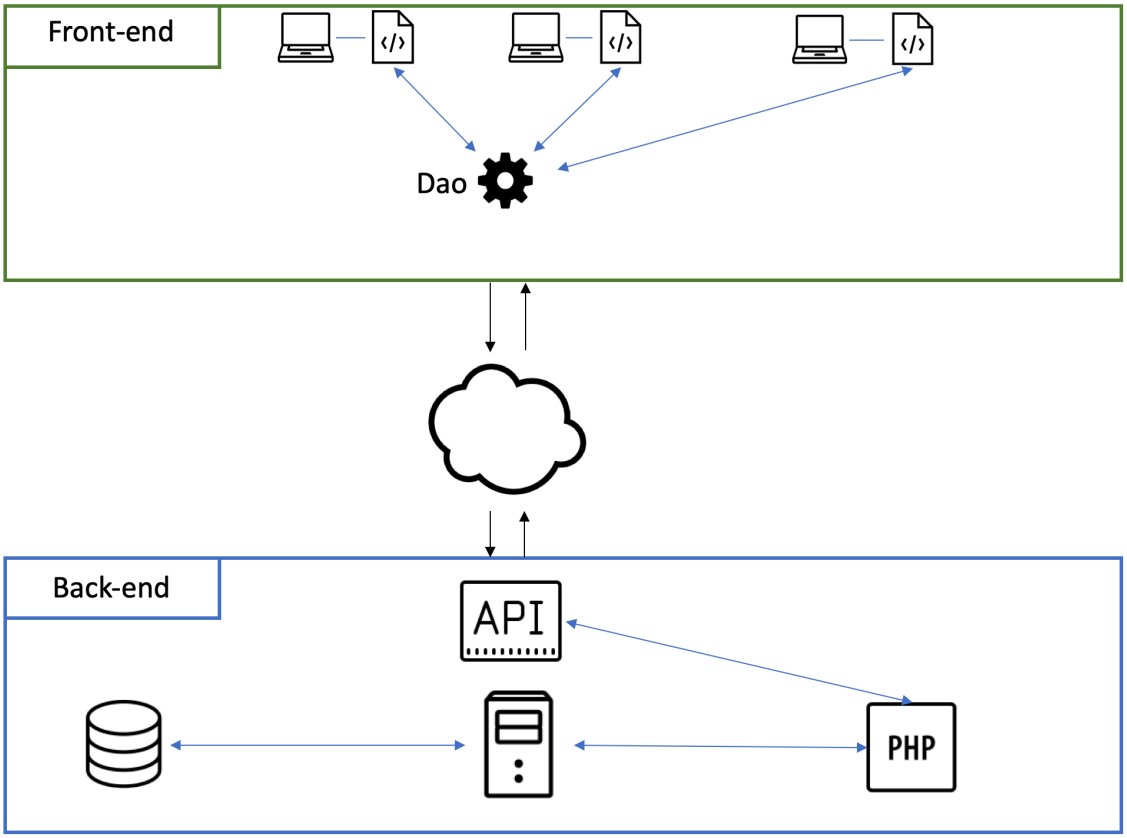


Figura 11 - Arquitectura

[3] Data Transfer Object
[4] Application Programming Interface

7. Diseño de la aplicación

En este capítulo, se describe la funcionalidad implementada en la aplicación, así como todos los detalles acerca del diseño de la aplicación. Se divide en tres apartados que corresponden al diseño de la parte del cliente (front-end), el patrón de diseño DAO, para la comunicación entre el cliente y el servidor, y la parte del servidor (back-end).

7.1. Front-end

El front-end se ha desarrollado en su totalidad mediante el framework Angular. Para ello se ha creado catorce bloques de código y un servicio para el patrón de diseño DAO. Cada bloque se encuentra en un directorio con el nombre del bloque conteniendo tres tipos de archivos que conforman el bloque: archivos HTLM, archivos CSS y archivos Typescript. La única excepción es el servicio DAO que contiene únicamente un archivo Typescript. Observar que se ha utilizado el lenguaje Typescript a lo largo de todo el desarrollo front-end para realizar transformaciones sobre el modelo de datos. A continuación, se describen los bloques.

7.1.1. Registro Profesional

Para poder registrar a un profesional en la aplicación, es imprescindible acceder a esta funcionalidad. Se accede a través de la pantalla principal *Log-in* (*Figura 12.a)*, mediante un enlace denominado *"Tengo un código"*. Al pulsar sobre este enlace, se cargará una vista HTML que solicitará un código de registro.

a) b)

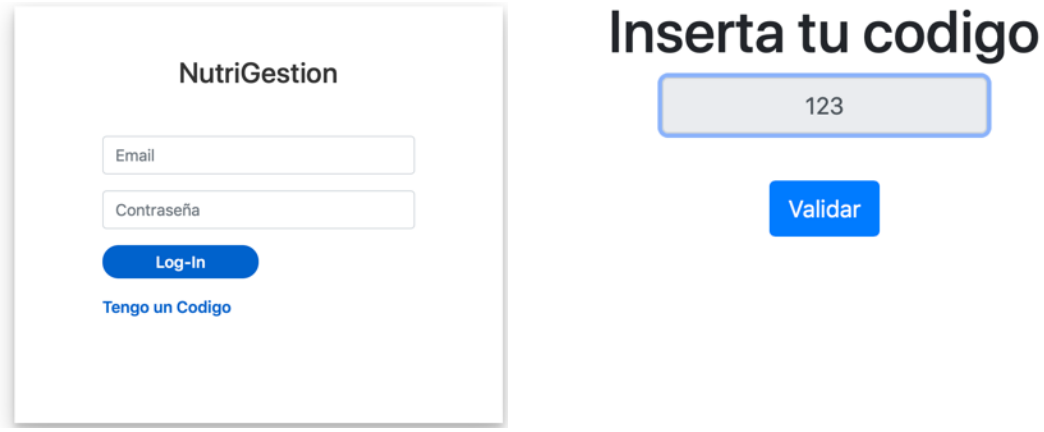


Figura 12 - a)Vista login, b)Vista insertar código

Tras introducir un código válido (*Figura 12.b)*, la aplicación cargará la vista que habilita para introducir los datos de un profesional. Tras validar los datos, se guardarán en la base de datos y se volverá a cargar la vista principal (*Figura 12.a*) para poder realizar la acción de Log-in con las credenciales creadas.

7.1.2. Log-in

Una vez se dispone de una cuenta profesional o de paciente, se puede realizar el log-in para acceder a la vista principal del profesional, si se realiza un log-in con una cuenta profesional, o la vista principal del paciente, si se realiza un log-in de paciente. Esto se realiza introduciendo las credenciales deseadas y validando los datos.
Una vez que la aplicación comprueba que las credenciales son correctas, se cargará la vista correspondiente.

a)

```
    text-align: center;
    color: #333;
}

.login-container{
    margin-top: 10%;
    margin-left: 35% ;

}
.form{
    padding: 5%;
    box-shadow: 0 5px 8px 0 rgba(0, 0, 0, 0.2), 0 9px 26px 0 rgba(0,
}

.login-container form{
    padding: 10%;
}
.btnSubmit
{
    width: 50%;
    border-radius: 1rem;
    padding: 1.5%;
    border: none;
    cursor: pointer;
}
.form .btnSubmit{
    font-weight: 600;
    color: #fff;
    background-color: #0062cc;
}

.form .ForgetPwd{
    color: #0062cc;
    font-weight: 600;
    text-decoration: none;
}
```

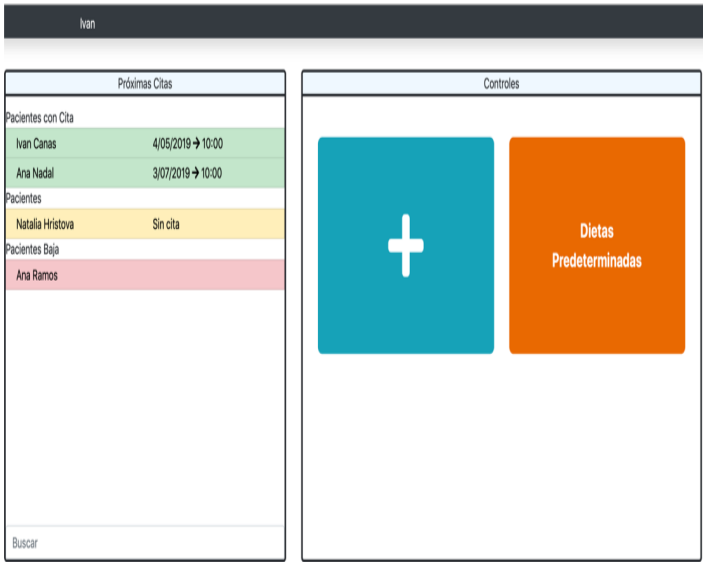
b)

```
email: string;
password: string;
valido = true;

logIn() {

  this.daoService.login(this.email, this.password).subscribe(
    R => {
      // let profesional = Object.assign(new Profesional(), R);  // aqui te
      if (!R) {
        this.valido = false;

      } else if (R.rol === 'profesional') {  // Login correcto, devuelve al
        const profesional = new Profesional();
        profesional.nombre = R.nombre;
        profesional.apellido = R.apellido;
        profesional.email = R.email;
        profesional.id = R.id;
        sessionStorage.setItem('token', R.token);
        // this.messenger.sendProfesional(profesional);
        this.route.navigate(['principal']);

      } else if (R.rol === 'paciente') {  // Login correcto, devuelve al pa
        const paciente = new Paciente();
        paciente.nombre = R.nombre;
        paciente.apellido = R.apellido;
        paciente.email = R.email;
        paciente.id = R.id;
        paciente.emailProfesional = R.emailProfesional;
        sessionStorage.setItem('token', R.token);
        this.route.navigate(['principalPaciente']);
      }
    }
    );
```

Figura 13 - a)CSS vista login, b)Typescript login

Observar en la *Figura 13.b* que el método *"logIn()"* pertenece al objeto de la clase DAO que se encarga de solicitar la información necesaria al servidor. Tras comprobar que las credenciales son correctas, se genera la clase correspondiente al rol de la entidad que ha solicitado el log-in (Profesional o Paciente) y se guarda en la sesión del navegador, el *"token"* de la entidad que ha obtenido de la base de datos para trabajar con ella más adelante. La *Figura 13.a* muestra el código CSS de la vista de log-in [2]. Una vez realizado el log-in, se cargará la vista representada en la *Figura 14.a* si el usuario es un profesional, o se cargará la vista representada en la *Figura 14.b* si el usuario es un paciente.

a)



b)

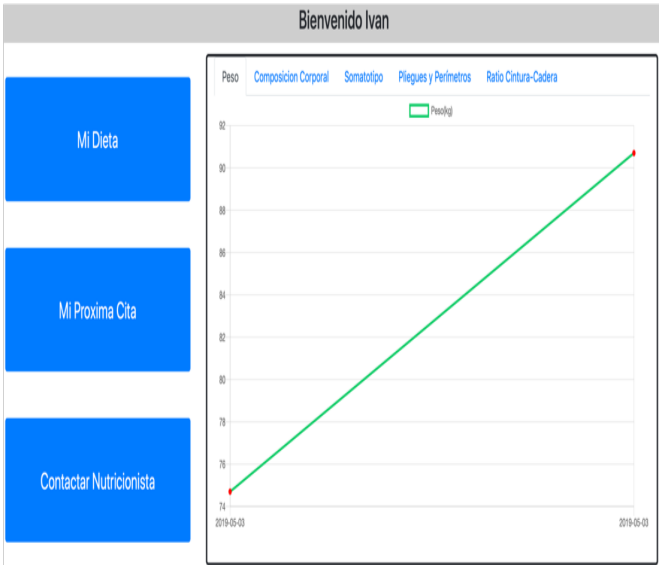


Figura 14 - a)Vista principal profesional, b) Vista principal paciente

7.1.3. Alta paciente

Una vez se accede a la aplicación como profesional, se carga la vista del profesional (*Figura 14.a)*. En esta vista, es posible dar de alta en el sistema más de un paciente. Para ello se debe pulsar sobre el botón "+" y aparecerá un elemento de tipo modal en la vista con todos los campos a rellenar necesarios para dar de alta a un nuevo paciente, este elemento modal aparece en la *Figura 15.c*. Esta acción pertenece al bloque denominado *"principal"* y se encuentra en el directorio con el mismo nombre que el bloque.

Existe un objeto "*nuevoPaciente"* de tipo *Paciente* en el controlador de la vista cuyos atributos se pueden modificar desde la vista gracias a la comunicación de doble vía que proporciona Angular. De este modo, a través del modal, se proporciona valor a los atributos del objeto para obtener un nuevo paciente completamente formateado en el modelo, el cual será posteriormente guardado en la base de datos a través del objeto DAO mediante el método *"insertaPaciente(Profesional, nuevoPaciente)"* mostrado en la *Figura 15.b.*

a)

```
<!-- Modal -->
<ng-template #template>
  <div class="modal-header">
    <h4 class="modal-title pull-left">Nuevo Paciente</h4>
    <button type="button" class="close pull-right" aria-label="Close" (click)="modal
      <span aria-hidden="true">&times;</span>
    </button>
  </div>
  <div class="modal-body">
      <div class="nuevoPacienteContainer ">
          <div class="row">
            <div class="col-md-12 form">
                <div>
                    <input type="text"
                     class="form-control" placeholder="Nombre"
                      name="nombre" [(ngModel)] = nuevoPaciente.nombre >
                    <input type="text"
                     class="form-control" placeholder="Apellido 1"
                      name="apellido" [(ngModel)] = nuevoPaciente.apellido >
                    <input type="text"
                    class="form-control" placeholder="Apellido 2"
                    name="apellido2" [(ngModel)] = nuevoPaciente.apellido2 >
                    <input type="numeric"
                    class="form-control" placeholder="Edad"
                    name="edad" (keydown)="checkNumber($event)" [(ngModel)] = nuevoP
                    <select class="form-control" [(ngModel)] = nuevoPaciente.sexo>
                        <option value="h">Hombre</option>
                        <option value="m">Mujer</option>
                      </select>
                    <input type="text"
                    class="form-control" placeholder="Email"
                    name="email" [(ngModel)] = nuevoPaciente.email >
                    <input type="password" class="form-control"
                    placeholder="Password" name="pass" [(ngModel)] = nuevoPaciente.pa
                </div>
                <div class="form-group">
                    <input type="button" class="btnSubmit" value="Crear" (click)="inse
                </div>
```

b)

```
insertaPaciente() {

  this.daoService.insertaPaciente(this.profesional, this.nuevoPaciente
    R => {

      if (!R) {

        alert('Error');

      } else {  // Login correcto, devuelve al profesional
        // window.location.reload();
        this.nuevoPaciente.emailProfesional = this.profesional.email;
        this.modalRef.hide();
        localStorage.removeItem('Paciente');
        localStorage.setItem('Paciente', JSON.stringify(this.nuevoPac
        this.route.navigate(['detallePaciente']);
      }
    }
    );
```

c)

Figura 15 - a)HTML alta paciente, b) Typescript alta paciente, c)Vista alta paciente

Una vez introducido el nuevo paciente en la base de datos, se almacenará en la sesión del navegador del cliente, el nuevo paciente creado como se aprecia en la *Figura 15.b*. Tras crearse el nuevo paciente y almacenarse el paciente en la sesión, la aplicación cargará la vista correspondiente a la ficha del nuevo paciente denominada "paciente-*detail",* otorgándole a esta vista desde el controlador, los datos del nuevo paciente almacenados en la sesión por la vista anterior, y asignándoselo así a su atributo correspondiente "*paciente".* La *Figura 15.a* representa el código HTML del elemento modal desplegado para la creación de un paciente.

7.1.4. Baja paciente

Esta funcionalidad, se lleva a cabo en el bloque perteneciente al directorio denominado "*paciente-detail".* La *Figura 16* muestra el código implementado en la aplicación, encargado de realizar la llamada al objeto DAO para dar de baja a un paciente.

```typescript
desactivarPaciente() {

  const respuesta = prompt('Escribe "si" para confirmar la desactivacion del paciente');
  console.log(this.paciente);

  if (respuesta === 'si') {

    this.daoService.desactivaPaciente(this.paciente).subscribe(
      R => {
        if (!R) {
          alert('Imposible eliminar paciente');
        } else {
          alert('Paciente desactivado');
          this.route.navigate(['principal']);
        }
      }
    );
  } else {
    alert('El paciente no se desactivará');
  }
```

Figura 16 - Typescript baja paciente

7.1.5. Añadir cita a paciente

Para añadir una nueva cita, debe estar cargada la vista del paciente al cual se le vaya a asignar una nueva cita. A fin de llevar a cabo esta acción se debe pulsar el botón verde cuyo texto es "*Nueva Cita*" que desencadenará la carga de un formulario de tipo modal (*Figura 17.b)* el cual contiene los campos que forman el objeto "Cita" que se encuentra en el controlador. Con el objetivo de rellenar los datos de la fecha, se dispone de una herramienta del tipo "*date picker*" que mejora la forma de introducir una fecha concreta por el usuario.
Se le debe asignar también una hora. Una vez rellenos los campos de la cita, se pulsará el botón "*Nueva Cita*" del modal, que desencadenará la ejecución del método "*addCita()*". Este método será el encargado de guardar los datos de esta nueva cita en la base de datos y asignarla al paciente como aparece en la *Figura 17.a*.

a)

```typescript
addCita() {

  this.fecha.setHours(this.hora);
  this.fecha.setMinutes(this.minuto);
  this.fecha.setSeconds(0);
  this.cita.fecha = this.fecha ;


  this.daoService.addCita(this.paciente, this.cita).s
    R => {

      if (!R) {
        alert('Fecha Errónea');
      } else {
        alert('Nueva Cita para ' + this.fecha);
        window.location.reload();
      }

  });
```

b)

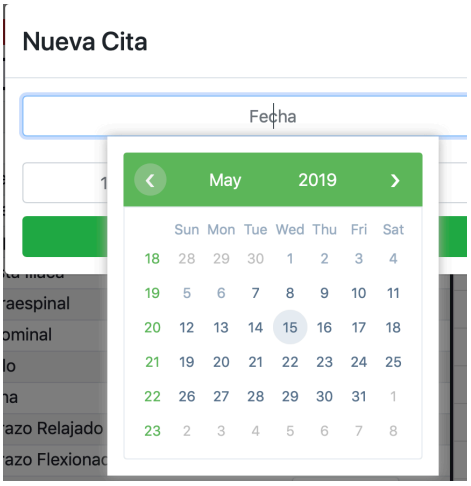


Figura 17 - a) Typescript añadir cita, b) Vista añadir cita

Tras insertar la cita y asignarla al paciente en la base de datos, se mostrará un cuadro emergente confirmando la fecha y hora de la cita. Este método se encarga de realizar la llamada correspondiente al objeto DAO, pasando como atributos el paciente al que se le va a añadir la cita, y el objeto Cita cuyos atributos contienen los datos de la nueva cita (*Figura 17.a)*.

7.1.6. Cancelar cita a paciente

Para cancelar una cita a un paciente, debe estar cargada la vista del paciente al que se desea cancelar su cita. Para desencadenar la acción se debe pulsar sobre el icono con forma de cruz roja que se encuentra a la derecha de la cita. Este método es el encargado realizar una llamada al objeto DAO y realizar la petición de cancelación de cita al servidor. Como se aprecia en la *Figura 18,* este objeto DAO lleva como atributo la cita que se quiere cancelar y que contiene el email del paciente al cual pertenece la cita. Si la acción tiene éxito, aparecerá una ventana emergente confirmando la cancelación de la cita.

```
cancelCita(cita: Cita) {

  if (cita.fecha === undefined) {
    alert('El paciente no tiene citas activas');
    return;
  }
  const dateString = cita.fecha.toString();
  const date = new Date(dateString);
  date.setMonth(date.getMonth() - 1);
  date.setHours(date.getHours() + this.desfase);
  const citaLocal = new Cita();
  citaLocal.email = cita.email;
  citaLocal.activo = cita.activo;
  citaLocal.fecha = date;

  this.daoService.cancelCita(citaLocal).subscribe(
    R => {
      if (!R) {
        alert('Imposible eliminar la cita');
      } else {
        alert('Cita cancelada');
        window.location.reload();
      }
    }
```

Figura 18 - Typescript cancelar cita

Una cita, contiene en la base de datos un campo denominado "*activo"* si este bit se encuentra a 1, significa que la cita se encuentra activa, por tanto, aparecerá como cita activa en la lista de citas de un paciente. Si, por el contrario, el bit *"activo"* se encuentra a 0, quiere decir que la cita ha sido cancelada y no forma parte de las citas activas en el paciente. Cancelar una cita, producirá un cambio en la base de datos, su bit activo de 1 a 0. De esta forma se mantiene un histórico de las citas al no borrarse en la base de datos, si no que cambia de estado el bit *"activo"*.

