Grado en Ingeniería Informática

Trabajo de Final de Grado

Sistema de ayuda a las pruebas de fragilidad

Autor:
Arturo Gascó Compte

Supervisor:
Sergio Aguado González
Tutor académico:
Reyes Grangel Seguer

Fecha de lectura: 22 de JUNIO de 2022
Curso académico 2021/2022

Resumen

La fragilidad es un estado de pre-discapacidad de riesgo de desarrollar una minusvalía que afecta sobre todo a las personas mayores. La detección temprana de la fragilidad permite identificar a las personas que aún poseen su independencia pero que se encuentran en una situación de pérdida funcional. En este documento se presenta el desarrollo de una aplicación informática para digitalizar el proceso que conlleva realizar las pruebas físicas para detectar el estado de fragilidad. La finalidad de esta aplicación es ayudar a los pacientes a llevar a cabo las pruebas físicas y a realizar un seguimiento de su estado de fragilidad a través de su teléfono móvil. Las principales tecnologías usadas para el desarrollo de la aplicación han sido Quarkus para la lógica del servidor y Flutter para la interfaz móvil.

Palabras clave

Fragilidad, Personas mayores, Quarkus, Flutter

Keywords

Frailty, Eldery people, Quarkus, Flutter

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El aumento de la esperanza de vida de las personas en los últimos años ha provocado un claro crecimiento del envejecimiento de la sociedad española. Con este proceso de envejecimiento aumentan los riesgos de sufrir alguna clase de enfermedad crónica, discapacidad o dependencia. La fragilidad [19, 20] es un estado de pre-discapacidad de riesgo de desarrollar discapacidad a partir de una situación de limitación funcional repetitiva. La detección temprana de la fragilidad permite identificar a las personas que aún poseen su independencia pero que se encuentran en una situación de pérdida funcional.

El concepto de fragilidad incluye componentes físicos, sociales y psicológicos. Entre ellos, la fragilidad física es el componente más investigado y se ha demostrado en muchos estudios que predice la discapacidad, la hospitalización y la mortalidad [5, 14, 22]. Por otra parte, aunque aún no se ha establecido una evaluación de referencia para la fragilidad social, varios estudios han demostrado los resultados adversos para la salud atribuibles a la fragilidad social, como la discapacidad y la mortalidad [15, 23]. En cuanto a la fragilidad psicológica, diversos estudios [6, 7] demuestran que está altamente ligada con la sensación de sentirse triste, alicaído/a o lo que comúnmente se conoce como 'estar de bajón'.

Por lo descrito anteriormente, la empresa Soluciones Cuatroochenta [3] ha decidido desarrollar un sistema informático para la digitalización y automatización del proceso de detección de fragilidad, que actualmente se realiza con el método tradicional de papel y lápiz. La empresa está especializada en desarrollar e implementar soluciones digitales cloud y de ciberseguridad [4], y desde hace unos años ha impulsado diferentes proyectos relacionados con la salud y el bienestar de las personas mayores.

El proyecto en el que trabaja actualmente la empresa se denomina Frailty Tests Assistance System (FTAS). Este proyecto pretende facilitar el proceso que actualmente conlleva evaluar a las personas mayores con síntomas de fragilidad. Para detectar los síntomas, se realizan tres pruebas físicas donde se miden diferentes parámetros dependiendo de la prueba. En la primera prueba, el paciente tiene que caminar tres metros en línea recta en el menor tiempo posible. En

la segunda, se facilita una silla al paciente y se le solicita realizar cinco sentadillas, de nuevo en el menor tiempo posible. Por último, en la tercera se indica al paciente que tiene que mantener el equilibrio con los pies juntos, esta vez durante el máximo tiempo posible. Todas estas pruebas son medidas por el médico de forma presencial con la ayuda de un simple cronómetro y son almacenadas más tarde en una base de datos de forma manual.

Teniendo en cuenta lo anteriormente descrito, la empresa propone en su proyecto digitalizar todo el proceso detallado debido a que actualmente es muy tedioso, especialmente para el profesional de la salud. El proyecto se estructura en tres subsistemas principales.