7.1.7. Buscar paciente

Esta funcionalidad, se encuentra implementada en el bloque de la vista principal del profesional denominado "*pacientes-list"*. Es la encargada de buscar un paciente en cualquiera de las tres listas de pacientes (con cita, activos, baja). Existe un cuadro de texto que se encuentra en la parte inferior de la vista principal. Al pulsar cualquier tecla sobre este bloque se desencadena la llamada a la función "*busca()*" que aparece en la *Figura 19*. Esta función utiliza el texto introducido en el cuadro de texto para buscar el contenido en las tres listas de pacientes al mismo tiempo. Para ello, se sirve de la función de typescript denominada *"filter()"* la cual permite encontrar en la lista de pacientes, texto coincidente con los atributos o funciones que se introduzcan. De este modo y en tiempo real, desaparecerán de las tres listas de pacientes, aquellos que no cumplan las condiciones de coincidencia. La búsqueda se realiza al comparar el texto escrito en el cuadro de texto con los nombres y los apellidos de los pacientes.

```
busca() {

  this.listaPacientesCita = this.listaPacientesCitaCopy.filter(
    F => {
      return F.nombre.toLowerCase().includes(this.textoBusqueda.toLowerCase()) ||
      F.apellido.toLowerCase().includes(this.textoBusqueda.toLowerCase());
    }
  );

  this.listaPacientesActivo = this.listaPacientesActivoCopy.filter(
    F => {
      return F.nombre.toLowerCase().includes(this.textoBusqueda.toLowerCase()) ||
      F.apellido.toLowerCase().includes(this.textoBusqueda.toLowerCase());
    }
  );

  this.listaPacientesBaja = this.listaPacientesBajaCopy.filter(
    F => {
      return F.nombre.toLowerCase().includes(this.textoBusqueda.toLowerCase()) ||
      F.apellido.toLowerCase().includes(this.textoBusqueda.toLowerCase());
    }
  );

}
```

Figura 19 - Typescript buscar

7.1.8. Añadir medidas y métricas al paciente

Es el resultado de haber obtenido unas medidas en una consulta y por ello, estas medidas pertenecen a un paciente. Esta funcionalidad, pertenece al bloque *"medidas"* que se encuentra en el directorio del proyecto con el mismo nombre y su función es la de añadir medidas en la ficha de un paciente en concreto. La estructura que sigue la vista tal y como aparece en la Figura 20.c es la de una tabla que contiene múltiples filas con el nombre de la medida a introducir, y un campo de texto donde se introducirá dicha medida. La codificación en HTML de la vista se representa en la Figura 20.a. Queda restringida la inserción en los campos de texto de cualquier carácter distinto de un número, tabulador o un punto. Una vez introducida una nueva medida en cualquiera de los campos, en controlador automáticamente llamará al componente de las métricas para recalcular todas las métricas en función de la nueva medida introducida. Al depender de manera absoluta, las medidas de las métricas del paciente, en esta funcionalidad se recalcularán las métricas y se guardarán en la base de datos, tanto las medidas introducidas, como las métricas calculadas sobre esas medidas. Los métodos "recalculaMetricas()" y "actualizaMedidas()"que aparecen en la *Figura 20.b*, se encargan de llevar a cabo estas dos acciones. Se dispone también de un elemento HTML que permite seleccionar de una lista todas las medidas correspondientes a dicho paciente. Una vez seleccionada de esta lista una medida, se recalcularán las métricas correspondientes a los datos de las medidas seleccionadas, y así se crea la capacidad de editar cualquiera de estos campos. En tiempo real, aparecerá en el bloque de métricas, estas métricas recalculadas.

a)

```
<tbody>
  <tr class="col-sm-12">
    <td class="col-sm-6 medida-nombre">Peso</td>
    <td class="col-sm-5 medida-dato"> <input (keydown)="ch
      (keyup)="recalculaMetricas(anatomia)" type="numeric
      class="form-control medidaInput" [(ngModel)] = ana
    <td class="col-sm-1 medida-unidades">Kg</td>
  </tr>
```

b)

```
actualizaMedidas() {
  this.daoService.actualizaMedidas(this.paciente, this.anat
    Med => {
      const metricas = new Metricas(this.paciente, this.ana
      // console.log(metricas);
      this.daoService.actualizaMetricas(this.paciente, metr
        Met => {
          window.location.reload();
        }
      );
    }
  );

}

recalculaMetricas(anatomia: Anatomia) {
  this.recalculaEvent.emit(anatomia);
}

check($event) {
  let caracter;
  caracter = $event.keyCode;
  return ((caracter > 47 && caracter < 58) || caracter ===
          caracter === 37 ||
          caracter === 39 || caracter === 9 || caracter ===
          caracter === 38 || caracter === 40;
}
```

c)

Medidas		
Peso	74.700	Kg
Altura	180.000	cm
Pliegue Triceps	10.000	mm
Pliegue Subescapular	16.000	mm
Pliegue Biceps	9.000	mm
Pliegue Cresta Iliaca	10.000	mm
Pliegue Supraespinal	12.000	mm
Pliegue Abdominal	16.000	mm
Pliegue Muslo	8.000	mm
Pliegue Pierna	7.000	mm
Perimetro Brazo Relajado	33.000	cm
Perimetro Brazo Flexionado	34.000	cm
Perimetro Cintura	81.000	cm
Perimetro Cadera	92.000	cm
Perimetro Pierna	38.000	cm
Perimetro Muñeca	14.000	cm
Diametro Humero	7.000	cm

2019-05-03 13:27:31

Actualiza Medidas/Guardar

Metricas		
IMC	23,056	Normopeso
Ratio Cintura cadera	0,880	Adecuado
Suma 6 pliegues	69,000	
Suma 8 pliegues	88,000	
% Grasa	9,837	
% Óseo	25,353	
% Músculo	40,710	
% Residual	24,100	
Masa Grasa	7,348	
Masa Osea	18,939	
Masa Muscular	30,411	
Masa Residual	18,003	
Somatotipo	Mesomorfo	
Endomorfo	3,803	
Mesomorfo	5,145	
Ectomorfo	2,706	

Figura 20 - a)HTML medidas, b)Typescript medidas, c)Vista medidas y métricas

El método *"actualizaMedidas()"* es el encargado de solicitar a la clase DAO el almacenamiento en el servidor de los datos introducidos. Para ello, se le pasa como argumentos las medidas recogidas de la vista y el paciente al que pertenecen. Así mismo este método guarda en la base de datos a través de la clase DAO las métricas calculadas para ese paciente. Por otro lado, el método *"recalculametricas()"* es responsable de generar un elemento del tipo *"EventEmitter"* cuya responsabilidad es la de enviar estas medidas al bloque de métricas para calcularlas a través de las medidas recibidas.

7.1.9. Añadir/Eliminar patología al paciente

Esta funcionalidad tiene como objetivo añadir una o varias patologías en la base de datos, a un paciente en concreto. Esta acción se lleva a cabo desde la ficha del paciente y pertenece al bloque llamado *"paciente-detail"*. Para ello, se debe pulsar en un botón amarillo cuyo texto es

"*Patologías"* que ejecuta la carga de un formulario de tipo modal mostrado en la *Figura 21.c* con todas las patologías almacenadas en la base de datos, pertenecientes a todos los profesionales y al profesional que ha accedido. Gracias al enlace de dos vías que proporciona Angular, la lista de patologías visible en el modal, pertenece al atributo del bloque de patologías denominado *listaPatologias.* Las patologías que pertenecen al paciente en concreto aparecen representadas con un botón deslizable azul, y representa al atributo array de patologías denominado en el controlador como *patologiasPaciente*. Se pueden añadir patologías disponibles de la lista al array de patologías del paciente, simplemente deslizando el botón deslizable de selección, quedando de esta manera incluidas en el array. Como muestra la *Figura 21.b,* la correcta inserción o extracción del array *"patologiasPaciente",* lo lleva a cabo de manera segura la función *"changePatologias()"* que es llamado cada vez que se mueve algún botón deslizable del modal. Una vez terminado de introducir las patologías que se quieren asignar al paciente, se deberá pulsar el botón guardar. Este botón ejecuta la llamada al método *actualizaPatologias()* que se encarga de ponerse en contacto con el DAO de manera que quedarían guardadas las patologías en la base de datos y asociadas al paciente. La *Figura 21.a* muestra el código CSS que permite utilizar los selectores en forma de deslizable de color azul [18].

a)

```css
/* El switch del modal */
.switch {
    position: relative;
    display: inline-block;
    width: 60px;
    height: 34px;
    float:right;
  }

  /* Escondo el por defecto*/
  .switch input {display:none;}

  /* Slider */
  .slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #ccc;
    -webkit-transition: .4s;
    transition: .4s;
  }

  .slider:before {
    position: absolute;
    content: "";
    height: 26px;
    width: 26px;
    left: 4px;
    bottom: 4px;
    background-color: white;
    -webkit-transition: .4s;
    transition: .4s;
  }
```

b)

```typescript
changePatologias(event, patologia: Patologia) {

  if ( event.target.checked ) { // Añado el elemento
    this.patologiasPaciente.push(patologia);
  } else {  // Quito el elemento
    const posicion = this.patologiasPaciente.findIndex(
      F => F.nombre === patologia.nombre
      );
    console.log(posicion);
    this.patologiasPaciente.splice(posicion, 1);
  }

  this.isDisabled = true;

}

actualizaPatologias() {
  this.daoService.actualizaPatologias(this.paciente, this.patologiasPaciente).subscribe(
    R => {
      window.location.reload();
    }
  );
}
```

c)

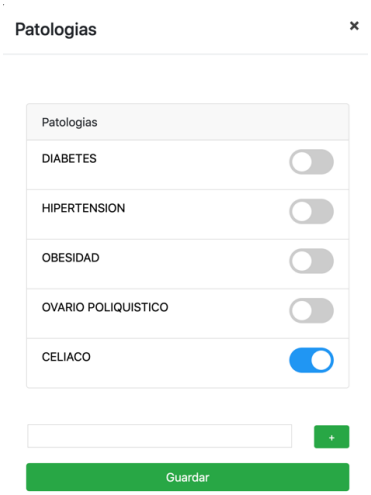


Figura 21 - a)CSS patologías, b)Typescript patologías, c) Vista patologías

7.1.10. Añadir patología a la lista de patologías

La aplicación permite introducir manualmente diferentes nombres de patologías a la base de datos. Estas patologías serán introducidas por un profesional, y dentro de la base de datos se guardarán junto con su correo electrónico. De esta forma, se asegura que la patología añadida por parte del profesional le pertenece única y exclusivamente a él no estando disponible para otros profesionales. En el objeto modal descrito en *"7.1.9 Añadir/Eliminar patología al paciente"*, se encuentra un campo de texto y un botón "+". Para introducir una nueva patología, basta con introducir el nombre de la nueva patología y pulsar el botón "+". De esta manera, se tiene disponible la patología en una lista para ser marcada.

```
agregarNuevaPatologia() {
  this.nuevaPatologia.nombre = this.nuevaPatologia.nombre.toLowerCase();
  this.daoService.insertaPatologia(this.nuevaPatologia, this.paciente.emailProfesional).subscribe(
    R => {
      if (!R) {
        alert('La patologia ya existe');
      } else {
        this.daoService.getListaPatologias(this.paciente.emailProfesional).subscribe(
          L => {
            this.listaPatologias = L;
            this.getListaPatologias();
          }
        );

      }
    }
  );
```

Figura 22 - Typescript añadir patología

De esta forma, al pulsar sobre el botón "+" se ejecuta la llamada al método *"agregarNuevaPtologia()"* mostrada en la *Figura 22* encargado de realizar una llamada a la clase DAO responsable de insertar esta nueva patología y asociarla al email del profesional que la ha creado. Si la patología ya existe, se mostrará una ventana emergente indicándolo.

7.1.11. Ver progreso en las gráficas del paciente

Esta funcionalidad permite al profesional ver la progresión de histórico de medidas y métricas de un paciente de forma gráfica. Para ejecutar esta funcionalidad, el profesional debe pulsar el botón azul cuyo texto es "*Progreso*" que ejecutará una llamada a la función *"progreso()"* perteneciente al bloque "p*aciente-detail"*. Este método realiza una petición a la clase DAO para obtener todas las medidas y métricas de un paciente y posteriormente guardarlas en el almacenamiento local de la sesión como se muestra en la *Figura 23*.
Una vez almacenados estos datos en la sesión, la aplicación cargará la vista *"progreso",* perteneciente al bloque del mismo nombre.

```
progreso() {
  localStorage.removeItem('anatomiaList');
  localStorage.removeItem('metricasList');
  localStorage.setItem('anatomiaList', JSON.stringify(this.anatomiaList));

  this.daoService.getMetricas(this.paciente).subscribe(
    R => {
      if (R === null) {
        alert('El paciente no contiene datos');
      } else {
        let metricasList: Metricas[] = [];
        metricasList = R;
        localStorage.setItem('metricasList', JSON.stringify(metricasList));
        this.route.navigate(['progreso']);
      }
    }
```

Figura 23 - Progreso paciente

Una vez cargada la vista de progreso, el controlador recuperará los datos almacenados en la sesión y le asignará estos datos a los arrays cuyos nombres son "*anatomiaList"* de la clase *"Medidas"* para el histórico de medidas y "*metricasList"* de la clase *"Metricas"* para el histórico

de métricas. Con todos estos datos, y con la ayuda de las librerías disponibles para representar gráficos de Chart.js, se puede realizar la representación de las gráficas deseadas. Para la representación gráfica, se observa en la *Figura 24.a* la necesidad de creación de un objeto tipo *"canvas"* al que se le asignan atributos que definen la apariencia y comportamiento de las gráficas. Los datos de las medidas y las métricas se proporcionan a los canvas creados con el formato y el orden deseado tal como muestra la *Figura 24.b*.

a)

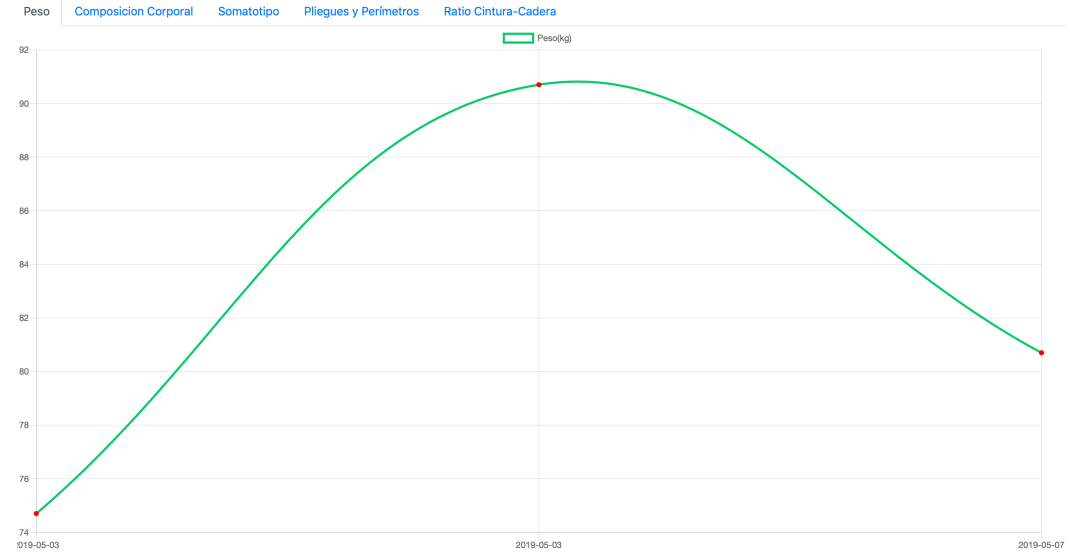
```
<tabset>
  <tab heading="Peso"  class="row"  id="tab1">

    <div class="col-sm-12" id="chartContainer">
        <canvas id="peso"></canvas>
    </div>

  </tab>
  <tab heading="Composicion Corporal">
    <div class="row">
        <div class="col-sm-6" id="chartContainer">
          <canvas id="composicion"></canvas>
        </div>
        <div class="col-sm-6" id="chartContainer">
          <canvas id="EvolucionCorporal"></canvas>
        </div>
    </div>

  </tab>
  <tab heading="Somatotipo">
    <div class="row">
        <div class="col-sm-12" id="chartContainer">
          <canvas id="somatotipo"></canvas>
        </div>
    </div>
  </tab>
```

b)

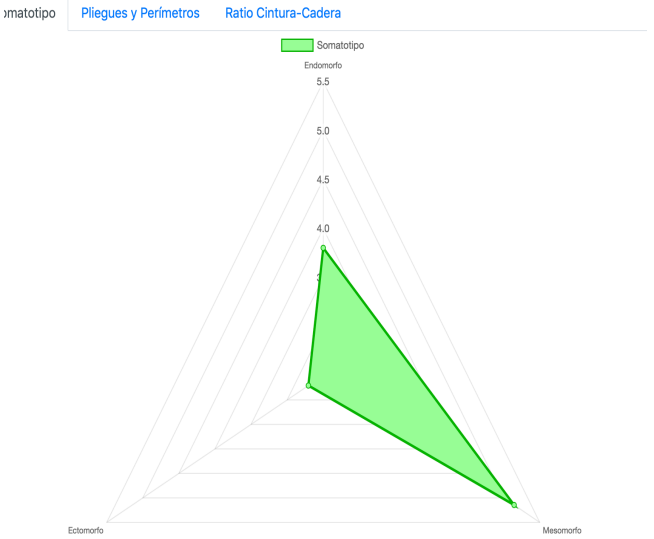
```
chartPeso() {

  let canvas: any;
  canvas = document.getElementById('peso');
  let ctx: any;
  ctx = canvas.getContext('2d');

  const labels: string[] = [];
  const dataset: number[] = [];

  this.anatomiaList.forEach(
    R => {
      labels.push(R.fechaModificacion.toString().substring(0, 10));
      dataset.push(Number(R.peso));
    }
  );

  const data = {
    labels,
    datasets: [{
        label: 'Peso(kg) ',
        backgroundColor: 'rgba(0, 0, 0, 0)',
        borderColor: 'rgb(0, 204, 102)',
        pointBorderColor: 'rgba(255, 0, 0, 1)',
        pointBackgroundColor: 'rgba(255, 0, 0, 1)',
        data:  dataset
    }]
  };

  const options = {
    responsive: true
  };
```

Figura 24 - a)HTML progreso, b) typescript progreso

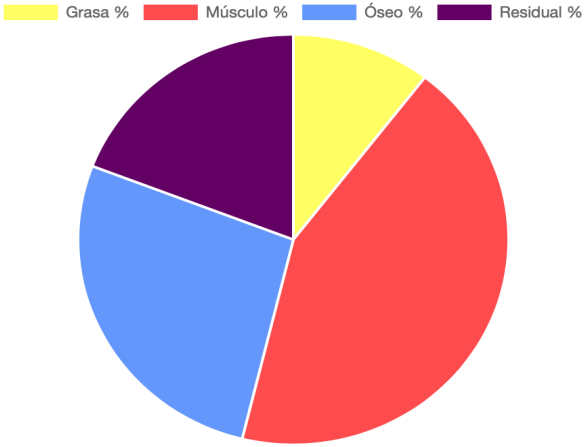
a)

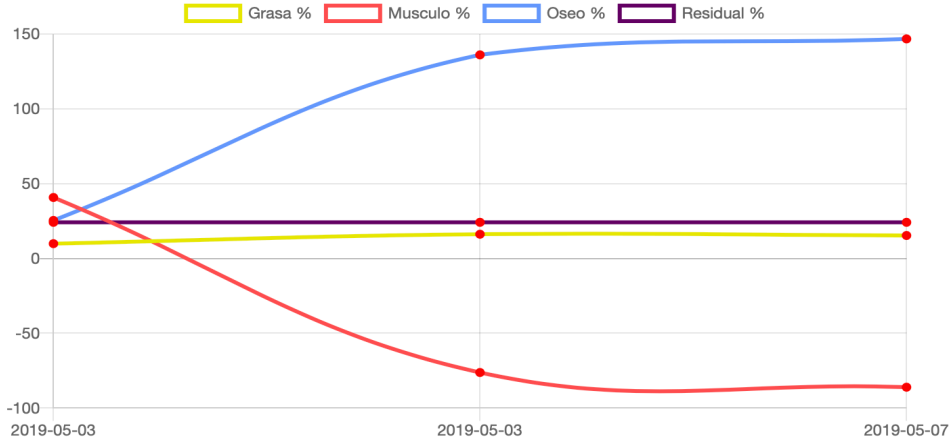

Peso
Composicion Corporal
Somatotipo
Pliegues y Perímetros
Ratio Cintura-Cadera
Peso(kg)
92
90
88
86
84
82
80
78
76
74
2019-05-03
2019-05-03
2019-05-07

b)