El primer subsistema debe permitir al médico o profesional clínico registrar y consultar el estado de sus pacientes. Esto es, el estado de fragilidad, la evolución en el tiempo, las pruebas físicas realizadas y los cuestionarios psicológicos contestados. Además, debe permitir al profesional introducir nuevos cuestionarios en el sistema. Esta funcionalidad debe ser desarrollada para un entorno web, ya que es el sistema habitual de trabajo de los médicos o profesionales clínicos. Cabe destacar que el registro de los pacientes debe ser llevado a cabo por los profesionales de la salud.

El segundo subsistema debe permitir a los pacientes o a sus cuidadores ver el tiempo empleado en realizar las diferentes pruebas físicas a lo largo del tiempo y debe posibilitar tanto la introducción manual de los datos obtenidos al realizar las pruebas físicas, como otros datos personales como el peso, la altura o la fuerza de agarre. Además, debe permitir que los pacientes puedan contestar a los cuestionarios psicológicos introducidos en el sistema. Toda esta funcionalidad ha de realizarse a través de una aplicación móvil.

El tercer subsistema debe permitir a los pacientes la posibilidad de realizar las diferentes pruebas físicas de manera autónoma y automática. Para ello, el entorno utilizado deberá ser una aplicación de escritorio para Windows con hardware específico. Esta aplicación hará uso de una cámara Microsoft Kinect v2 [16] que es capaz de hacer una detección y seguimiento del cuerpo humano en tiempo real para, de esta manera, digitalizar las pruebas físicas.

El trabajo descrito en este documento aborda el segundo subsistema mencionado y por lo tanto, quedan fuera del alcance tanto el primer como el tercer subsistema.

1.2. Objetivos del proyecto

El principal objetivo de la aplicación es poner a disposición de las personas mayores un sistema informático de monitorización que permita un mejor manejo y tratamiento de la fragilidad, con el fin de evitar una discapacidad física de las personas mayores. Este objetivo se puede desglosar en los siguientes apartados:

- Digitalizar el proceso de recolección de datos de los pacientes en los centros médicos y hospitalarios, así como desde el propio domicilio del paciente.
- Facilitar a los pacientes la inserción de datos manuales relativos a las pruebas físicas.
- Proporcionar a los pacientes la posibilidad de insertar datos manuales de carácter personal.

- Ofrecer a los pacientes la posibilidad de responder a diferentes cuestionarios de carácter cognitivo.
- Ayudar a los pacientes a hacer un seguimiento de su evolución de fragilidad.
- Posibilitar la identificación de los pacientes a través de su teléfono móvil en los distintos dispositivos donde se realizan las pruebas físicas.

El **alcance funcional** del subsistema desarrollado en este proyecto tiene cinco funcionalidades diferenciadas que debe proporcionar y que se describen a continuación.

- El paciente debe validar su usuario mediante el documento nacional de identidad para poder acceder al resto de funcionalidades.
- El paciente puede introducir de manera manual cualquier dato recogido habitualmente de forma automática al realizar las pruebas físicas. Estos datos incluyen el tiempo empleado en realizar la prueba, la fecha, la velocidad media, etc. Además, también puede introducir datos más generales como el peso, la altura o la fuerza de agarre.
- El paciente puede ver un gráfico sencillo con el tiempo medio empleado en la realización de cada una de las pruebas. De esta manera es capaz de hacer un seguimiento y ver la evolución de las pruebas realizadas.
- El paciente tiene la posibilidad de contestar a cuestionarios cognitivos acerca del estado de fragilidad u otros aspectos, que el médico ha introducido previamente en el sistema.
- La aplicación del paciente dispone de un escáner QR que se usa para leer un código disponible en los dispositivos con la cámara Kinect v2 instalada, y que permite la identificación del paciente en estos dispositivos.