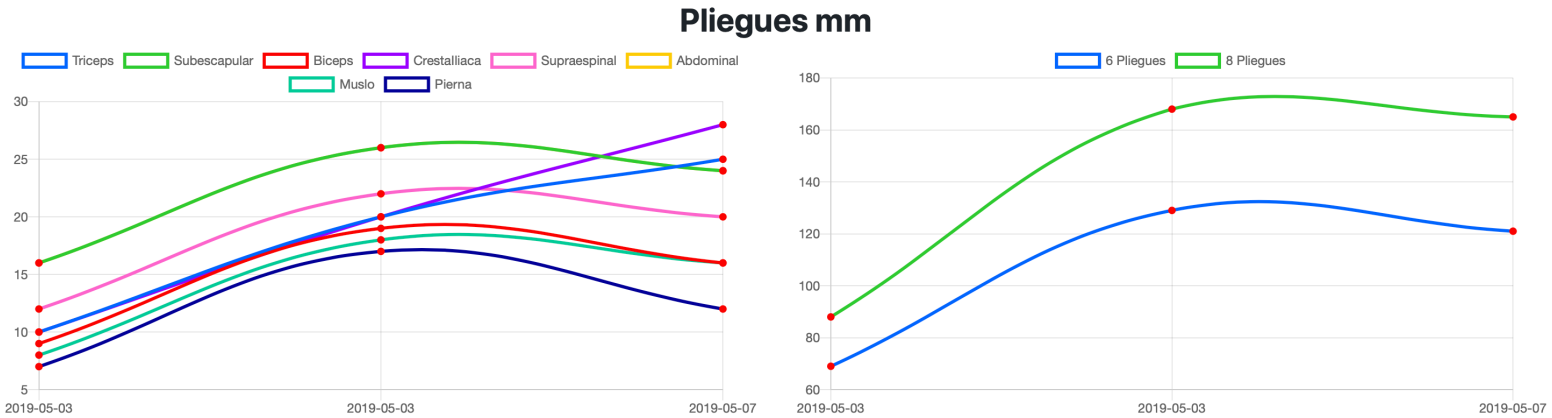

omatotipo
Pliegues y Perímetros
Ratio Cintura-Cadera
Somatotipo
Endomorfo
5.5
5.0
4.5
4.0
Ectomorfo
Mesomorfo

c)


Grasa %
Músculo %
Óseo %
Residual %


Grasa %
Musculo %
Oseo %
Residual %
150
100
50
0
-50
-100
2019-05-03
2019-05-03
2019-05-07

d)



e)

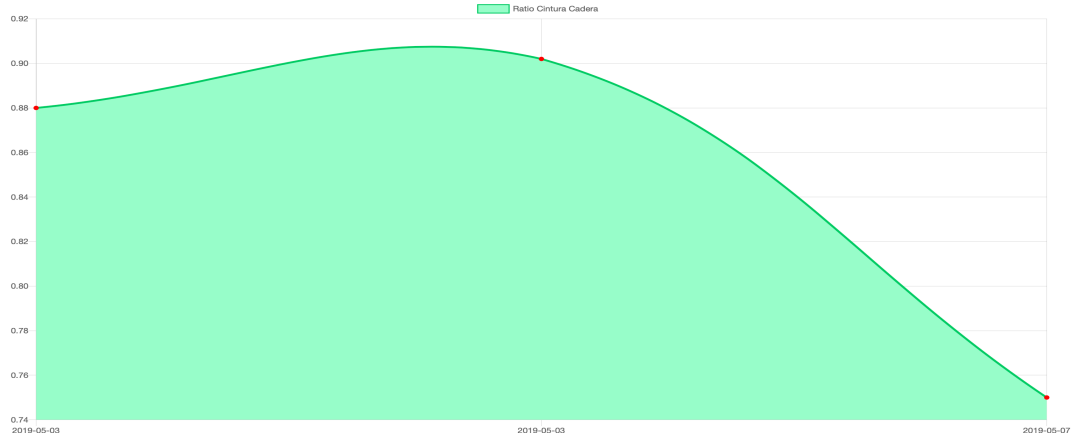


Figura 25 - Gráfica peso, b)Gráfica somatotipo, c)Gráfica composición corporal, d) Gráfica pliegues y perímetros, e) Gráfica ratio cintura-cadera

Las *Figuras 25.a, 25.b, 25.c, 25.d y 25.e* muestran las diferentes representaciones de gráficas implementadas en la aplicación utilizando las librerías de Chart.js modificando algunos de sus parámetros para obtener la representación deseada.

7.1.12. Añadir/Eliminar elementos a una dieta

Esta funcionalidad se encarga de añadir o eliminar una clase *"Alimento"* del array correspondiente a su franja horaria como muestra la *Figura 26.c* que corresponde a la vista generada por el archivo HTML mostrado en la *Figura 26.a*. Pertenece al bloque *"paciente-dieta"* y a través de ella, es posible añadir o eliminar alimentos a una dieta. Para añadir alimentos a una franja horaria es necesario pulsar el botón verde "+" y, por el contrario, para eliminar el último alimento, es necesario pulsar el botón rojo "- ". Al pulsar el botón verde "+" se ejecuta una llamada al controlador, al método *"mas()"* el cual se encargará de crear una nueva clase "Alimento" vacía y la añadirá al array de la franja horaria correspondiente como muestra la *Figura 26.b*. Mediante el enlace de doble vía de Angular, se pueden rellenar los atributos y campos correspondientes al alimento a través de la vista. Por el contrario, para eliminar un alimento, al pulsar sobre el botón rojo "-"se ejecuta una llamada al método "menos()" del controlador, el cual se encarga de eliminar el último alimento del array correspondiente a la franja horaria seleccionada como muestra la *Figura 26.d*. En cada pulsación de tecla en el campo "nombre" del alimento, se ejecutará una llamada a la clase DAO encargada de consultar en la base de datos si el alimento introducido ya existe. Si la clase DAO devuelve un alimento, significa que existe en la base de datos, por lo tanto, se obtendrá las unidades del alimento que ha devuelto la clase DAO y se añadirán estas unidades al campo *"unidades"* del alimento correspondiente.

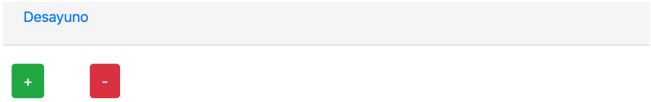
a)

```
<tab heading="Lunes" id="tab1" >
  <div>
    <accordion >
      <accordion-group heading="Desayuno" [isOpen]="isOpenLunes">
        <div *ngFor="let d of desayunoLunes" class="row">
          <div class="col-sm-2">
            <label>Alimento</label>
          </div>
          <div class="col-sm-2">
            <input (keyup)="onKeyUp(d)" [(ngModel)]="d.nombre"
            type="text" class="form-control" >
          </div>
          <div class="col-sm-2">
            <label>Cantidad</label>
          </div>
          <div class="col-sm-2">
            <input [(ngModel)]="d.cantidad"
            (keydown)="checkNumber($event)" value = "0" type="text"
            class="form-control" >
          </div>
          <div class="col-sm-2">
              <select [(ngModel)]="d.unidades" id="select"
              class="form-control" [class.greenBoxed]="d.id != undefined">
                <option *ngTemplateOutlet="unidades" value="d.unidades">
```

b)

```
mas(dia: string, franja: string) {
  switch (dia) {
    case 'Lunes':

      switch (franja) {
        case 'desayuno':
        this.desayunoLunes.push(new Alimento(franja));
        break;
        case 'postdesayuno':
        this.postDesayunoLunes.push(new Alimento(franja));
        break;
        case 'comida':
        this.comidaLunes.push(new Alimento(franja));
        break;
        case 'merienda':
        this.meriendaLunes.push(new Alimento(franja));
        break;
        case 'cena':
        this.cenaLunes.push(new Alimento(franja));
        break;
      }
```

c)



d)

```
menos(dia: string, franja: string) {

  switch (dia) {
    case 'Lunes':

      switch (franja) {
        case 'desayuno':
        this.desayunoLunes.pop();
        break;
        case 'postdesayuno':
        this.postDesayunoLunes.pop();
        break;
        case 'comida':
        this.comidaLunes.pop();
        break;
        case 'merienda':
        this.meriendaLunes.pop();
        break;
        case 'cena':
        this.cenaLunes.pop();
        break;
      }
```

Figura 26 - a)HTML alimento dieta, b)typescript añadir alimento, c) typescript eliminar alimento, d)Vista franja horaria

7.1.13. Crear dieta paciente

Esta funcionalidad, consiste en añadir varios alimentos en las diferentes franjas horarias de los distintos días de la semana. Una vez añadidos todos los alimentos deseados, se puede asignar un nombre a la dieta y tras pulsar el botón verde con el texto *"Guardar Dieta"* como muestra la *Figura 27.c,* se ejecuta así la llamada al método del controlador *"guardarDieta()"* encargado de transformar los alimentos de las diferentes franjas horarias y días en una clase del tipo *"Dieta"* formada por atributos del tipo array de alimentos como muestra la *Figura 27.b*. A continuación, se realiza una llamada a la clase DAO encargada de solicitar el almacenamiento de la dieta en el servidor, y asociarla al paciente al que pertenece. La *Figura 27.a* muestra el documento HTML que contiene la llamada al controlador "*guardarDieta()"* a través de la pulsación del botón verde "*Guardar Dieta"* que aparece en la *Figura 27.c*.

a)

```
<div class="row col-sm-12 botonera">
  <div class="col-sm-2">
    <input [(ngModel)]="dieta.nombre" placeholder="Nombre de la dieta"
    type="text" class="form-control" >
  </div>
  <div class="col-sm-2">
      <button (click)="guardarDieta()" type="button"
      class="btn btn-success">Guardar Dieta</button>
  </div>
  <div id="textoToggle" class="col-sm-1">
      Abrir Todos
  </div>
  <div class="col-sm-1 switch">
      <label class="switch">
        <input type="checkbox">
        <span (click)="expandCollapse()" class="slider round"></span>
        </label>
  </div>
</div>
```

b)

```
guardarDieta() {


  this.desayunoLunes.forEach(element => {
    element.nombre = element.nombre.toLowerCase(
    this.dieta.lunes.push(element);
  });
  this.postDesayunoLunes.forEach(element => {
    element.nombre = element.nombre.toLowerCase(
    this.dieta.lunes.push(element);
  });
  this.comidaLunes.forEach(element => {
    element.nombre = element.nombre.toLowerCase(
    this.dieta.lunes.push(element);
  });
  this.meriendaLunes.forEach(element => {
    element.nombre = element.nombre.toLowerCase(
    this.dieta.lunes.push(element);
  });
  this.cenaLunes.forEach(element => {
    element.nombre = element.nombre.toLowerCase(
    this.dieta.lunes.push(element);
  });
```

c)

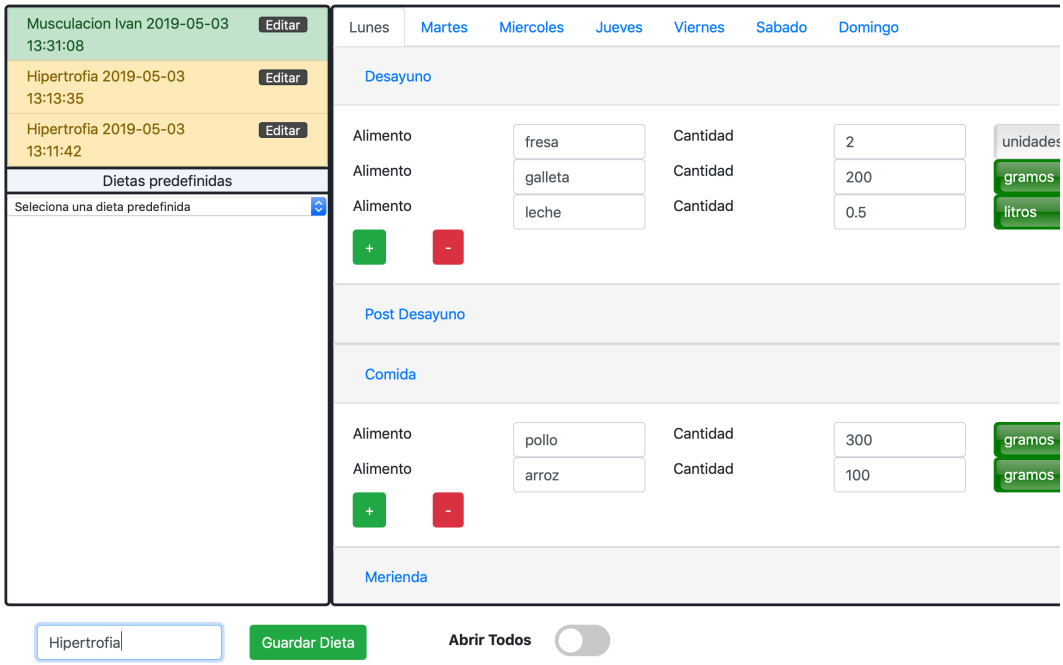


Figura 27 - a)HTML dieta, b) Typescript dieta, c)Vista dieta

7.1.14. Crear una dieta predefinida para el profesional

La aplicación proporciona la capacidad de crear una dieta predefinida para un profesional. Esta funcionalidad, es una modificación de la funcionalidad explicada en *"7.1.13. Crear dieta paciente"*. El único cambio realizado, es que se accede a través de la pantalla principal del profesional pulsando el botón naranja con el texto "*Dietas Predefinidas"* el cual trae a la vista el bloque "*paciente-dieta*" con la diferencia de que, a la hora de guardar la dieta, esta se asignará al profesional que la ha creado para ser accesible más adelante.

7.1.15. Cargar una dieta predefinida paciente

El profesional es capaz de crear dietas predefinidas. De manera que asocia una dieta a un profesional en la base de datos. Todas estas dietas, se recuperarán cada vez que se cargue la vista que permite crear y asignar cualquier dieta a un paciente. De esta forma el profesional no tiene que volver a escribir la dieta desde el principio cada vez que pretenda asignar dietas similares a distintos pacientes. Por tanto, cuando un profesional carga la vista de creación de dietas con la apariencia de la *Figura 28*, el bloque "*dietas-list"* se encargará de realizar una llamada a la clase DAO para pedir todas las dietas que están asignadas al profesional como muestra la *Figura 29.b*. Estas dietas se almacenan en un array de la clase *Dieta*, y será accesible a través de un componente HTML de tipo "*select"* como muestra la *Figura 29.a* desde la vista.
En el momento de seleccionar cualquier dieta de la lista de dietas predefinidas, se poblará la vista correspondiente a la edición de una dieta con todos los datos y alimentos que dicha dieta predefinida contiene. A través de esta acción, es posible añadir nuevos alimentos, o eliminar cualquiera de los precargados. Una vez terminada la carga y/o edición de la dieta predeterminada, la aplicación puede guardar dicha dieta y asociarla a un paciente.

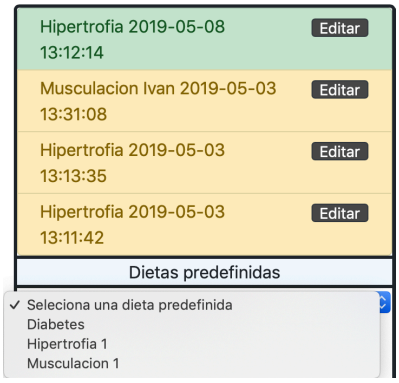


Figura 28 - Vista carga dieta

a)

```
<div class="predefinidas">
  <div class="minititulo">
    Dietas predefinidas
  </div>
  <select (change)="onChange(dieta)" [(ngModel)]="dieta" class="selec
    <option>
      Seleciona una dieta predefinida
    </option>
      <option *ngFor="let d of dietas" [ngValue]="d">
          <span *ngIf="d.nombre == ' '">{{d.fecha}}</span>
          <span *ngIf="d.nombre != ' '">{{d.nombre}}</span>
      </option>
    </select>
</div>
```

b)

```
getDietasProfesional() {
  const p = new Paciente();
  p.email = this.profesional.email;
  this.daoService.getDietas(p).subsc
    R => {
      this.dietas = R;
    }
  );
}
```

Figura 29 - a) HTML cargar dieta, b) Typescript cargar dieta

7.1.16. Editar dieta

El profesional es capaz de editar cualquiera de las dietas que haya creado tanto las asignadas a pacientes como las predefinidas creadas por él mismo. Esta acción se lleva a cabo en el bloque "*dietas-list" que* se encarga de solicitar a la clase DAO, todas las dietas asociadas a un paciente, y las predefinidas por el profesional. Mediante el botón *"editar*" que aparece al lado de cada dieta como muestra la *Figura 30.a*, es posible editar cualquiera de los alimentos de la dieta seleccionada. Al pulsar el botón editar, se ejecuta una llamada "editarDieta()" (*Figura 30.b*) perteneciente al bloque "*paciente-dieta" (Figura 30.c)* mediante un elemento "*Eventemitter".* Este elemento es el encargado de realizar una llamada a una función de otro bloque y proporcionar a dicho bloque el objeto *"Dieta*" a editar. El bloque "*paciente-dieta"* es el encargado de representar y crear una dieta. En el momento que la dieta llega al bloque "*paciente-dieta",* desencadena la ejecución de la función "*editarDieta()"* encargada de poblar con alimentos todas las franjas horarias correspondientes a la dieta que será editada. De este modo el profesional es capaz de añadir o eliminar los alimentos deseados. La carga de la dieta a editar se realiza mediante un bucle encargado de incluir todos los alimentos de la dieta, en la franja horaria y día que le corresponde.

a)

b)

c)

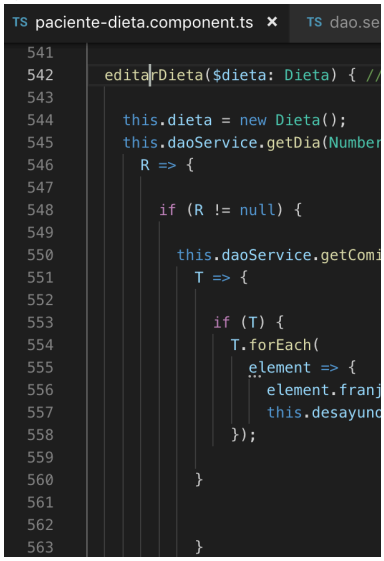


Figura 30 - a) Vista editar dieta, b) Typescript lista-dietas, c) Typescript paciente-dieta

7.1.17. Visualizar dieta

Esta funcionalidad posibilita la visualización de cualquier dieta asignada a un paciente o una dieta predeterminada asignada a un profesional, de una forma gráfica en forma de tabla como muestra la *Figura 31.c*. Pertenece al bloque *"dieta-visor"* el cual recibe la información de una dieta a representar, solicita a la clase DAO todos los datos necesarios para representar correctamente los alimentos, cantidades y unidades de los alimentos de una dieta. Para acceder a esta funcionalidad, es necesario pulsar sobre cualquier dieta que se quiera visualizar desde el bloque "*dietas-list"*. Al pulsar sobre cualquier dieta, se ejecuta la función *"verDieta()"* encargada de guardar los datos de la dieta a visualizar en el almacenamiento local, para que posteriormente el bloque de visor dieta, recoja estos datos de la sesión y pueda representar la dieta seleccionada como muestran la *Figura 31.a* y *Figura 31.b*.

a)

```
<div class= "dia">
    Lunes
</div>
<div class= "franja">
    Desayuno
</div>
<table class="table table-striped">
    <thead>
        <tr>
        <th>Alimento</th>
        <th>Cantidad</th>
        <th>Unidades</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let lunes of alimentoLunes" >
        <td *ngIf="lunes.franja === 'desayuno'">
            {{lunes.nombre}}</td>
        <td *ngIf="lunes.franja === 'desayuno'">
            {{lunes.cantidad}}</td>
        <td *ngIf="lunes.franja === 'desayuno'">{
            {lunes.unidades}}</td>
        </tr>
    </tbody>
```

b)

```
this.dietaSelecionada = JSON.parse(localStorage.getItem('dieta'));

this.daoService.getDia(this.dietaSelecionada.lunes).subscribe(
  R => {

    if (R != null) {
      this.diaLunes.desayuno = R[0].desayuno;

      this.daoService.getComida(R[0].desayuno).subscribe(
        T => {

          if (T) {
            T.forEach(
              element => {
                const alimento = new Alimento('desayuno');
                alimento.nombre = element.nombre;
                alimento.cantidad = element.cantidad;
                alimento.unidades = element.unidades;
                this.alimentoLunes.push(alimento); ////////////////
              });

          }


        }
      );
```

c)

Lunes			Martes		
Desayuno			Desayuno		
Alimento	**Cantidad**	**Unidades**	**Alimento**	**Cantidad**	**Unidades**
fresa	2.00	unidades	aguacate	2.00	unidades
galleta	200.00	unidades	leche	0.20	litros
leche	0.50	litros	Media Mañana		
Media Mañana			**Alimento**	**Cantidad**	**Unidades**
Alimento	**Cantidad**	**Unidades**	Comida		
Comida			**Alimento**	**Cantidad**	**Unidades**
Alimento	**Cantidad**	**Unidades**	lentejas	600.00	gramos
pollo	300.00	gramos	carne	1000.00	gramos
arroz	100.00	gramos	yogurt	2.00	unidades
Merienda			Merienda		
Alimento	**Cantidad**	**Unidades**	**Alimento**	**Cantidad**	**Unidades**
Cena			Cena		
Alimento	**Cantidad**	**Unidades**	**Alimento**	**Cantidad**	**Unidades**
			pavo	200.00	gramos

Figura 31 - a)HTML vista dieta, b) Typescript vista dieta, c) vista dieta

7.1.18. Acciones de paciente

En este apartado se engloban las funcionalidades: "*paciente consulta su progreso*", "*paciente consulta su dieta*", "*paciente consulta su próxima cita*" y "*contactar con el profesional*". Todas estas funcionalidades, están englobadas dentro del bloque "*principal-paciente*". Este bloque se ejecuta cuando un paciente accede a la aplicación y contiene los demás bloques necesarios para realizar todas las operaciones mencionadas. Al iniciarse la vista, la aplicación se encarga de obtener todos los datos del paciente (*Figura 32.a)*, sus medidas y sus métricas. Para ello, realiza una petición a la clase DAO, proporcionándole el "token" que se ha generado en el log-in y se encontraba almacenado en la sesión. Con este "token" la aplicación es capaz de obtener el paciente, y con los datos del paciente, obtiene las citas, las medidas, las métricas y su dieta activa a través de la clase DAO como muestra la *Figura 32.b*. Los datos obtenidos, serán entregados a los diferentes bloques a través del controlador para poder representarlos en la vista representada en la *Figura 32.c*.

En esta vista principal-paciente, existen tres botones que tienen asociados cada uno de ellos una acción:

- **Mi dieta:** Ejecuta la carga de la vista "*dieta-visor*" con los datos de la última dieta activa para el paciente.
- **Mi próxima cita:** Muestra un elemento modal en la vista con la información de la próxima cita asignada al paciente como muestra la *Figura 32.d*.
- **Contactar Nutricionista:** Realizará una llamada al sistema operativo del cliente para invocar al programa de correo electrónico por defecto instalado. Así se añadirá a la dirección del correo, el email de su profesional asignado.

En esta vista se mostrará mediante el bloque *"progreso",* las gráficas de las medidas y métricas del paciente para que pueda visualizarlas. Para llevar a cabo todas estas acciones, están presentes en el controlador, todas las clases y estructuras necesarias para manejar los datos obtenidos, y proporcionarlos a los diferentes componentes.

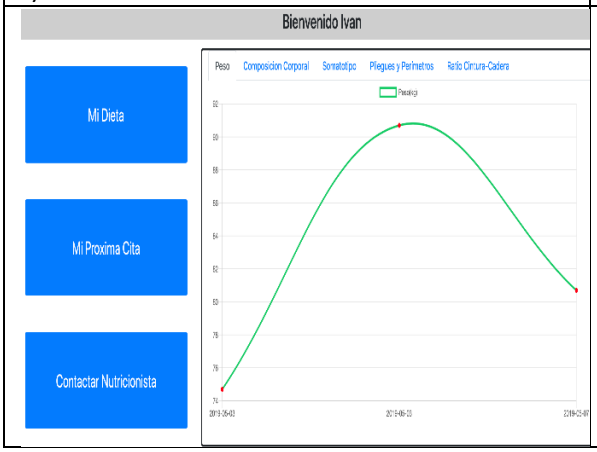
a)

```
getPaciente() {
  const token = sessionStorage.getItem('token');
  this.daoService.getPaciente(token).subscribe(
    R => {
    if (!R) {  // Fracaso
      this.autorizado = false;
    } else {  // Exito
      this.autorizado = true;
      this.paciente.nombre = R.nombre;
      this.paciente.email = R.email;
      this.paciente.emailProfesional = R.emailProf
      this.getCitaPacienteActiva();
      this.getAnatomiayProgreso();
    }
  });
}
```

b)

```
getAnatomiayProgreso() {

  this.daoService.getAnatomia(this.paciente).subscribe (
    R => {
      if (R !== null) {
        this.anatomiaList = R;
        this.anatomiaList.reverse();
        this.daoService.getMetricas(this.paciente).subscribe(
          M => {
            if (M !== null) {
              this.metricasList = M;
              this.progreso = true;
              localStorage.removeItem('anatomiaList');
              localStorage.removeItem('metricasList');
              localStorage.setItem('anatomiaList', JSON.stringify(this.anatomiaList));
              localStorage.setItem('metricasList', JSON.stringify(this.metricasList));
            }
          }
        );
      }
    }
  );
}

email() {
  window.location.href = 'mailto:' + this.paciente.emailProfesional;
}
```

c)



d)

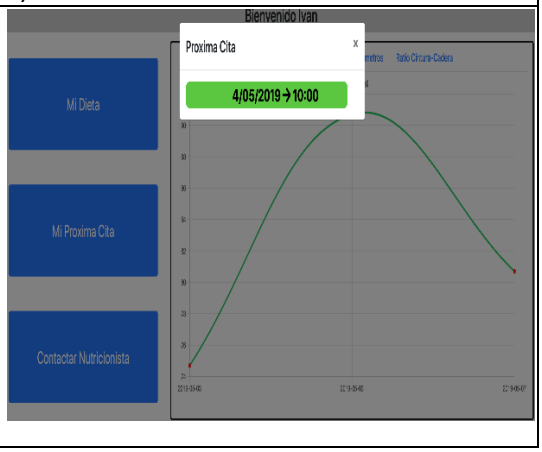


Figura 32 - a)Typescript obtener paciente, b)Typescript obtener anatomía, c) Vista paciente, d) próxima cita paciente

7.2. DAO

DAO[16] es un patrón de diseño cuya funcionalidad es la de separar la lógica de la aplicación y la lógica de acceso a los datos. Así el patrón DAO proporciona las acciones necesarias para insertar, borrar, actualizar y obtener información de la fuente de datos. Este patrón, devolverá al controlador la información solicitada correctamente encapsulada en la clase deseada, o con

un formato óptimo. En esta aplicación, el patrón DAO, viene representado en un servicio de Angular, el cual se estructura dentro de una clase, representada como una capa de abstracción entre el controlador, y los datos almacenados en el servidor. La clase DAO recibe las peticiones del controlador con la información necesaria, formatea adecuadamente esta información, la encapsula en una estructura en concreto si fuese necesaria, y se encarga después de realizar la petición al servidor. Observar que en Angular el DAO realiza una petición asíncrona devolviendo de esta manera un objeto de tipo *"Observable"* cuya función es la de notificar al controlador en el momento en el que reciba los datos del servidor. Todas las acciones de lectura, escritura o modificación de datos en el servidor por parte de la aplicación, se lleva a cabo a través de los elementos de la clase DAO.

```
getPaciente(token2: string): Observable<Paciente> {
  const cadena = {
    action: 'getPaciente',
    token: token2

 };
  // console.log(JSON.stringify(cadena));
  return this.http.post<Paciente>(Url, cadena);
}

insertaPaciente(profesional: Profesional, paciente: Paciente): Observable<boolean> {

  const cadena = {
    action: 'insertaPaciente',
    Profesional: profesional,
    Paciente: paciente

 };
  console.log(JSON.stringify(cadena));
  return this.http.post<boolean>(Url, cadena);
}


getPacientesList(profesional: Profesional): Observable<Paciente[]> {
  const cadena = {
    action: 'getPacientesList',
    Profesional: profesional

 };
  // console.log(JSON.stringify(cadena));
  return this.http.post<Paciente[]>(Url, cadena);
}
```

Figura 33 - Typescript DAO

Estas funciones del DAO reciben a través de los argumentos los datos necesarios para llevar a cabo las acciones solicitadas, como es el caso de la función "*getPacientesList()"* que tiene como argumento un objeto de tipo "*Profesional*" del cual se desea obtener su lista de pacientes, como muestra la *Figura 33*. Estos argumentos, se encapsulan en una variable denominada "*cadena"* estructurada en formato clave-valor lo que hace que sea fácilmente interpretada por la API alojada en el servidor y así proporciona a la API una correcta estructura de datos. La variable "*cadena"* es enviada a la API a través del método "*post*" del objeto "*HttpClient",* y devuelve un objeto del tipo "*Observable*" que recibirá la clase invocadora del método a la espera de una respuesta del servidor.

7.3. Back-end

Para el desarrollo de esta Aplicación se ha construido con PHP una API en el lado del servidor con el propósito de realizar llamadas internas a la base de datos dentro del mismo, enviar información a la parte front-end y poblarla de datos. Se ha utilizado también PHP para validar las credenciales introducidas por el cliente a través del front-end y de esta manera, notificar si puede acceder a la aplicación o si no.

7.3.1. API

En el back-end se ha creado una API[5] en PHP cuya misión es la de recibir peticiones a través del método "*post*" y llevar a cabo las funciones necesarias para satisfacer estas peticiones. Estas peticiones pueden ser de inserción, eliminación o modificación de la información almacenada en la base de datos y se implementan usando la librería "*mysqli*".

```php
$conn = new mysqli($servername, $username, $password, $dbname);
$data = file_get_contents("php://input");
$data = json_decode($data); // ,true
```

Figura 34 - Conexión PHP MariaDB

Tal y como muestra la *Figura 34*, la variable *"$conn"* contendrá el objeto con el que se establece la comunicación a la base de datos el cual es un objeto de la clase "*mysqli*". Todas las operaciones a realizar sobre la base de datos se llevarán a cabo a través de esta variable que contiene en dicho objeto, las credenciales necesarias, para comunicarse con la base de datos. La variable "*$data"* será utilizada para almacenar la estructura de datos que llega a la API a través del método *"post"* y será transformada a una estructura del tipo *"json".*

En el archivo *"config.php"* mostrado en la *Figura 35* se encuentran todas las credenciales y datos necesarios para acceder a la base de datos adecuada.

```php
<?php
header('Access-Control-Allow-Origin: *');
header("Access-Control-Allow-Headers: *");
header("Access-Control-Allow-Credentials: true");
header('Access-Control-Allow-Methods: GET, POST, PATCH, PUT, DELETE, OPTIONS');
header('Access-Control-Allow-Headers: Origin, Content-Type, X-Auth-Token');
//Estas cabeceras me habilitan peticiones desde otros servidores.
//Eliminarlas en produccion.
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "nutriGestion";
$pepper = "jje[)=836!43¿Ç0´Ç1";
?>
```

Figura 35 - Archivo Config.php

[5] Application Programming Interface

La creación de un profesional se lleva a cabo mediante la instanciación de un objeto de la clase Profesional como muestra la *Figura 36*.

```php
function newProfessional(){ // inserta un  nuevo profesional en db

    $correoTrim = str_replace("@","",$this->email);
    $correoTrim = str_replace(".","",$correoTrim);

    //salt and pepper
    $salt = rand(100000000,999999999);
    $pepper = "jje[)=836!43¿Ç0´Ç1";
    $passUser = $this->pass;
    $passUser .= strval($salt);
    $passUser .= strval($pepper);
    $passUser = password_hash($passUser,PASSWORD_DEFAULT);
    //end salt and pepper

    $token = sha1($this->email. $this->pass. $correoTrim);

    $sql = "INSERT INTO profesional (nombre, email, password, salt, activo, token)
            VALUES ('$this->nombre','$this->email', '$passUser','$salt' , 1, '$token');";


    return $this->conn->query($sql);

}
```

Figura 36 - Archivo profesional.php

Para crear un profesional, es necesario instanciar una clase del tipo *"Profesional.php"* la cual contiene el método *"newProfesional()"*. Este método utiliza los atributos de la clase para introducir un nuevo profesional en la base de datos. Para ello, debe crear datos adicionales que son requisitos necesarios para obtener seguridad en la validación de credenciales. Estos datos son:

- **Salt:** Es un número aleatorio el cual se almacenará en la base de datos junto con el profesional creado.
- **Pepper:** Es una cadena de caracteres invariable con la que se genera la contraseña a través de la función *"password_hash()"* perteneciente a PHP.
- **Token:** Representa una cadena de caracteres generada a través de la función de PHP *"sha1()"* y está asociado a un usuario registrado. Este token es considerado como una clave única con la que el front-end validará al usuario en el back-end con el objetivo de obtener sus datos.

La autenticación de credenciales de un usuario que quiera acceder al sistema se lleva a cabo el archivo "*auth.php*" representado en la *Figura 37*. La clase DAO realiza una petición a este archivo que se encarga de buscar si existe el email en la base de datos, y de comprobar si la contraseña que introduce el usuario coincide con la cadena de caracteres almacenada en la base de datos.

```php
$sql = "SELECT * FROM profesional where email = '$email';";
$result = $conn->query($sql);
$fetch = $result->fetch_assoc(); //Tengo en fetch la respuesta de db

if( $result->num_rows == 1){ //Es un email profesional

    $correcto = checkPassword($fetch, $passUser , $pepper);

    if ($correcto){ //loginCorrecto
        $fetch["rol"] = "profesional";      //Le paso el rol que tien
        echo json_encode($fetch);
        return;
    } else {
        return false;
    }

}
```

Figura 37 – Archivo auth.php

La comprobación de la contraseña se lleva a cabo mediante la función *"checkPassword()"*. Esta función concatena las cadenas de caracteres de la contraseña introducida por el usuario en la vista, la sal almacenada en la base de datos correspondiente a dicho usuario, y la pimienta almacenada en el archivo "*config.php"* con el objetivo de crear una cadena de caracteres de gran tamaño. Una vez creada esta cadena de caracteres se pasa por argumento a la función "*password_verify()*" de PHP que la compara con la cadena de caracteres almacenada como contraseña en la base de datos correspondiente al email del usuario.
La comprobación a través del método "*password_verify()*" es debido a que en el momento de crear dicho usuario, se almacenó en el campo *"contraseña"* de dicho usuario en la base de datos, una cadena de texto, cuyo contenido es la concatenación en el mismo orden de una cadena de caracteres generada aleatoriamente denominada "sal", la pimienta del archivo "config.php" y la contraseña introducida por el usuario. Tras la creación de esta cadena de caracteres, se produce la contraseña del usuario a través de la función de PHP "*password_hash()"* y es almacenado como contraseña del usuario el resultado producido. Si la validación es correcta, se añade a la respuesta que enviará la API, el rol correspondiente a la cuenta de usuario que ha realizado la petición de log-in.
Todas las demás acciones de inserción, eliminación o modificación de datos se realizan en el archivo "*api.php"* como muestra la *Figura 38*. Este archivo se encarga de recibir cualquier petición del front-end, decidir qué acción se quiere llevar a cabo, y ejecutar dicha acción. Para ello, todas las acciones disponibles, reciben como parámetro una variable *"$data"* que contiene la información necesaria proporcionada por el front-end para llevar a cabo la tarea.

```php
function getListaPatologias($conn, $data){

    $email = $data->Email;
    $sql = "SELECT * FROM patologia
            WHERE email = '$email' OR email = 'all' OR email = ''
            ORDER BY id asc;";
    $result =  $conn->query($sql);

    while($row = $result->fetch_assoc()){
        $json[] = $row;
    }

    if ($result->num_rows == 0) {
        return FALSE;
    }

    return json_encode($json);
}

function getListaPatologiasPaciente($conn, $data){

    $paciente = $data->Paciente;
    $sql = "SELECT b.* FROM patologia_paciente a, patologia b
            WHERE a.email = '$paciente->email'
                AND a.id = b.id
                ORDER BY id asc;";
    $result =  $conn->query($sql);

    while($row = $result->fetch_assoc()){
        $json[] = $row;
    }

    if ($result->num_rows == 0) {
        return FALSE;
    }
    return json_encode($json);
}
```

Figura 38 - api.php

En la *Figura 38*, se aprecia parte del archivo "*api.php*" en el que se representan dos funciones de la API. Estas dos funciones son:

- **getListaPatologias:** Esta función se encarga de devolver con un formato tipo *"json"* un array con todas las patologías existentes en la base de datos para un email de paciente dado a través de la variable *$email.*
- **getListaPatologiasPaciente:** Es la función encargada de obtener un array de todas las patologías que están asociadas a un paciente. Los datos del paciente vienen a través del parámetro *$data* que contiene al objeto "*Paciente"* el cual es extraído y almacenado en la variable *$paciente.* Con todos estos datos, la función realiza la llamada a la base de datos y devuelve un array con la lista de patologías que corresponden a dicho paciente.

Una vez los datos son obtenidos por la API de la base de datos, se devuelven al front-end.

8. Evaluación

Este capítulo contiene el formulario de evaluación que se ha utilizado para llevar a cabo dicha evaluación, así como los resultados obtenidos.

8.1. Formulario de evaluación

En esta sección de muestra cómo se ha llevado a cabo una evaluación de la aplicación. Para ello se ha redactado un guion con un conjunto de acciones que el usuario evaluador debe llevar a cabo en una instancia de la aplicación instalada en un servidor accesible desde internet. A continuación, el evaluador debe rellenar un formulario en línea donde deberá responder a preguntas relacionadas con la prueba realizada.

El formulario de evaluación tiene como objetivo proponer una serie de acciones a realizar por parte de un usuario. Se divide en dos partes: requisitos y acciones. Los requisitos son unas necesidades de obligatorio cumplimiento que debe tener en cuenta el usuario antes de realizar la prueba con el objetivo de que ésta sea satisfactoria.

En la *Figura 39* se muestra la parte del formulario de evaluación que contiene una introducción explicando la prueba que se va a realizar y se enumeran los requisitos a cumplir por parte del usuario. Una vez que el usuario ha cumplido los requisitos, pasará a realizar las acciones solicitadas para la prueba.

El presente documento, muestra una guía para la evaluación de la herramienta NutriGestión. Va dirigida a los profesionales de la nutrición que trabajan con múltiples pacientes y será accesible tanto a profesionales como pacientes de estos profesionales.

Esta evaluación será completamente anónima y medirá la usabilidad de la herramienta y el grado de satisfacción del usuario al realizar las diferentes acciones.

A continuación, se propondrán diferentes actividades a realizar dentro de la aplicación siguiendo los pasos de esta guía en orden secuencial tal y como están descritos las diferentes acciones.

Si presenta alguna dificultad, y no sabe cómo realizar cualquiera de las acciones, pase a la siguiente acción de la lista y refléjelo, en el cuestionario más adelante.

En caso de presentar algún fallo técnico, si es posible, pase a la siguiente acción de la lista. Si los fallos técnicos impiden la realización de los pasos posteriores, salte al siguiente paso que sea posible y por favor, refleje estas dificultades en el cuestionario más adelante.

Para la correcta evaluación de la aplicación, es necesario cumplir los requisitos enumerados más abajo en este documento.