Por lo que respecta al **alcance organizativo**, los usuarios que van a usar el subsistema desarrollado en este proyecto serán los pacientes con riesgo de fragilidad dados de alta por el profesional clínico o médico. En el caso de que el paciente sea dependiente, la aplicación podrá ser usada de la misma manera por el cuidador que esté a su cargo. De esta manera, será el cuidador el que podrá realizar todas las funcionalidades descritas anteriormente.

En relación al **alcance informático**, para facilitar la validación del paciente en un dispositivo con una cámara Kinect v2 mediante la aplicación móvil, se ha usado un código QR. La aplicación móvil escanea el código QR que contiene el identificador de la cámara. Este dato junto al identificador del usuario son enviados al servidor. Una vez recibidos en el servidor, estos datos son enviados de nuevo al dispositivo escaneado para que sea notificado que un usuario ha accedido a él. Para esta comunicación entre servidor y dispositivo con la cámara Kinect v2 se usa el protocolo de comunicación MQTT [18], que se caracteriza por ser un protocolo sencillo y ligero basado en la arquitectura publicación/suscripción. Por otra parte, para realizar el inicio de sesión del paciente en la aplicación móvil se envía el DNI introducido al servidor y se comprueba si existe en la base de datos. Esta comunicación se realiza mediante peticiones HTTP. Cabe destacar que los pacientes son registrados por el profesional médico en el primer subsistema descrito en la sección anterior.

1.3. Descripción del proyecto

El estado de fragilidad se puede dividir en fragilidad física, fragilidad social y fragilidad cognitiva o psicológica. La fragilidad física es la más conocida y se intenta diagnosticar, entre otros métodos, mediante la realización de las tres pruebas físicas descritas anteriormente. La fragilidad social y la fragilidad cognitiva se diagnostican mediante la realización de diferentes cuestionarios cognitivos por parte del paciente. Es el profesional médico el que se encarga de asignar al paciente qué cuestionario debe realizar. Mediante la aplicación desarrollada en este proyecto se abordan los tres tipos de fragilidad. Por una parte, la aplicación permite introducir datos de forma manual referentes a las pruebas físicas. Además, tiene la capacidad de escanear un código QR para ayudar a identificar al paciente en el dispositivo donde se van a realizar las pruebas. Por otra parte, la aplicación proporciona la posibilidad de responder distintos cuestionarios cognitivos que han sido asignados al paciente por parte del médico en el primer subsistema.

La aplicación presentada en este documento está basada en el segundo subsistema descrito anteriormente, quedando fuera del alcance tanto el subsistema que permite controlar el estado de los pacientes al profesional médico, como el subsistema que permite la digitalización y automatización de las pruebas físicas de fragilidad a través de la cámara Kinect v2. Por lo tanto, el proyecto se centra en el segundo subsistema donde se ha desarrollado una aplicación móvil, anteriormente descrita, para los pacientes.

Mediante la unión de los tres subsistemas se espera que las personas mayores con altas probabilidades de sufrir fragilidad consigan revertir este estado. Además, también se pretende que los pacientes puedan realizar un seguimiento de su estado para que sean conscientes del mismo. También se espera que pueda servir de ayuda al profesional médico, no solo para hacer un seguimiento de la evolución de sus pacientes, sino para permitir realizar intervenciones tempranas para prevenir la discapacidad y la dependencia del paciente.

Por último, a continuación se detallan los aspectos tecnológicos más importantes del subsistema desarrollado.

Por un lado, para las validaciones de los usuarios, almacenamiento de los datos y tratamiento de la información es necesario interactuar con un programa servidor. Este servidor es común a los tres subsistemas descritos y por lo tanto, además del segundo subsistema detallado en este documento, tanto el primer como el tercer subsistema necesitan tener acceso e interactuar con él. El programa servidor ha sido desarrollado mediante Quarkus [12], un framework creado en Kubernetes [10] que utiliza el lenguaje de programación Java y permite un despliegue en contenedores de una forma muy rápida. Además, para el almacenamiento de los datos se ha utilizado por una parte una base de datos PostgreSQL [21] para los datos generales y otra base de datos Cassandra [2] para guardar las estructuras de datos que representan el cuerpo humano mediante puntos o articulaciones.