Requisitos

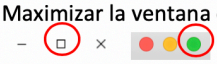
1. Ordenador de escritorio o portátil
2. Sistema operativo Windows, Mac o Linux
3. Navegador web (Firefox, Chrome, Opera o Safari)

Figura 39 - Formulario evaluación I

Como observa en la *Figura 40*, el usuario debe maximizar la ventana del navegador y acceder a la dirección IP que aparece en el paso 3. Esta dirección IP, es en la que se encuentra el servidor privado con la aplicación instalada. Tras acceder a la aplicación los siguientes pasos que debe realizar son los referentes a la creación de un usuario con rol profesional y la creación de un paciente al que se le asocia una cita, una patología y diferentes medidas.

Acciones

1. Abrir alguno de los navegadores web de los requisitos.
2. Maximizar la ventana del navegador, o en su defecto que se vea lo más grande posible



3. Copia, Pegar y acceder a la siguiente dirección en el navegador web abierto: http://83.50.125.87:8080
4. Una vez dentro de la aplicación, lo primero que debemos hacer es crearnos una cuenta de profesional para empezar a trabajar, estas cuentas son limitadas, y para ello es necesario un código de registro, tu código es el ***1231***
5. Regístrate como profesional con un correo electrónico válido y acuérdate de tu contraseña
6. Accede a la aplicación con tu correo electrónico y tu contraseña
7. Añade un nuevo paciente:
    a. nombre: Ivan
    b. apellido 1: Canas
    c. apellido 2: Ramos
    d. edad: 28
    e. Sexo: Hombre
    f. email: icanas1231@ucm.es
    g. contraseña: ivan
8. Agrega una nueva cita a nuestro paciente Ivan, esta cita será para el 30 de junio de 2019 a las 17:15 H.
9. Ivan es diabético y tiene hipertensión, te lo ha dicho en la consulta, añádele estas patologías en su historial.
10. Como es tu primera consulta, le realizas a Iván sus medidas antropométricas en tu consulta. Introduce sus medidas y guárdalas. Sus medidas son las siguientes:
    a. Peso: 74.7
    b. Altura: 179.8
    c. Pliegue tríceps: 10.8

Figura 40 - Formulario evaluación II

Como se aprecia en la *Figura 41*, se pide al usuario que realice las acciones de añadir unos alimentos a una dieta en diferentes franjas horarias y días, así como guardar y editar la dieta y crear una dieta predeterminada.

11. Viendo los datos que muestran sus métricas y tras comunicarte el paciente que quiere ganar musculatura, decides ponerle a dieta acorde a sus deseos. Para ello, debes crear una dieta. Su dieta consistirá en lo siguiente:
    a. Desayuno del lunes:
        i. 2 manzanas
        ii. 200 gramos de garbanzos

    b. Comida del martes:
        i. 300 gramos de arroz
    c. Cena del martes:
        i. 1 manzana,
        ii. 2 plátanos
        iii. 100 gramos de arroz
    d. Comida del sábado:
        i. 3 patatas
        ii. 500 gramos de pollo
        iii. 200 gramos de fresas.
    e. La dieta se llamará hipertrofia
12. Guarda la dieta creada
13. Te has equivocado, y se te ha olvidado que el paciente tiene que beber dos litros de leche, edita la dieta creada y añade los dos litros de leche a la dieta, el Post-Desayuno del lunes y quita los 200 gramos de fresas de la comida del sábado.
14. Guarda la dieta con el mismo nombre de Hipertrofia
15. A pesar de haber hecho cambios, y que, en tu dieta actual, ya no aparecen los 200 gramos de fresa, quieres visualizar por pantalla la dieta anterior llamada Hipertrofia.
16. Ve a la página principal donde aparece tu lista de pacientes
17. Debido a que crear una dieta desde cero para cada paciente nuevo, es muy tedioso, en un rato libre, decides crear una dieta predeterminada.
18. Esta dieta contiene lo siguiente:
    a. Desayuno de lunes: 3 galletas

Figura 41 - Formulario evaluación III

En la *Figura 42* se refleja la petición al usuario para llevar a cabo la creación de nuevos pacientes, y para ello, se le proporcionan los datos que deben tener estos usuarios.

19. Tu consulta es un éxito y han aparecido 3 pacientes nuevos, añádelos a la aplicación, sus datos son los siguientes:
    a. Natalia
        i. Nombre: Natalia
        ii. Apellido 1: Hristova
        iii. Apellido 2: Migdalova
        iv. Edad: 25
        v. Sexo: Mujer
        vi. Email: natalia1231@ucm.es
        vii. Contraseña: natalia
    b. Ana
        i. Nombre: Ana
        ii. Apellido 1: Ramos
        iii. Apellido 2: Casado
        iv. Edad: 42
        v. Sexo: Hombre
        vi. Email: anaRamos1231@ucm.es
        vii. Contraseña: anaramos
    c. Ana
        i. Nombre: Ana

Figura 42 - Formulario evaluación IV

La *Figura 43* muestra una consecución de acciones a realizar por parte del usuario cuyo principal objetivo es el de añadir y quitar citas a diferentes pacientes, actualizar sus medidas, editar sus dietas y observar su progreso mediante gráficas.

20. Hoy es un nuevo día y aparece una paciente en tu consulta, te dice que se llama Ana, actualmente tienes dos pacientes con el nombre Ana en tu consulta. Ana te dice que se apellida Nadal, para ello, vas a buscar Nadal en el cuadro de búsqueda que hay en la lista de pacientes y accederás a su ficha.
21. Ana te dice que quiere una nueva cita para el 3 de Agosto a las 9:30 de la mañana, añádele la cita deseada.
22. Hoy es 30 de Mayo de 2019 y son las 17:15 tenías una cita con Iván. Iván te comunica que ha ido al médico y le han comunicado que es celiaco, añade celiaco a la lista de patologías y márcasela a Iván.
23. Tras tomarle las medidas a Iván, ves que ahora pesa 90 kilos, todos sus pliegues han aumentado 10 milímetros, sus perímetros han aumentado 20 centímetros y sus diámetros 30 centímetros. Márcalo en sus medidas y guárdalas.
24. Ahora quieres consultar sus medidas y métricas anteriores.
25. Ahora quieres volver a ver las actuales.
26. Iván te pregunta por su progreso, muéstrale las diferentes gráficas de su progreso, y la progresión que llevan sus medidas.
27. Quieres cambiarle la dieta a Iván ya que esta no le ha ido muy bien. Para ello vas a aprovechar la dieta predeterminada a la que llamaste “Musculación 1” y le añadirás a la cena del lunes 2 litros de leche. A esta dieta la llamarás “Musculación Ivan”
28. Ha terminado la consulta de Iván deberás darle por atendido para que desaparezca su cita. Iván te ha dicho que ya te llamará para concertar la siguiente.
29. Ahora te ha llamado Ana Nadal por teléfono, y te ha dicho que el 3 de Agosto a las 9:30 de la mañana no va a poder venir, que mejor a las 10:00. Cámbiale la fecha a Ana Nadal.
30. Hoy, Ana Ramos te ha llamado por teléfono y te ha dicho que no va a continuar con la consulta, que quiere que la desactives por que no va a volver. Desactiva a Ana Ramos.
31. Te ha llamado por teléfono Iván, te pide una cita para mañana a las 14:00, Añádesela.

Figura 43 - Formulario evaluación V

En la parte final de la evaluación, como muestra la *Figura 44* se solicita al usuario que realice log-in como uno de los pacientes creados y desempeñe las funciones asociadas al paciente. Estas acciones son las de visualizar sus gráficas de progreso, consultar su dieta, su próxima cita y escribir un correo electrónico a su nutricionista. Finalmente, se le proporciona un enlace para responder al cuestionario creado con el fin de evaluar la herramienta.

32. Ahora tomarás el papel del paciente, en este caso eres Iván.
33. Como Iván, quieres utilizar la aplicación, para ello ve a la pantalla de Log-in dándole a tu navegador al botón "atrás" o escribe la dirección del punto 3 en el navegador y realiza el Log-in con los datos de Ivan. Email: icanas1231@ucm.es Contraseña: ivan
34. Visualiza tu progreso de peso, composición corporal, somatotipo, pliegues y perímetros y el ratio cintura-cadera.
35. Consulta tu próxima cita.
36. Escribe un correo a tu nutricionista.

Cuestionario

Una vez finalizado, por favor, responde a las preguntas del siguiente cuestionario:
Pinchando aquí: CUESTIONARIO
O navegando a la siguiente dirección: **https://forms.gle/3pLy8JfaqY98J7Cq7**

Recuerda que el cuestionario es completamente anónimo.

Gracias.

Figura 44 - Formulario evaluación V

8.2. Resultados

La aplicación ha sido evaluada por once personas las cuales han realizado un cuestionario de evaluación. Una vez finalizada la evaluación, los datos obtenidos del cuestionario han sido los siguientes:

- La mayoría de los encuestados (54,5%) tienen una edad de 25 años como muestra la *Figura 45*.

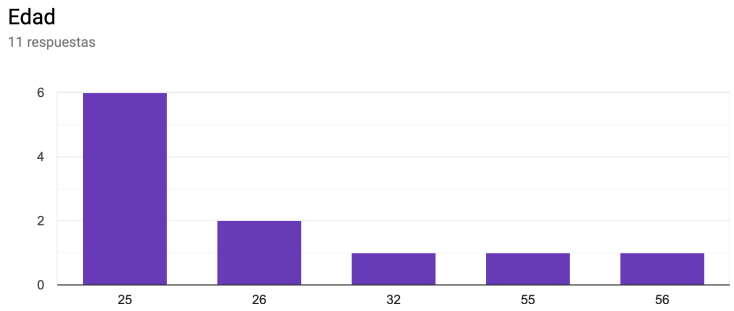


Figura 45 - Resultados edad

- La *Figura 46* muestra que la mayoría de los evaluadores eran mujeres con un 63,6%.

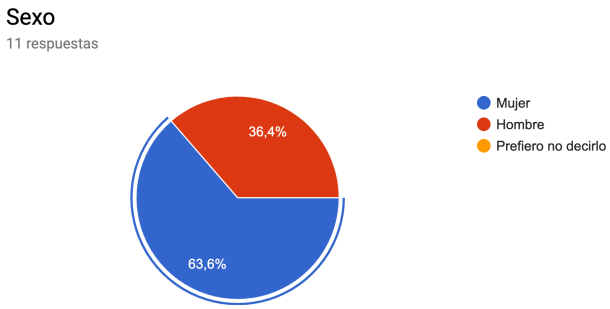


Figura 46 - Resultados gráfica sexo

- El nivel de estudios de la mayoría de los encuestados pertenece al campo de ciencias de la salud (63,6%) representado en la *Figura 47*.

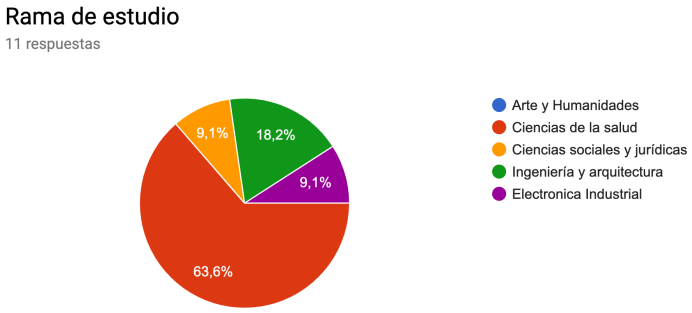


Figura 47 - Resultados rama de estudio

- El 100% de los encuestados, dice no haber tenido ninguna dificultad a la hora de acceder a la aplicación (*Figura 48)*.

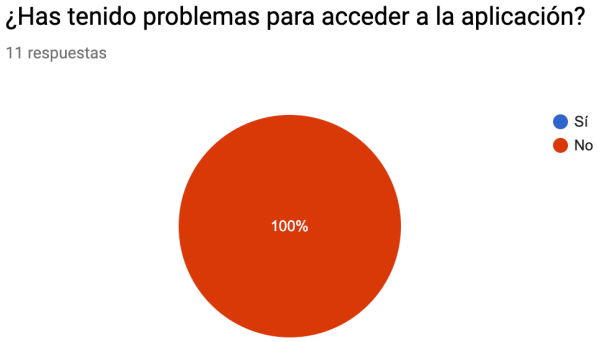


Figura 48 - Resultados usuarios problemas

- En una escala del 1 al 10 siendo 1 nada intuitiva y 10 muy intuitiva, los encuestados, en su mayoría (54,5%) considera que la aplicación presenta un valor 9 a la hora de valorar esta característica. También se aprecia que al menos 1 persona, no considera la aplicación intuitiva (*Figura 49)*.

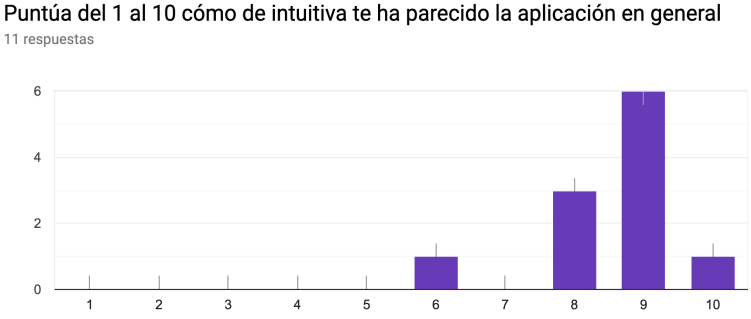


Figura 49 - Resultados intuitiva

- En una escala del 1 al 5 siendo 1 nada intuitiva y 5 muy intuitiva, se ha preguntado a los encuestados la capacidad de intuir el acceso a la ficha de un paciente. Cinco de los once encuestados valora la aplicación con un 5, tres de ellos lo han calificado con un 4 y uno con una puntuación de 3(*Figura 50*)

Figura 50 - Resultados ficha paciente

- En una escala del 1 al 5 siendo 1 nada sencillo y 5 muy sencillo, se ha preguntado a los encuestados la facilidad de añadir medidas a un paciente. Cinco de los once encuestados valora la aplicación con un 5, dos de ellos lo han calificado con un 4 y otros dos con una puntuación de 3(*Figura 51*)

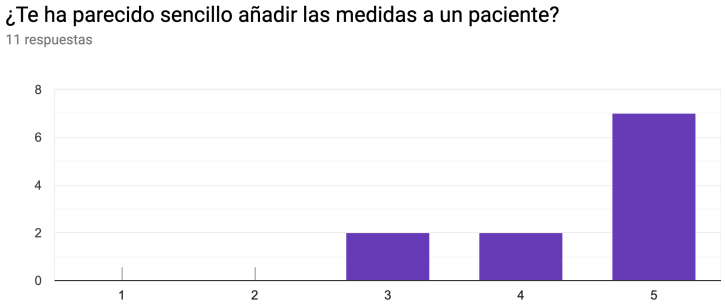


Figura 51 - Resultados medidas

- Se les ha preguntado si añadirían alguna métrica o medida más a la aplicación, siendo mayoría la respuesta "NO". Por otra parte, existe un encuestado que ha sugerido a esta pregunta la inclusión de *"datos de una bioimpedancia"* como muestra la *Figura 52.*

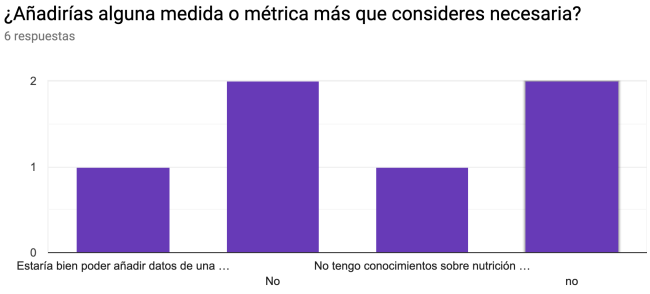


Figura 52 – Resultados necesidad de más medidas o métricas

- En una escala del 1 al 5 siendo 1 nada intuitivo y 5 muy intuitivo un usuario manifiesta que no le ha resultado intuitivo la forma de implementar la creación de dietas predeterminadas para utilizarlas posteriormente marcando la respuesta con un 2. Siendo mayoría (72,7%) los que han marcado esta respuesta con un 5 (*Figura 53*).

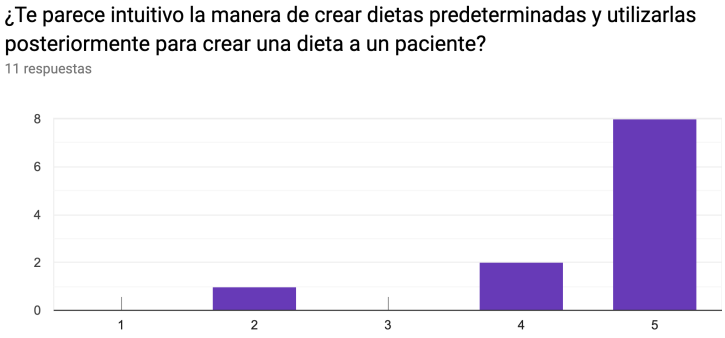


Figura 53 - Resultados facilidad dietas predeterminadas

- Se ha preguntado a los encuestados si han podido realizar todas las acciones propuestas en el guion para el paciente Iván. El 81,8% de ellos, dicen haber podido realizar todos los puntos del guión. Dos encuestados no han podido realizar todos debido a la no existencia de un cliente de correo predeterminado instalado. Y por otra parte no ha encontrado el campo "Perímetro Biepicondilar" (*Figura 54)*.

Figura 54 - Resultados realización acciones

- Se les ha preguntado, que indiquen el número que corresponde al guion con el que han tenido alguna dificultad (*Figura 55)*. Estos puntos problemáticos son:
    - Dos encuestados han presentado problemas con el punto 1 siendo este punto "Abrir alguno de los navegadores web de los requisitos."
    - Un encuestado ha presentado problemas con el punto 4, encargado de introducir el código de registro.
    - Un encuestado ha presentado problemas con el punto 10 cuyo objetivo era el de introducir las medidas a un paciente.
    - Un encuestado ha presentado problemas con el punto 13 encargado de modificar una dieta añadiendo un alimento más.
    - Un encuestado ha presentado problemas con el punto 14 cuya función es guardar la dieta del punto 13 junto con un nombre dado.
    - Un encuestado ha presentado problemas con el punto 16 el cual pide al usuario ir a la página principal del profesional.
    - Dos personas han presentado problemas con el punto 26 cuyo objetivo era el de mostrar a un paciente sus gráficas de progreso.
    - Un encuestado ha presentado problemas con el punto 27 encargado de añadir una dieta predeterminada a un paciente añadiendo un alimento más.
    - Un encuestado ha presentado problemas con el punto 28 que consiste en marcar como atendido al paciente "Iván".
    - Un encuestado ha presentado problemas con el punto 29 encargado de cambiar una cita a un paciente.