Por otro lado, la aplicación móvil se debe poder instalar en los dispositivos de los pacientes o sus cuidadores sin importar el sistema operativo. Por esta razón, la herramienta elegida para el desarrollo fue Flutter [11], un framework de código abierto de Google para crear aplicaciones compiladas de forma nativa y multiplataforma a partir de un único código base. El lenguaje de programación que usa este framework es Dart [8].

Por último, para realizar el despliegue de los gestores de bases de datos, el servicio MQTT y el programa servidor se ha utilizado la herramienta Docker [13].

1.4. Estructura de la memoria

Este documento está estructurado de la siguiente manera. Además del Capítulo 1 en el cual se incluye la motivación, objetivo y descripción del proyecto, siguen el Capítulo 2 donde se muestra la metodología, planificación y seguimiento del proyecto. En el Capítulo 3 se detalla el análisis del sistema, la arquitectura utilizada y el diseño de la interfaz de la aplicación móvil. En el Capítulo 4 se describen los detalles de la implementación y las pruebas realizadas para verificar y validar la aplicación. Por último, en el Capítulo 5 se expone la experiencia obtenida por el autor del proyecto descrito en este documento en el ámbito formativo, en el ámbito profesional y en el ámbito personal.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Para gestionar el desarrollo del proyecto, se ha empleado una metodología ágil, concretamente Scrum [17], debido a dos razones principales. Por una parte, la empresa está acostumbrada a este tipo de metodología, ya que es la que usan en otros proyectos de manera habitual. Por otra parte, el proyecto presentado en este documento no estaba completamente definido al inicio y ha variado durante el desarrollo del mismo, por lo que el uso de una metodología ágil era una buena opción.

Una de las características a destacar de la metodología ágil que se ha usado son los tres roles principales que se definen: el propietario del producto, el scrum manager y el equipo de desarrollo. Para este proyecto, tanto el rol del propietario del producto como el rol del scrum manager ha sido llevado a cabo por el supervisor de la empresa, Sergio Aguado. El cuál ha sido el encargado de planificar las reuniones correspondientes a la metodología ágil, es decir, las reuniones diarias, las de planificación del sprint y las de revisión.

Otra de las características de la metodología ágil ya mencionada son las iteraciones o sprints. En este caso, el proyecto ha sido desarrollado mediante sprints de dos semanas de duración. Dado que las horas totales de la estancia en prácticas son 300 y que cada día laboral ha sido de seis horas, el número total de sprints realizados ha sido de cinco.

Por otra parte, las reuniones diarias se han llevado a cabo tres días a la semana: lunes, miércoles y jueves. En caso de inicio de un nuevo sprint, el primer lunes se realizaba una reunión de planificación para definir los principales objetivos del sprint, estimar futuras tareas y dialogar sobre las historias de usuario que se incorporaban al nuevo sprint. Las reuniones diarias se destinaron a conocer el trabajo realizado, los problemas que se habían encontrado y cuál era el trabajo que se iba a realizar hasta la siguiente reunión diaria. Cabe destacar que en estas reuniones también participó el alumno en prácticas Miguel Pardo, cuyo trabajo consistía en desarrollar el tercer subsistema del proyecto FTAS, con la aplicación de escritorio conectada a una cámara Kinect v2. La comunicación con Miguel ha sido muy importante, ya que él ha hecho uso del programa servidor que se ha desarrollado en este proyecto. Concretamente, el

subsistema que ha desarrollado Miguel necesita obtener el DNI del paciente que está accediendo al dispositivo. Además, al terminar una prueba física la aplicación de escritorio desarrollada por Miguel envía al servidor todos los datos recogidos durante la realización de la prueba para ser almacenados en la base de datos.