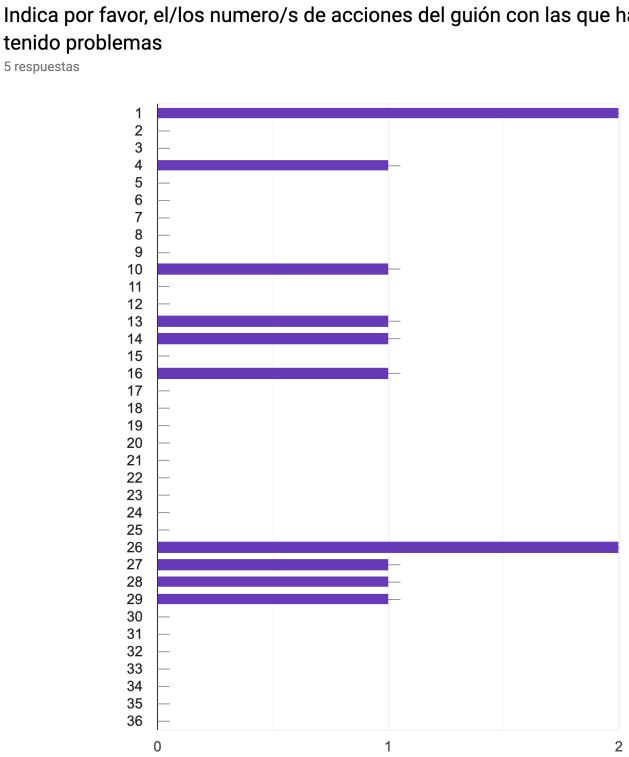


Figura 55 - Resultados puntos problemáticos

- Se ha pedido a los evaluadores que indiquen anomalías y problemas encontrados en la aplicación. Las respuestas más generalizadas son la no funcionalidad del "Pad numérico del teclado" y la confusión a la hora de volver a la página principal. Muestran también la dificultad de encontrar el campo "*Diámetro biepicondilar del fémur"* en el apartado de medidas de un paciente, y la incomodidad de utilizar el carácter *punto* en lugar de la *coma* (*Figura 56).*

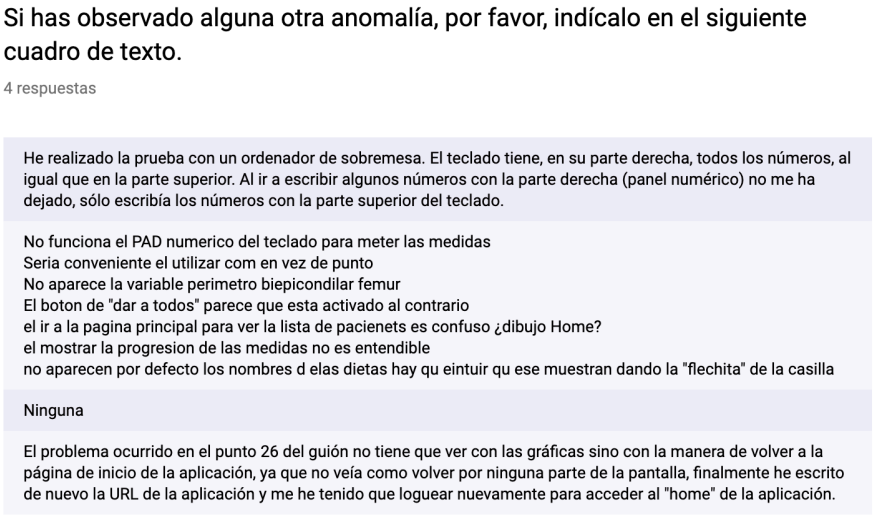


Figura 56 - Resultados anomalías en la aplicación

- Finalmente, se les ha preguntado si añadirían alguna funcionalidad que les resultase útil a la hora de utilizar la aplicación (*Figura 57)*. Estas son:
    - Poder utilizar el teclado numérico del teclado
    - Poder guardar más datos clínicos del paciente como su historial clínico y dietético, un recuerdo de 24 horas y un cuestionario de frecuencia de consumo de alimentos.

- Recomendación de ejercicio físico.
- Recetas de cocina.
- Poder imprimir la dieta de un paciente.
- Los datos antropométricos son útiles para ciertas patologías.
- Datos y acciones para obtener el cálculo (TMR) de Harris y Benedicto o (GET) necesarios para un paciente.

Por último, ¿Añadirías alguna funcionalidad que te resultase útil a la hora de utilizar la aplicación?

6 respuestas

El poder utilizar el pad numerico del teclado
utilizar las comas ne vez d elso puntos

Poder guardar más datos del paciente, historial clínico y dietético (recuerdo 24 horas, cuestionario de frecuencia de consumo de alimentos), por ejemplo

Pienso que sería util algún apartado en el que se le pueda recomendar algún tipo de ejercicio físico al paciente, para que lo pueda compaginar con la alimentación de la dieta, pero comprendo que bastante denso es el tema de la nutrición como para juntarlo con el deporte.

Añadir algunos platos con la receta incluida

Añadiría varias sugerencias:
1. - es necesario que el paciente conozca su dieta o bien espero que tenga acceso a sus dietas o bien espero que apliques un botón icono para imprimir o enviar al paciente.

2.- los datos antropométricos solo los encuentro útiles para patologías ciertas patologías

3.- No hay datos ni acciones para obtener calculo de (TMR) de Harris y benedict o (GET) necesaria para el paciente. en mi opinión deberías integrar un calculo o un algoritmo de cálculos para obtener estos datos.

buen trabajo animo!!!

Figura 57 - Resultados sugerencias

9.A. Conclusiones y trabajo futuro

Se ha creado una aplicación web destinada a los profesionales de la nutrición que reúne las funcionalidades y herramientas necesarias para facilitar su trabajo en una consulta de nutrición. Ofrece al profesional la capacidad de almacenar las medidas antropométricas de los pacientes, calcular unas métricas a partir de estas medidas, almacenar estos datos a modo de histórico para ser consultados en cualquier momento, asociar patologías a los pacientes para así crear diagnósticos y tratamientos con más información disponible, la visualización de los datos a través de diferentes gráficas o un control de las próximas citas de los pacientes permitiendo de esta manera facilitar la organización por turnos de una consulta. Otra característica importante es la facilidad para crear dietas con alimentos, su cantidad y unidades en las diferentes franjas horarias del día. Con respecto al paciente, éste puede visualizar la dieta creada por el profesional, consultar su próxima cita, u observar la evolución de sus medidas y métricas a través de las diferentes gráficas. Por último el perfil del administrador se encarga de gestionar a los diferentes usuarios de la aplicación.

Como líneas de trabajo futuro, se encuentran:

- Inclusión de mayor cantidad de medidas y métricas relevantes para el nutricionista.
- Mayor cantidad de gráficas que complementen estas nuevas medidas y métricas con el objetivo de aportar más información sobre el estado de salud al nutricionista y al paciente.
- Inclusión de un apartado de recetas de cocina utilizando los alimentos incluidos en la dieta del paciente, de manera que al incluir en una franja horaria ciertos alimentos, la aplicación pueda sugerir recetas de cocina que contengan dichos alimentos.
- Ofrecer la posibilidad a un paciente de marcar en su dieta los alimentos que ha ido consumiendo. De esta manera el nutricionista podría realizar un mejor seguimiento de los hábitos alimenticios del paciente.
- Añadir a los datos del paciente, un cuestionario de frecuencia de consumo de alimentos, favoreciendo así el conocimiento de los hábitos alimenticios del paciente antes de realizar la primera consulta.
- Funcionalidad que abarque el ámbito deportivo. De esta manera se propondrán rutinas de ejercicios que complementen el ejercicio físico con el tratamiento de nutrición.
- Contabilizar las calorías y valores nutricionales de los alimentos de una dieta, con el objetivo de proponer un objetivo de ingesta de calorías y nutrientes diarios al paciente.

9.B. Conclusions and future work

A Web application for nutritionists has been created that brings all the tools necessary to make easier the nutritionist work in a nutrition office. It offers the professional the ability to store the patients anthropometric measures, calculate some metrics from these data, store this data in a historical way to be consulted at any time, associate different pathologies to the patients so professionals can make diagnoses and provide better treatments with more available information, the visualization of the data through different graphs, and control the patients next appointments making easier the organization by turns in a nutrition office. Another important feature is the ability to create diets with different foods, their quantity and units in the different time bands of the day. In the patient's side, they can visualize the diet created by the professional, consult its next appointment, or observe the evolution of its measurements and metrics through the different graphs. Finally, the administrator is responsible for managing the different users of the application.

The future work lines are:

- Inclusion of a greater number of measurements and metrics relevant to the nutritionist
- More graphs to complement these new measurements and metrics in order to provide more information on the health status of the patient.
- Inclusion of recipes section using food included in the patient's diet, so by including certain foods in any time slot, the application may suggest cooking recipes containing such foods.
- Offer a patient the possibility to mark in his diet the food he has been consuming. In this way the nutritionist could make a better follow-up of the patient's eating habits.
- Add to the patient's data, a questionnaire of food frequency consumption to obtain a better knowledge of the patient eating habits before making the first meeting.
- A feature that includes sport exercises. In this way it will be propose some exercises routines to combine the physical exercise with the diet treatment.
- A feature that counts the calories and nutritional values of foods that are present in the diet, with the aim of proposing a goal of calories intake, as well as a daily nutrients goal to the patient.

Bibliografía

[1] *Angular.* Recuperado de https://angular.io

[2] *Anirudhabhowmik.* "anirudha Bhowmik registration form hover effect". Recuperado de https://bootsnipp.com/snippets/or3WG

[3] *Apache Friends. "Xampp".* Recuperado de https://www.apachefriends.org/es/index.html

[4] *Apache Software Foundation. "Apache HTTP Server Project".* Recuperado de https://httpd.apache.org

[5] *Bootstrap Team*. Recuperado de https://getbootstrap.com

[6] *Charts.js.* Recuperado de https://www.chartjs.org

[7] *Font icons. "FontAwesome".* Recuperado de https://fontawesome.com

[8] *GitHub Inc.* Recuperado de https://github.com

[9] Heath, B.H., & Carter, J.E.L. (1967). A modified somatotype method. American Journal of Physical Anthropology, 27, 57-74.

[10] *mariaDB Foundation. "MariaDB".* Recuperado de https://mariadb.org

[11] *Microsoft.* Recuperado de https://www.typescriptlang.org

[12] *Node.js Foundation.* Recuperado de https://nodejs.org/en/

[13] *Nutrium.io.* Recuperado de https://blog.nutrium.io/es/masa-muscular-y-osea-ecuaciones-predictivas/

[14] *PHP Group. "PHP".* Recuperado de https://php.net

[15] *Postman Inc.* Recuperado de https://www.getpostman.com

[16] *Shubham. "DAO Design Pattern".* Recuperado de https://www.journaldev.com/16813/dao-design-pattern

[17] *Valor Software. "ngx-bootstrap*". Recuperado de https://valor-software.com/ngx-bootstrap/#/

[18] *W3schools. "How TO - Toggle Switch".* Recuperado de https://www.w3schools.com/howto/howto_css_switch.asp

[19] Yuhasz, M. "The Effects of Sports Training on Body Fat in Man With Predictions for Optimal Body Weight," Doctoral Dissertation, University of Illinois, Urbana, 1962.

Anexos

Anexo I: Guía de uso

Para poder ejecutar correctamente la aplicación web, es necesario cumplir los siguientes requisitos:

- Navegador web actualizado capaz de renderizar HTML y CSS (Firefox, Chrome, Opera, Safari).
- JavaScript activado.
- Conexión a internet.
- Almacenamiento en sesión activado en el navegador web.
- Recomendado pantalla igual o superior a 13 pulgadas.
- Resolución de pantalla igual o superior a 1366 x 768 píxeles.

A continuación, se van a mostrar las diferentes formas de utilizar la aplicación.

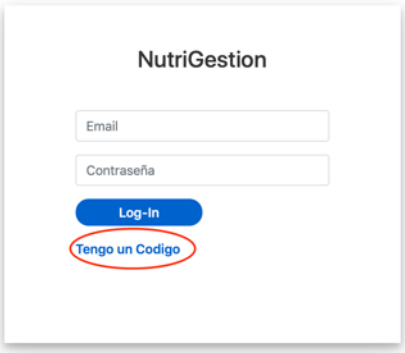
1. Registro de profesional

El primer paso para utilizar la aplicación por un profesional es crear una cuenta con el rol de profesional. Para ello, debe contactar con el administrador para que le proporcione un código de acceso único y poder registrarse.
Una vez obtenido el código, el usuario debe pulsar sobre el enlace *"Tengo un código"* como muestra la *Figura 58.a* y será redirigido a la vista de validación de código representada en la *Figura 58.b*. Tras introducir el código proporcionado por el administrador, la aplicación lo validará y cargará la vista que permite crear una nueva cuenta de profesional que muestra la *Figura 58.c*.
Tras crearse esta cuenta de profesional, la aplicación redirigirá al usuario a la página principal de log-in.

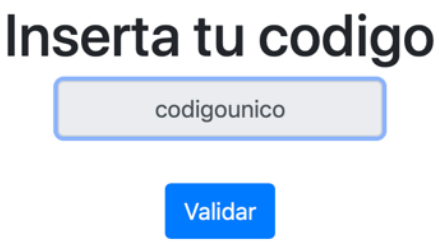
2. Log-in profesional

Una vez realizado el punto anterior, disponemos de una cuenta profesional con unas credenciales. En la pantalla de log-in, introduce las credenciales recientemente creadas y pulsa sobre el botón *"log-in"*. De esta manera se concederá el acceso a la aplicación si las credenciales son correctas y cargará la vista principal del profesional como muestra la *Figura 58.d*.

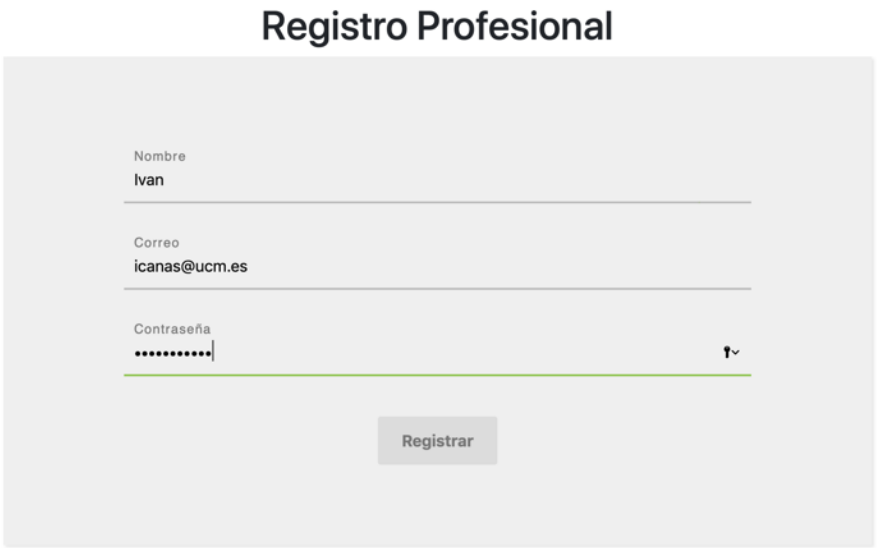
a)



b)



c)



d)

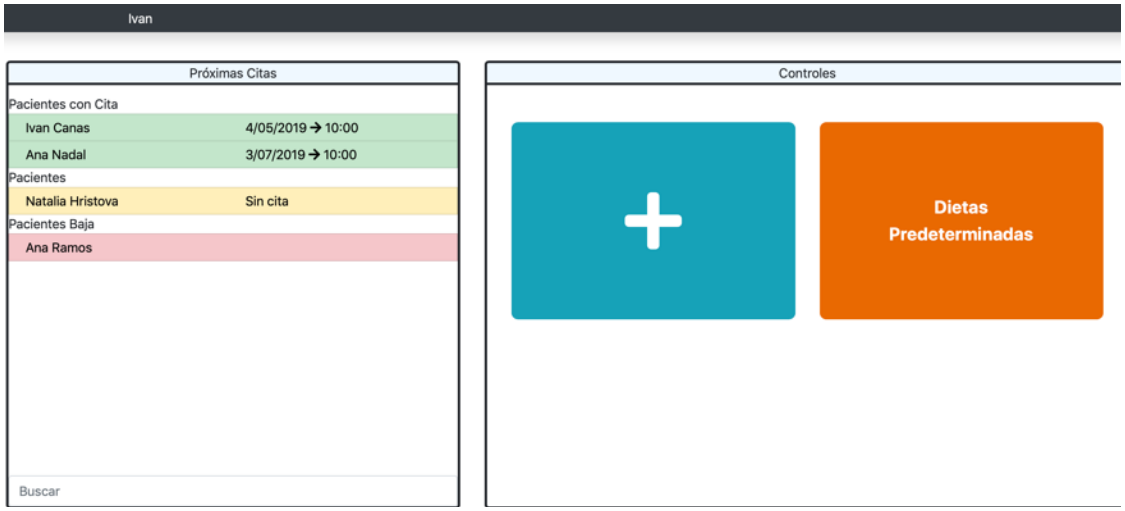
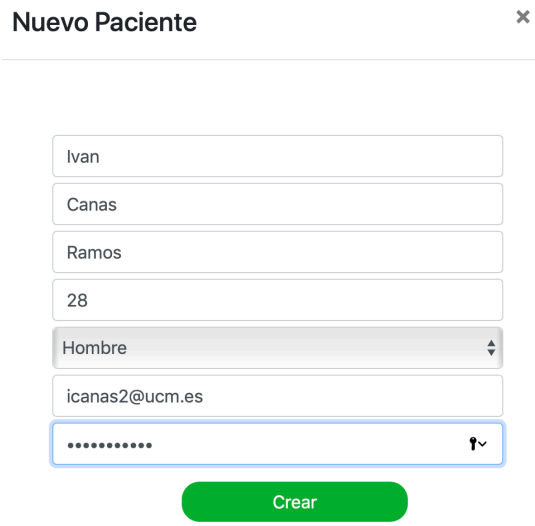


Figura 58 - a)Acceso a introducir código, b)Introducir código, c)registro profesional, d) Pantalla principal

3. Alta nuevo paciente

Para dar de alta un nuevo paciente, se debe pulsar el botón “+” del apartado “controles” de la vista principal. De esta forma aparecerá una pequeña ventana que contiene el formulario necesario para incluir los datos que identifican al nuevo paciente. Una vez rellenos los datos que aparecen en la *Figura 59.a*, se debe pulsar sobre el botón “crear”. De esta manera, el nuevo paciente creado aparecerá en la lista de pacientes, y la aplicación cargará la vista que corresponde a su ficha como muestra la *Figura 59.b*.

a)



b)

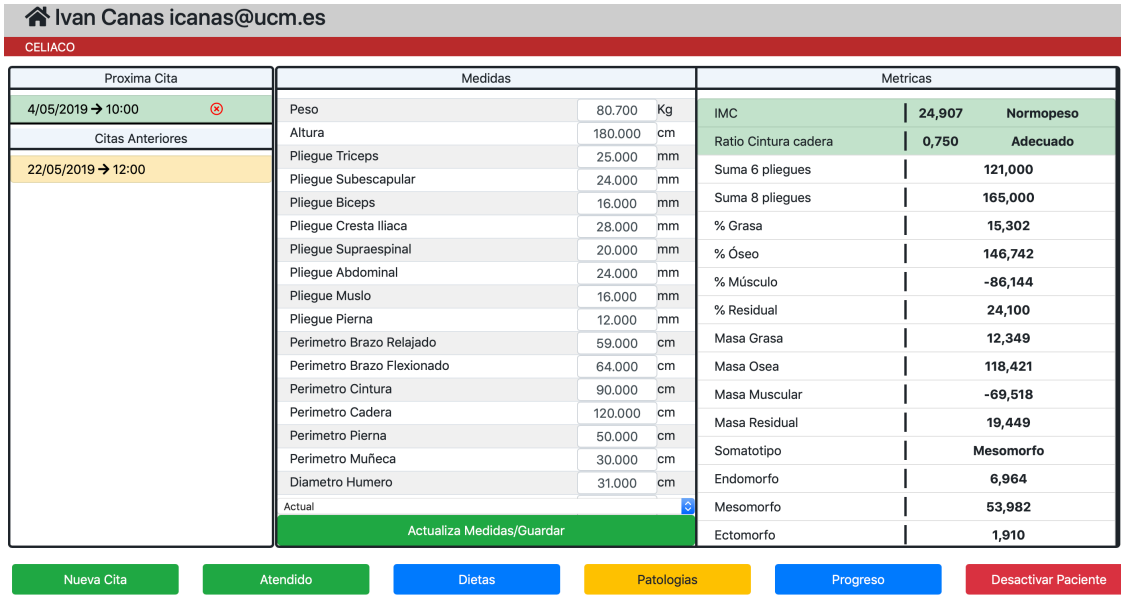


Figura 59 - a)Modal nuevo paciente, b) Ficha paciente

4. Añadir cita a un paciente

Se pueden añadir citas a cualquier paciente. Estas citas se almacenan en un histórico en la sección de citas. Únicamente puede existir una cita activa, representada con un color verde en la misma sección. Si se desea cancelar una cita, basta con pulsar el icono ⊗ que se encuentra al lado de la cita activa. Para añadir una nueva cita, basta con pulsar el botón Nueva Cita que

desplegará un formulario con los datos necesarios para añadir una nueva cita como muestra la *Figura 60*.
Si se añade una nueva cita, estando activa en ese momento una cita, la cita antigua quedará desactivada pasando a estar activa esta nueva cita.

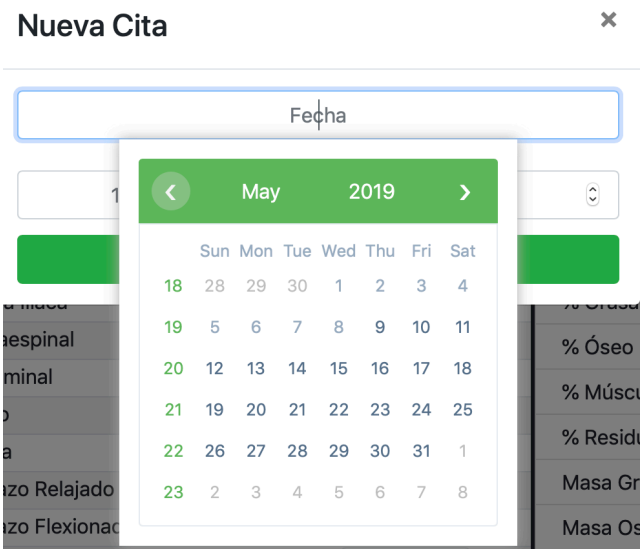


Figura 60 - Formulario creación de cita

Una vez se ha atendido a un paciente con cita, bastará pulsar el botón Atendido para cancelar la cita actual.