Finalmente, los lunes de cada dos semanas, al acabar el sprint, se realizaba una reunión de revisión del sprint completado en la cual se comentaba la carga de trabajo asignada, se revisaban las historias de usuario para comprobar si tenían la funcionalidad totalmente implementada para poder darlas por cerradas y se obtenía una visión global del estado del proyecto a través de una demo o aplicación de prueba.

2.2. Planificación

A continuación, en el Cuadro 2.1 se presenta la pila del producto con las historias de usuario iniciales definidas en el proyecto. Dado que la metodología usada para el desarrollo del proyecto ha sido Scrum, la pila del producto se definió teniendo en cuenta la posibilidad de que durante el desarrollo del proyecto se pudieran añadir nuevas historias de usuario. Para poder abordar todo el trabajo, para cada historia de usuario se definieron tareas que había que llevar a cabo hasta poder dar por finalizada dicha historia de usuario.

ID	Descripción	Puntos de historia
HU01	Como paciente quiero validar mi usuario mediante el DNI para acceder a la aplicación móvil.	13
HU02	Como paciente quiero ser identificado en la aplicación con la cámara mediante un código QR para poder realizar las pruebas físicas.	21
HU03	Como paciente quiero ser capaz de introducir manualmente los datos obtenidos al realizar las pruebas físicas para tener un forma alternativa de registrar los datos obtenidos.	21
HU04	Como paciente quiero ser capaz de introducir manualmente datos generales sobre mi salud, como el peso, la altura y la fuerza de agarre para poder generar datos que ayuden a diagnosticar mi estado de fragilidad.	21
HU05	Como paciente quiero ver los cuestionarios disponibles en el sistema para poder seleccionar uno para responder.	8
HU06	Como paciente quiero responder a los diferentes cuestionarios psicológicos disponibles en el sistema para aportar datos que ayuden a diagnosticar mi estado de fragilidad.	13
HU07	Como paciente quiero que todos los datos introducidos manualmente sean enviados al servidor para su posterior almacenamiento.	34
HU08	Como paciente quiero consultar el tiempo medio empleado en realizar cada una de las pruebas físicas para poder ver mi evolución en el tiempo.	21

Cuadro 2.1: Pila del producto inicial definida para el proyecto.

Para gestionar el desarrollo ágil, se ha usado la herramienta de gestión de proyectos Jira. El resto de imágenes de esta sección han sido tomadas en esta herramienta de gestión. En la Figura 2.1 se presentan la pila del producto inicial con las historias de usuario descritas en el Cuadro 2.1 junto con algunas tareas extra necesarias para alcanzar los objetivos. Como se puede observar el el Cuadro 2.1, las historias de usuario han sido estimadas en puntos de historia y a todas las tareas se les ha asignado una prioridad según la preferencia del propietario del producto.

Figura 2.1: Pila del producto inicial del proyecto en Jira.

Finalmente, en la Figura 2.2 se muestra una imagen del tablero Kanban del primer sprint definido para el desarrollo del proyecto. Como se puede observar, se definieron tres estados para una tarea o historia: por hacer, en progreso y hecho.