5. Añadir/Modificar medidas y métricas

Existe la posibilidad de asignar unas medidas a un paciente. Esta asignación de medidas produce un cálculo de sus métricas en tiempo real. Para llevar a cabo esta acción basta con acceder a la ficha del paciente en cuestión y rellenar sus datos correspondientes a sus medidas. Una vez añadidas todas las medidas deseadas, se debe pulsar sobre el botón "Actualizar medidas/Guardar" para guardar las nuevas medidas y métricas del paciente. Es posible obtener medidas y métricas del histórico del paciente seleccionando de la lista situada encima del botón "Actualizar medidas/Guardar". Al seleccionar cualquiera de sus medidas del histórico, se cargarán las medidas y las métricas seleccionadas. Cuando se cargan unas medidas del histórico y se modifican, se guardarán en el histórico del paciente como unas nuevas medidas. El histórico se mantiene intacto.

6. Dietas paciente

El profesional dispone de la capacidad de asignar múltiples dietas a un paciente pulsando sobre el botón Dietas. Se cargará una vista (*Figura 61*) en la que se permite añadir tantos alimentos como se desee en las diferentes franjas diarias disponibles, en cualquiera de los diferentes días de la semana. Para ello, se pulsará sobre el botón "+" que nos añade un nuevo alimento a la lista, y tan solo se debe rellenar el nombre del alimento, su cantidad y sus unidades. Si el alimento ya está presente en la base de datos, se cargarán automáticamente sus unidades. Una vez rellena toda la dieta a gusto del profesional, se le puede asignar un nombre. Tras pulsar sobre el botón guardar, la dieta formará parte del paciente, pudiendo ver en su histórico en la parte izquierda de la vista, las diferentes dietas asignadas a este paciente a lo largo del tiempo. Es posible editar una de estas dietas. Para ello, se debe pulsar sobre el botón Editar que se encuentra al lado de cada dieta, y automáticamente se cargará en la zona de edición de ditas la dieta seleccionada para añadir, eliminar o editar cualquier alimento. Es posible también obtener cualquier dieta predefinida creada por el paciente, seleccionándola de la lista de dietas predefinas que se encuentra debajo del histórico de dietas. Al seleccionar cualquiera de las dietas predefinidas, se cargará en la vista de edición de dietas para añadir, eliminar o modificar alimentos. Una vez completada la dieta, al pulsar sobre el botón Guardar Dieta, se asignará la dieta al paciente y formará parte de su histórico convirtiéndose en la dieta activa.

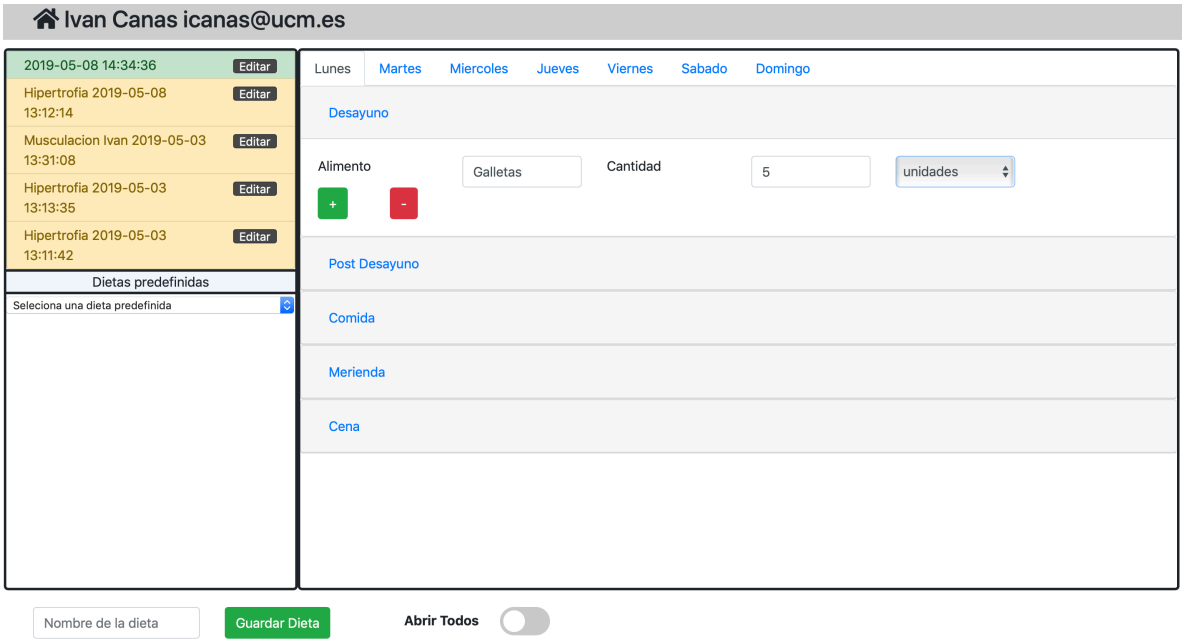


Figura 61 - Vista de dietas

El profesional es capaz de ver de forma gráfica y correctamente formateada, cualquier dieta con solo pulsar en la dieta deseada a visualizar en la lista de dietas.

7. Patologías

El profesional tiene la capacidad de asignar una o varias patologías a un paciente. Para llevar a cabo esta acción, tan solo debe acceder a la ficha de un paciente y pulsar sobre el botón Patologias. Esta acción proporciona una pequeña ventana con todas las patologías disponibles que el profesional puede asignar al paciente como muestra la *Figura 62*. Para asignar la patología deseada, basta con pulsar sobre el elemento deslizable y asignar una o varias patologías a un paciente. Al asignar cualquier patología al paciente, esta aparecerá en una barra roja en la zona superior de la ficha del paciente. El profesional es capaz de añadir cualquier patología que no se encuentre disponible en la lista de patologías inicial. Para ello, basta con introducir el nombre de la nueva patología y pulsar el botón +. De esta manera, la nueva patología esta lista para ser asignada a cada paciente que el profesional considere necesario.

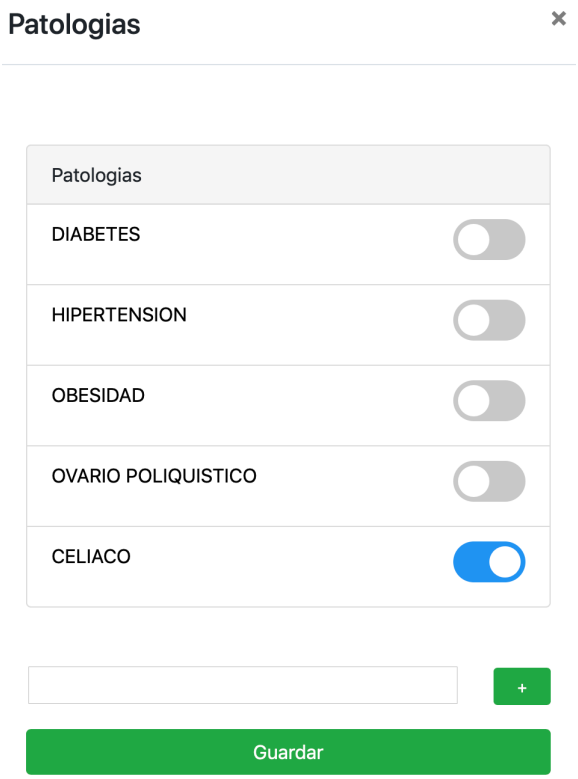
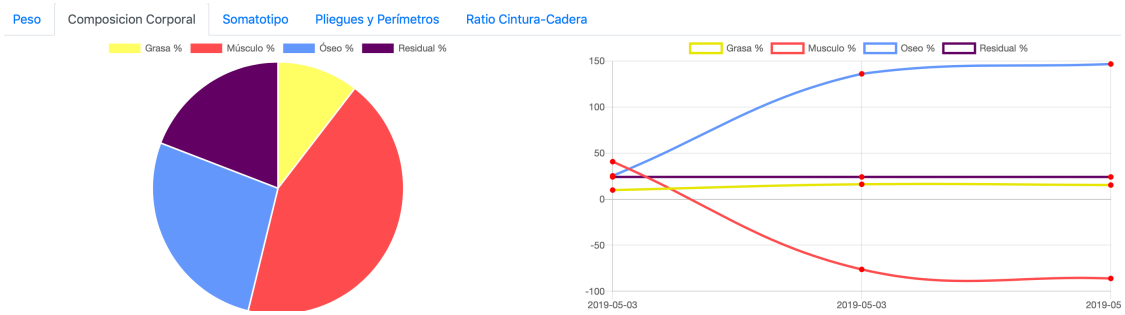


Figura 62 - Modal patologías

8. Progreso del paciente

Se permite visualizar el progreso de cualquier paciente a partir de su histórico de medidas y métricas. Para llevar a cabo esta acción, desde la ficha del paciente se pulsará el botón Progreso
Al pulsar este botón se cargará una vista donde se muestra mediante gráficas, la evolución y distintos datos del paciente representados por la *Figura 63.a* y la *Figura 63.b*. Para visualizar las diferentes gráficas, bastará con moverse por las diferentes pestañas disponibles. Para obtener más detalle de un punto en concreto, tan solo hay que mantener el puntero del ratón sobre el punto a consultar, y se cargará una pequeña nota dando más detalles sobre la información de dicho punto.

a)



b)

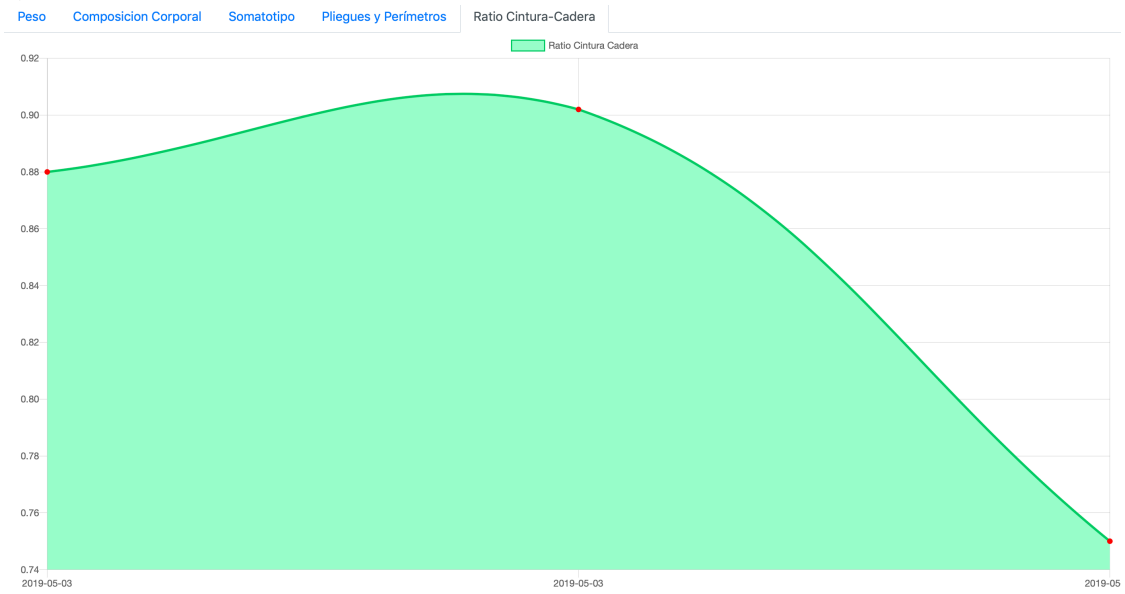


Figura 63 - a)Muestra 1 de gráfica, b)Muestra 2 de gráfica

9. Baja de un paciente

Se puede desactivar un paciente accediendo a la ficha de dicho paciente y pulsando sobre el botón Desactivar Paciente. Esto produce un cuadro de confirmación para desactivar dicho paciente. Para llevar a cabo la acción, es necesario escribir "si" en dicho campo de texto y pulsar *"Aceptar"*. En el caso de introducir cualquier otro texto, el paciente no se desactivará. Al desactivar el paciente, este pasará a formar parte de la sección de pacientes de baja en la lista de pacientes. Todos los datos del paciente se encuentran en la base de datos, pero será inaccesible para el profesional. Para reactivar al paciente, tan solo es necesario dar de alta un nuevo paciente con el email asociado a este paciente. El volver a dar de alta un paciente de baja, puede servir para cambiar sus datos, así como su contraseña.

10. Dietas predeterminadas

El profesional tiene la capacidad de crear dietas predeterminadas con el propósito de utilizarlas a la hora de crear cualquier dieta a cualquier paciente. Esto otorga la posibilidad de no tener que escribir desde cero, una dieta que se asemeja a una dieta predeterminada dieta en concreto. Para ello, desde la página principal del profesional, en el cuadro de control, existe un botón naranja con el texto *"Dietas Predeterminadas"* como muestra la *Figura 64.* Al pulsar sobre este botón, se abrirá un editor de dietas. Cualquier dieta creada en este editor, se asignará al profesional y formará parte de sus dietas predeterminadas. Al igual que con las dietas del paciente, estas dietas se pueden editar y visualizar a través de la lista de dietas disponibles en el panel izquierdo.

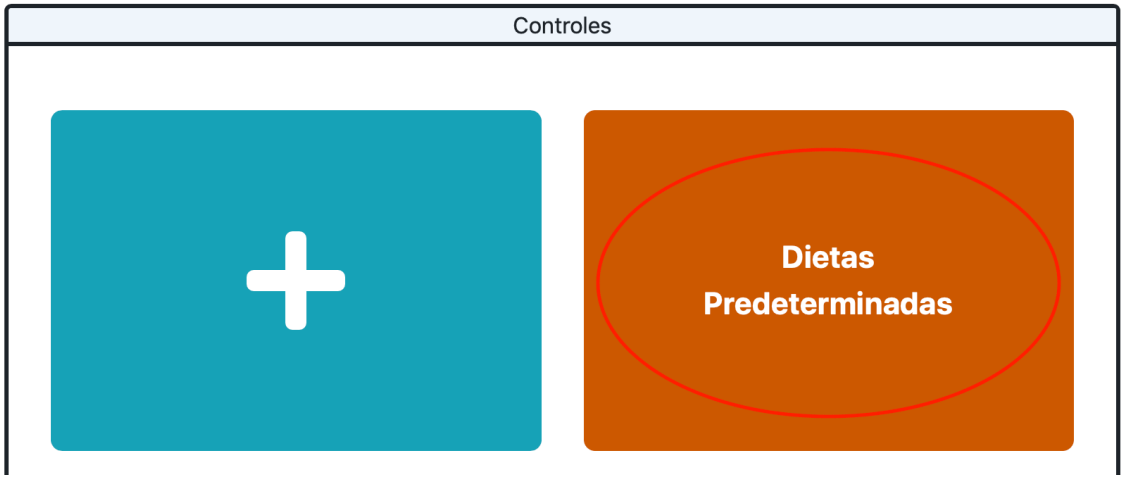


Figura 64 - Acceso a dietas predeterminadas

11. Buscador

Existe la posibilidad de buscar a un paciente en concreto de la lista de pacientes presente en la vista principal del profesional. Esta búsqueda se realiza al introducir el nombre o el apellido del paciente a buscar en el cuadro de búsqueda situado en la parte inferior de la lista de pacientes. De esta forma, se eliminarán de la lista de pacientes en tiempo real, los nombres de los pacientes que no cumplan con el término de búsqueda introducido. La vista del buscador se muestra en la *Figura 65.*

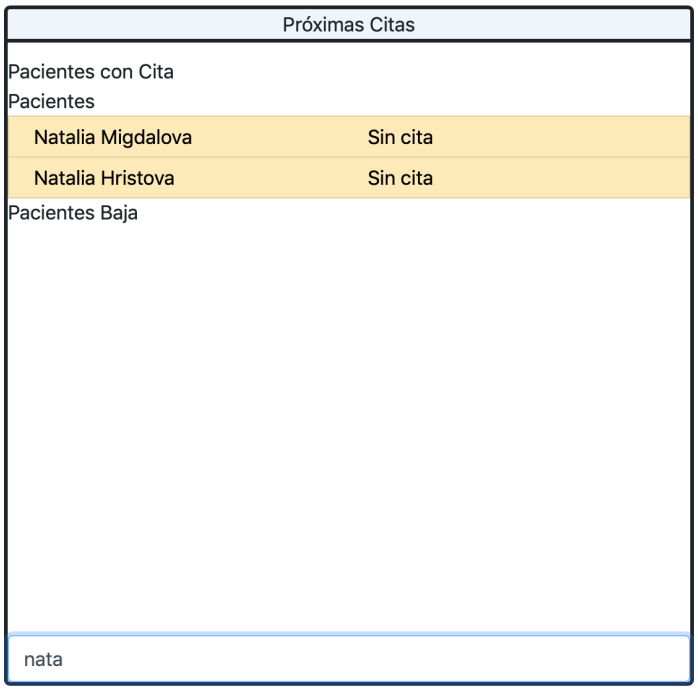
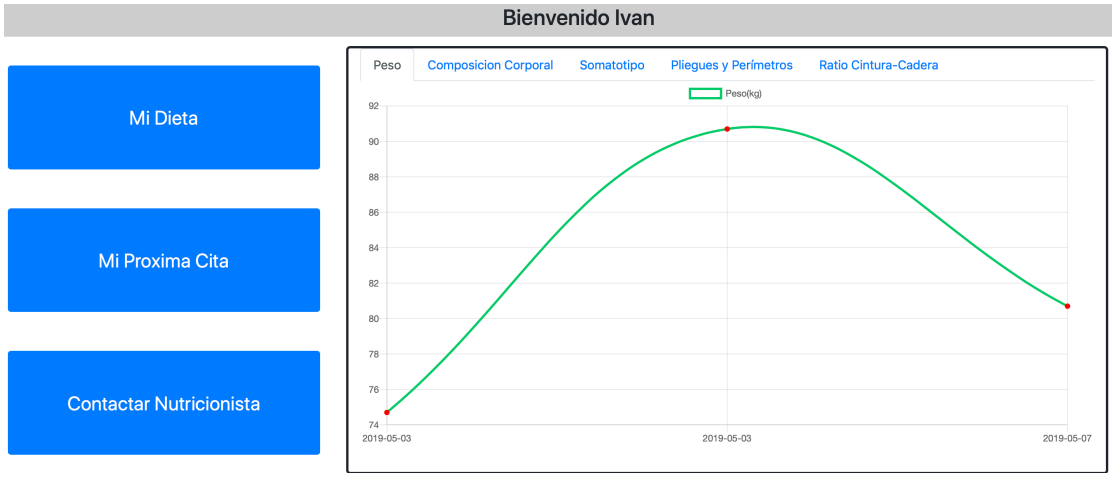


Figura 65 - Búsqueda pacientes

12. Acciones de paciente

Existe la posibilidad que el paciente dado de alta por un profesional acceda a la aplicación con las credenciales de su ficha. Para ello, desde el log-in basta con introducir las credenciales de un paciente y la aplicación cargará la vista de paciente con los datos del paciente en concreto. Esta vista se muestra en la *Figura 66.a.* Este paciente podrá consular su próxima cita pulsando en el botón Mi Proxima Cita. Al pulsar este botón se mostrará una ventana emergente donde se muestra su próxima cita asignada por el profesional. El paciente podrá visualizar también su dieta activa pulsando el botón Mi Dieta. Esto mostrará en una nueva vista correctamente formateada, la última dieta que le ha asignado su profesional como muestra la *Figura 66.b*. Otra de las funcionalidades disponibles es la de contactar con su profesional pulsando el botón Contactar Nutricionista. De esta forma, se realizará una llamada al sistema operativo, abriendo el editor de correo electrónico por defecto y colocando en la dirección de envío el email de su profesional para poder comunicarse con el a través de correo electrónico. La vista principal del paciente, consiste en la aparición de los tres botones mencionados anteriormente, y las gráficas correspondientes a su histórico de datos. Obteniendo la misma funcionalidad que en se explica en el apartado "*Progreso del paciente".*

a)



b)

Lunes

Desayuno		
Alimento	**Cantidad**	**Unidades**
fresa	2.00	unidades
galleta	200.00	unidades
leche	0.50	litros
Media Mañana		
Alimento	**Cantidad**	**Unidades**
Comida		
Alimento	**Cantidad**	**Unidades**
pollo	300.00	gramos
arroz	100.00	gramos
Merienda		
Alimento	**Cantidad**	**Unidades**
Cena		
Alimento	**Cantidad**	**Unidades**

Martes

Desayuno		
Alimento	**Cantidad**	**Unidades**
aguacate	2.00	unidades
leche	0.20	litros
Media Mañana		
Alimento	**Cantidad**	**Unidades**
Comida		
Alimento	**Cantidad**	**Unidades**
lentejas	600.00	gramos
carne	1000.00	gramos
yogurt	2.00	unidades
Merienda		
Alimento	**Cantidad**	**Unidades**
Cena		
Alimento	**Cantidad**	**Unidades**
pavo	200.00	gramos

Miercoles

Desayuno		
Alimento	**Cantidad**	**Unidades**
Media Mañana		
Alimento	**Cantidad**	**Unidades**
Comida		
Alimento	**Cantidad**	**Unidades**
Merienda		
Alimento	**Cantidad**	**Unidades**
Cena		
Alimento	**Cantidad**	**Unidades**

Jueves

Desayuno		
Alimento	**Cantidad**	**Unidades**
Media Mañana		
Alimento	**Cantidad**	**Unidades**
Comida		
Alimento	**Cantidad**	**Unidades**
Merienda		
Alimento	**Cantidad**	**Unidades**
Cena		
Alimento	**Cantidad**	**Unidades**

Figura 66 - a)Vista principal paciente, b)Vista dieta formateada

Anexo II: Guía de instalación

A continuación, se van a describir los pasos necesarios para realizar la instalación del proyecto. En primer lugar, los requisitos técnicos para poder llevar a cabo la instalación son:

- Sistema operativo Windows, OSX o Linux.
- Servidor web Apache
- PHP instalado y habilitado en el servidor web
- MySQL instalado y habilitado para enviar y recibir peticiones.
- Navegador web actualizado capaz de ejecutar JavaScript y renderizar correctamente CSS

Observar que se utilizará el mismo equipo tanto como cliente como servidor de manera que la dirección de acceso a través del navegador será desde *http://localhost*.