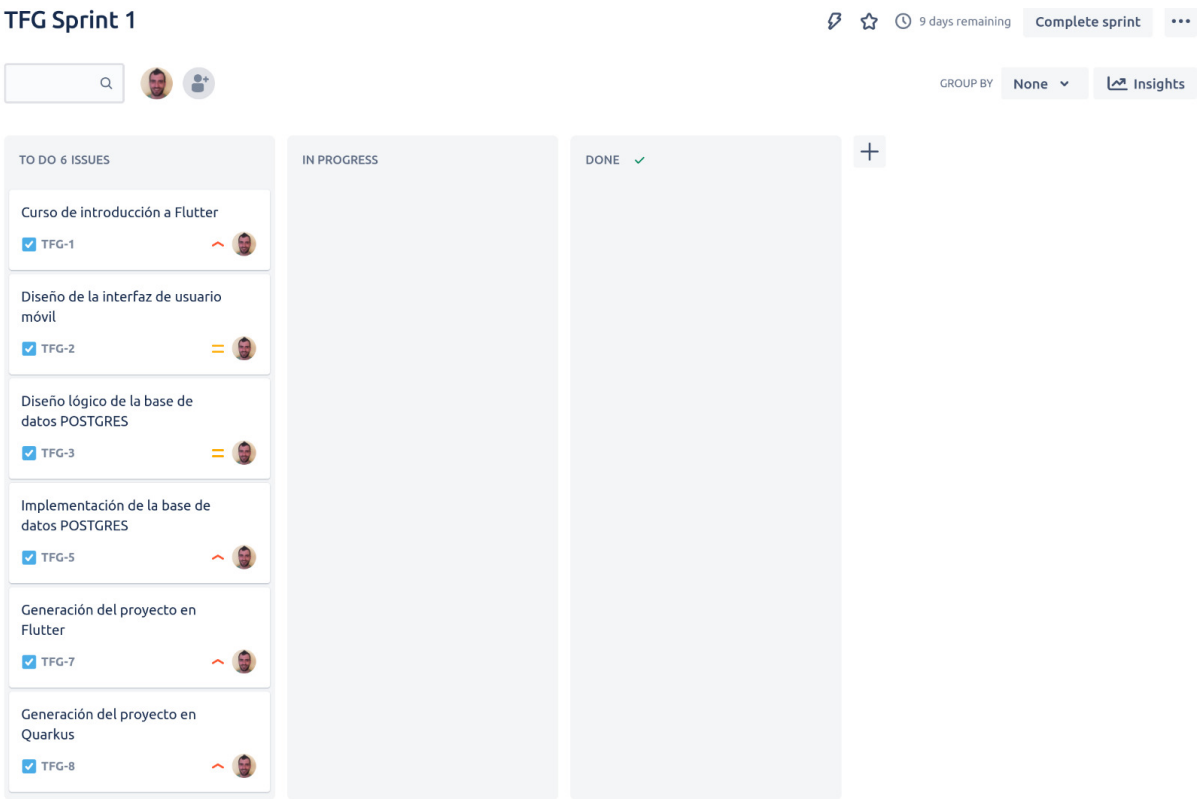

Figura 2.2: Tablero Kanban para el primer sprint del proyecto.

2.3. Estimación de recursos y costes del proyecto

En esta sección se detallan los recursos utilizados para la realización del proyecto, así como el cálculo detallado de sus costes.

2.3.1. Recursos

En esta sección se describen los recursos humanos, hardware y software necesarios para llevar a cabo el desarrollo del proyecto.

Al tratarse de un proyecto correspondiente al Trabajo Final del Grado en Ingeniería Informática, y por lo tanto ser un trabajo individual, el autor de este documento y su supervisor son los únicos recursos humanos asignados al proyecto.

Por lo que respecta a los recursos hardware, teniendo en cuenta que la aplicación debe ser probada tanto en dispositivos con el sistema operativo Android como iOS, se ha utilizado un ordenador con el sistema operativo macOS instalado para facilitar la emulación de ambos tipos de dispositivo. Los requisitos mínimos de este ordenador deben ser de 4GB de RAM para poder emular los dispositivos de forma eficiente, un procesador Intel o AMD de 64 bits con al menos 3.2 GHz y un conector USB 3.1 para conectar dispositivos físicos.

Por otra lado, en lo que se refiere a los recursos software, los entornos que se han utilizado para el desarrollo son Jetbrains IntelliJ para el desarrollo del servidor en Quarkus, ya que este entorno está preparado y especializado para el desarrollo de aplicaciones con el lenguaje de programación Java, y Microsoft Visual Studio Code para la aplicación móvil debido a su adaptabilidad y rapidez. Además, se ha empleado la herramienta Figma para el diseño de interfaces, ya que posee algunas características que ayudan a la posterior implementación del código fuente en Flutter. Por último, para probar las rutas de envío y consulta de información al servidor se ha utilizado Postman, ya que permite realizar de manera muy sencilla peticiones HTTP para probar el envío y respuesta de las consultas necesarias.