1. Xampp

El primer paso será el de la instalación de Xampp en el equipo. Para ello, se accede a su página web oficial[6] (*Figura 67)* seleccionando el instalador correspondiente al sistema operativo utilizado.

Figura 67 - Página principal de Xampp

Una vez descargado, se ejecuta el instalador. En un paso de la instalación (*Figura 68)* se deberán marcar las siguientes opciones: Apache, MySQL, PHP y PHPmyAdmin.

[6] www.apachefriends.org/es/index.html

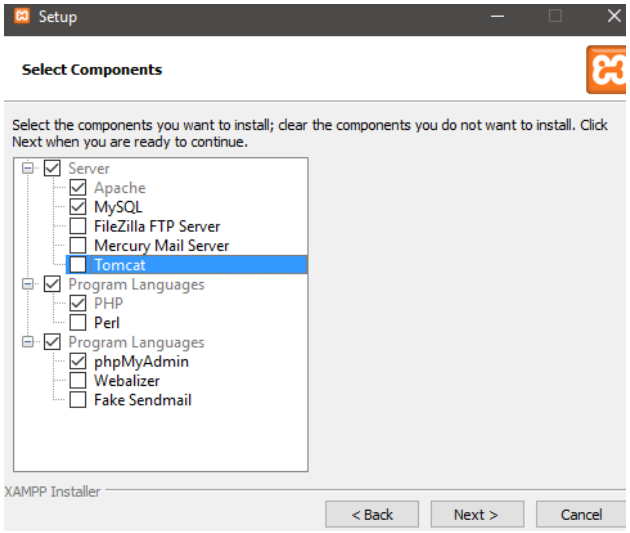


Figura 68 - Instalación Xampp

2. Arranque del servidor Apache y MariaDB

Tras la instalación de Xampp, se ejecuta para poder arrancar el servidor web, y la base de datos.

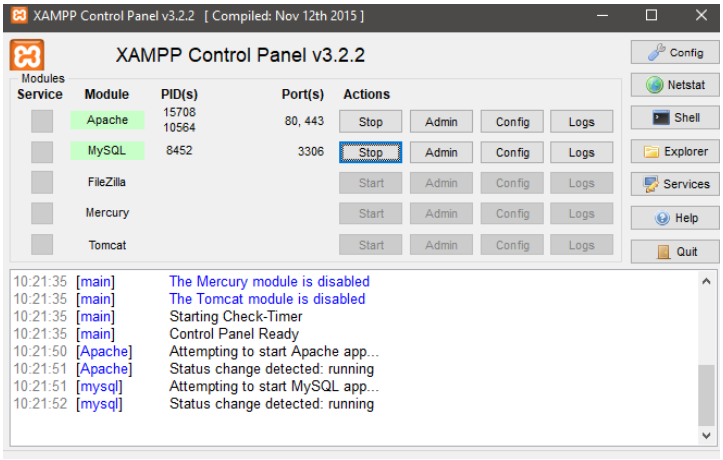


Figura 69 - Panel control Xampp

Una vez cargado el panel de control de Xampp, se debe pulsar los botones “Start” de Apache y MySQL mostrados en la *Figura 69*. En el momento en el que se vuelvan verde los cuadros de texto, podremos asegurar que los servicios están corriendo en el sistema y accesibles a través de la dirección http://localhost.
Una vez instalado Xampp, es imprescindible eliminar todo el contenido que se encuentre en el directorio *“/htdocs/”*.

En Windows con una instalación por defecto se encuentra en “*C:\xampp\htdocs*”

3. Carga de la base de datos

Para que la aplicación funcione, es imprescindible tener una base de datos con las tablas y datos acordes a los requisitos de la aplicación. Para repoblar la base de datos, abrimos un navegador

web y accediendo a la dirección "http://localhost/phpmyadmin/". Una vez dentro de phpMyAdmin, se abre la sección SQL como está señalado en la Figura 70.

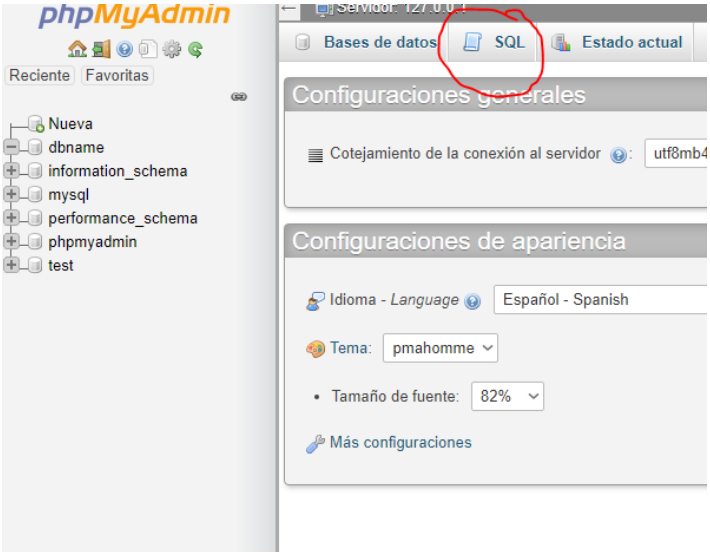


Figura 70 - Vista phpMyAdmin

Para ello, se debe descomprimir la carpeta del proyecto, obteniendo así 3 directorios y dos archivos "*.sql*". A continuación, se debe abrir con un editor de texto, el archivo disponible en la raíz del proyecto el archivo *"sql"* cuyo nombre es *"CreateTable-Datos.sql"*. Se debe copiar todo su contenido y debe ser pegado en el elemento de tipo "*textBox"* que proporciona phpMyAdmin como muestra la *Figura 71*.

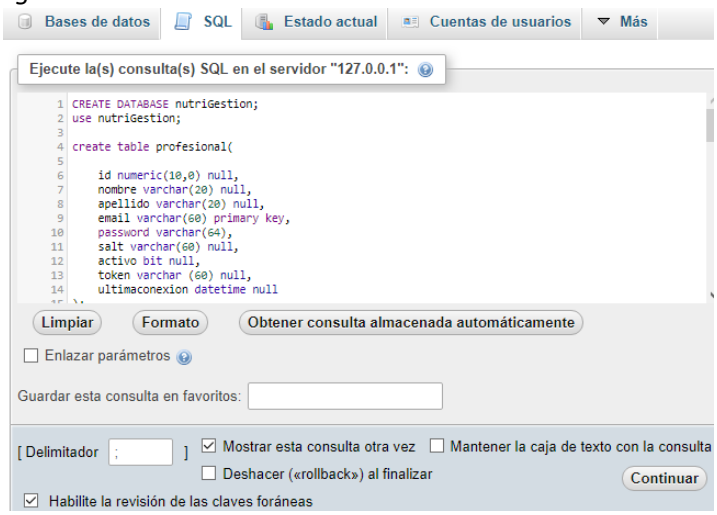


Figura 71 - Inserción datos en phpMyAdmin

Tras pulsar sobre el botón "*Continuar*" se cargarán todas las tablas y datos necesarios en la base de datos para el correcto funcionamiento de la aplicación.

4. Instalación del proyecto

Existen dos métodos de instalación del proyecto:

a) Carga del proyecto precompilado

Es posible cargar el proyecto colgándolo en la carpeta raíz del servidor apache. Para ello, hay que descomprimir la carpeta del proyecto, obteniendo así 3 directorios y dos archivos *".sql"*. Los archivos *".sql"* contienen la estructura de la base de datos, uno de ellos con datos precargados, y el otro únicamente con códigos de registro y patologías.

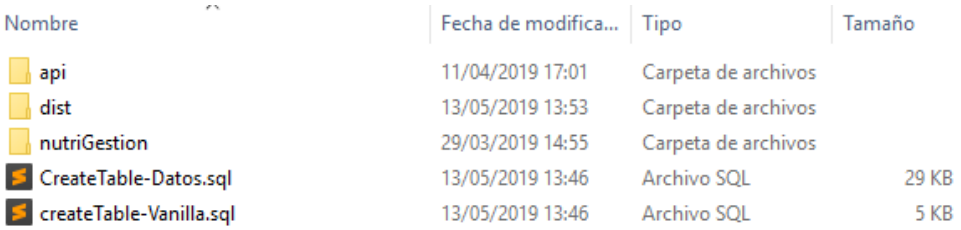


Figura 72 - Estado tras descomprimir el archivo del proyecto

La estructura de directorios de la carpeta descomprimida es la siguiente (*Figura 72*):

- **api:** Contiene los archivos PHP que se ejecutarán en el servidor perteneciente al back-end
- **dist:** Contiene el proyecto compilado y listo para ser interpretado por el servidor Apache.
- **nutriGestion:** Contiene el código fuente del proyecto. Es necesario compilarlo con Node.js para que genere otra carpeta "dist" o arrancarlo en vivo.

Una vez es descomprimido el proyecto, se mueven **todos** los archivos que contiene la carpeta *"dist/nutriGestion/"* a la carpeta "*htdocs"* de XAMPP. Esta ruta se encuentra en el directorio donde se haya instalado XAMPP. En un computador con sistema operativo Windows, por defecto es *"C:\xampp\htdocs".* De esta manera queda de la misma manera que en la *Figura 73*:

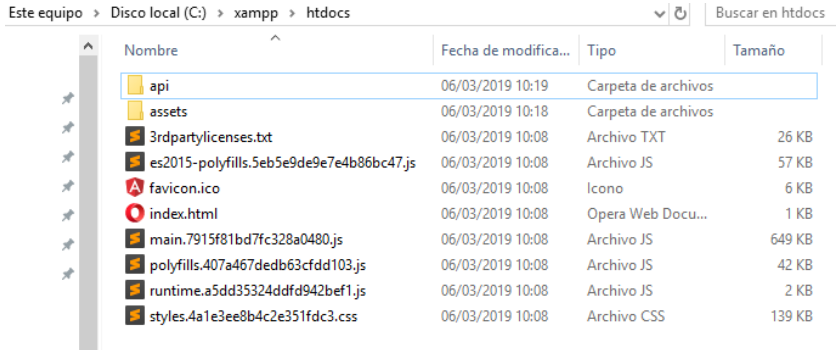


Figura 73 - Proyecto precompilado en htdocs

En este punto, la aplicación está disponible y lista para acceder a ella a través de la dirección *"http://localhost"*. Al cargar la base de datos con el archivo *"CreateTable-Datos.sql".* Se almacenan en la base de datos unos usuarios con datos precargados en la aplicación. Para utilizar la aplicación con estos datos, las credenciales de log-in son las siguientes:

Profesional:

- Profesor
  - Correo electrónico: profesor@ucm.es
  - Contraseña: profesor

Paciente:

- Iván
    - Correo electrónico: icanas@ucm.es
    - Contraseña: ivan
- Natalia
    - Correo electrónico: nataliah@ucm.es
    - Contraseña: nataliah
- Ana 1
    - Correo electrónico: ananadal@ucm.es
    - Contraseña: ananadal
- Ana 2
    - Correo electrónico: anaramosl@ucm.es
    - Contraseña: anaramos
- Adrián
    - Correo electrónico: adrianlopez@ucm.es
    - Contraseña: adrianlopez
- Julio (desactivado)
    - Correo electrónico: juliohernandez@ucm.es
    - Contraseña: juliohernandez

Códigos de registro para nuevos profesionales:

- 123
- 124
- 125
- 126
- 127

b) Carga del proyecto en vivo

Además de todos los requisitos mencionados en la primera página, son necesarios:

- La última versión LTS de Node.js https://nodejs.org/en/
- Instalar Angular a través de comandos Node.js https://angular.io/guide/quickstart
    1. Abrir una consola.
    2. Acceder en la consola a: {ruta del proyecto descomprimido}/nutrigestion
    3. Tecleamos en la consola -> `npm install -g @angular/cli`
    4. Pulsar *"Enter"*
    5. Es recomendable tener algún editor de textos completo como puede ser *"Sublime Text"* o "V*isual Studio Code".*
    6. Si tenemos a disposición un editor de textos, abrimos con el editor de textos, la carpeta entera de nuestro proyecto llamada "src".

Una vez se cumplen los requisitos, el usuario está preparado para poder compilar y realizar cambios en directo en el proyecto.

5. Ejecución en vivo

Para la compilación de los cambios en vivo, el usuario debe abrir una consola del sistema y posicionarla en la carpeta raíz del proyecto descomprimido *(nutrigestion/)*. Una vez en la carpeta raíz del proyecto, se debe ejecutar el siguiente comando: `ng serve –open` .De esta forma, y tras esperar un tiempo hasta que cargue el proyecto, se abrirá un navegador web el cual

automáticamente accede a la dirección http://localhost:4200/login , quedando de la manera que muestra la *Figura 74*:

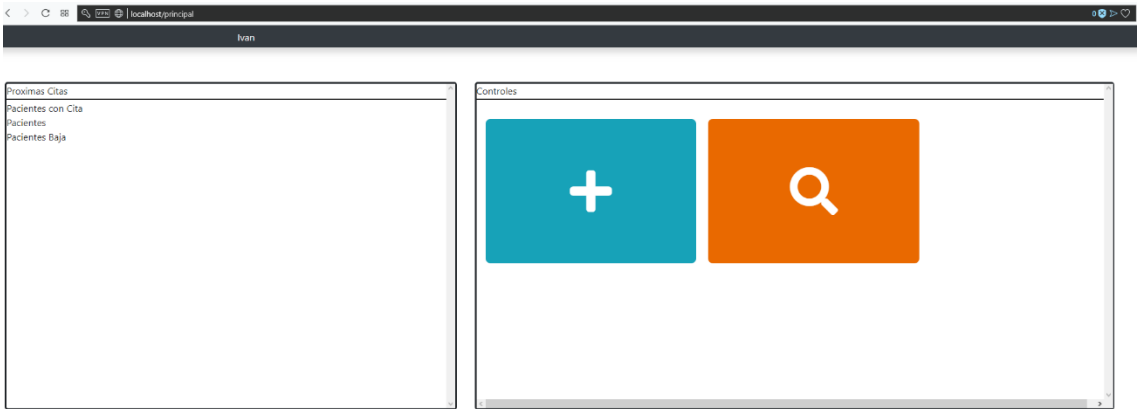


Figura 74 - Proyecto cargado en localhost:4200

En este punto, y gracias a Node.js, cualquier cambio realizado en los archivos de código fuente del proyecto, se verán reflejados en tiempo real en la aplicación.

6. Compilación

A diferencia de la compilación en vivo, la compilación estática se encargará de compilar el proyecto para cargarlo en un servidor de producción. Para realizar esta compilación estática, se debe abrir una consola del sistema y posicionarla en la carpeta raíz del proyecto. A continuación, se escribe el siguiente comando: `ng build –prod`
Este comando es el encargado de compilar el proyecto. Tras unos minutos de compilación, generará un directorio "/dist/nutriGestion/" en la raíz del proyecto, con todos los archivos compilados y necesarios para la ejecución en un servidor de producción.

Anexo III: Métricas

Las métricas son las relaciones numéricas que entre las medidas y los datos de un paciente. En esta aplicación se han utilizado unas fórmulas matemáticas específicas para el cálculo de estas métricas:

- **Índice de masa corporal:** $\frac{peso\ (kg)}{altura^2(m)}$
- **Ratio cintura – cadera:** $\frac{perímetro\ cintura}{perímetro\ cadera}$
- **Suma de los 6 pliegues:** $pliegue\ abdominal(mm)\ +\ pliegue\ muslo(mm)\ +\ pliegue\ pierna(mm)\ +pliegue\ subescapular(mm)\ +\ pliegue\ supraespinal\ (mm)+\ pliegue\ tríceps(mm)$
- **Suma de los 8 pliegues:** $pliegue\ abdominal\ (mm)+\ pliegue\ muslo(mm)\ +\ pliegue\ pierna(mm)\ +pliegue\ subescapular(mm)\ +\ pliegue\ supraespinal\ (mm)+\ pliegue\ tríceps\ (mm)+\ pliegue\ bíceps(mm)\ +\ pliegue\ cresta\ ilíaca(mm)$
- **Porcentaje grasa hombre** [19]: $0.1051\ \times\ suma6Pliegues(mm)\ +\ 2.585$
- **Porcentaje grasa mujer** [19]: $0.1548\ \times\ suma6Pliegues(mm)\ +\ 3.58$
- **Masa Ósea**[13]: $3.02\ \times\ altura(cm)^2\ \times\ Perímetro\ muñeca\ (m)\ \times\ (Diametro\ BiepicondilarFemur(m)\ \times\ 400)^{\ 2}$
- **Masa Residual hombre** [13]: $peso(kg)\ \times\ 0.241$
- **Masa Residual mujer** [13]: $peso(kg)\times 0.209$
- **Endomorfo:** $(-0.7182\ +\ (0.1451\ \times\ (pliegue\ tríceps(mm)\ +\ pliegue\ subescapular(mm)\ +\ pliegue\ supraespinal(mm))\ \times\ \frac{170.18}{altura})^2\ +\ 0.000004\ \times\ (pliegue\ tríceps(mm)\ +\ pliegue\ subescapular\ (mm)+\ pliegue\ supraespinal(mm))\ \times\ \frac{170.18}{altura})^3\ )$
- **Mesomorfo:** $0.858\ \times\ diámetro\ húmero(cm)\ +\ 0.601\ \times\ diámetro\ biepicondilar\ fémur(cm)\ +\ 0.188\ \times\ (perímetro\ brazo\ flexionado\ (cm)-\frac{pliegue\ tríceps(mm)}{10})\ +\ 0.161\ \times\ (perímetro\ pierna(cm)\ -\ \frac{pliegue\ pierna(mm)}{10})\ -\ 0.131\ \times altura(m)\ +\ 4.5$
- **IP:** $\sqrt[3]{\frac{altura(cm)}{peso(kg)}}$
- **Ectomorfo IP >= 40.75:** $0.732\ \times\ IP\ -\ 28.58$
- **Ectomorfo IP < 40.75 y IP 38.25:** $0.463\ \times\ IP\ -\ 17.63$
- **Ectomorfo IP < 38.25:** 0.1
- **Somatotipo** [9]: max (endomorfo, mesomorfo, ectomorfo)