2.3.2. Costes

En primer lugar, la estimación del coste de recursos humanos se ha realizado de la siguiente manera. El salario medio de un programador en España es de 29.800€ brutos por año, alrededor de 2.128€ al mes o 12€ la hora. Dado que el autor de este documento ha sido el único programador y que el tiempo de desarrollo ha sido de 300 horas, el coste asciende a unos 3.600€ aproximadamente como se observa en la Expresión 2.1.

$$\text{Salario bruto programador} = 12€/hora * 300 \text{ horas} = 3.600€ \tag{2.1}$$

Sin embargo, hay que tener en cuenta que este valor es solo el salario bruto que recibiría el trabajador. También se deben añadir los costes asociados a la Seguridad Social a cargo de la empresa. Este valor oscila entre un 20 y 30 por ciento del coste expresado, siendo por tanto las Expresiones 2.2 y 2.3 el valor real del coste mensual y total respectivamente de los recursos humanos que se deben tener en cuenta.

$$\text{Coste mensual RRHH} = 2.128€/mes + (2.128€/mes * 0,3) = 2.766€/mes \tag{2.2}$$

$$\text{Coste total RRHH} = 3.600€ + (3.600€ * 0,3) = 4.680€ \tag{2.3}$$

Finalmente, una vez calculados los costes asociados a los recursos humanos, es necesario también tener en cuenta otros gastos y costes indirectos como el servidor de la base de datos, luz, agua y el centro de trabajo, además de los recursos hardware y software de desarrollo.

El coste de la luz y el agua se estima en 150€/mes basados en los registros históricos de la empresa. Además, el espacio de trabajo tiene un coste mensual de 700€. Teniendo en cuenta que el espacio de trabajo está pensado para 90 personas, la parte proporcional asignada a este proyecto es de aproximadamente 8€.

La justificación de los costes amortizados se detalla a continuación. En primer lugar, la Expresión 2.4 muestra el cálculo del coste amortizado del ordenador teniendo en cuenta que el coste del Apple Macbook Air utilizado para el desarrollo del proyecto es de 1.129€ y que se espera amortizar el ordenador en un periodo de cinco años.

$$\text{Coste ordenador} = 1.129€/(5 \text{ años} * 12 \text{ meses}) = 18,82€/mes \tag{2.4}$$

En segundo lugar, el servidor para alojar la base de datos tiene un coste medio de 4.000€ y se espera amortizarlo en un periodo de tres años. En la Expresión 2.5 se detalla el cálculo realizado como en el caso anterior.

$$\text{Coste servidor} = 4.000€/(3 \text{ años} * 12 \text{ meses}) = 112€/mes \tag{2.5}$$

En tercer lugar, el entorno de programación IntellIJ utilizado para el desarrollo del programa servidor necesita una licencia de uso cuyo valor es de 499€. Dado que el periodo de licencia es de un año, el coste amortizado de este recurso asciende a un total de 42€/mes como se puede apreciar en la Expresión 2.6.

$$\text{Coste entorno de programación} = 499€/(1 \text{ año} * 12 \text{ meses}) = 42€/mes \tag{2.6}$$

En cuarto lugar, el precio del curso de Flutter realizado para la adquisición de los conocimientos necesarios para el desarrollo del proyecto tiene un valor de 94,99€. La Expresión 2.7 muestra el coste mensual teniendo en cuenta que la empresa espera amortizar el curso en tres años.

$$\text{Coste curso Flutter} = 94,99€/(3 \text{ año} * 12 \text{ meses}) = 3€/mes \tag{2.7}$$

Por último, en la Expresión 2.8 se muestra el cálculo realizado para obtener el coste amortizado de otros recursos de trabajo generales como las sillas, mesas, conectores, pizarras, etc. En este caso se ha tenido en cuenta que la empresa pretende amortizar los recursos en un periodo de cinco años.

$$\text{Coste recursos de trabajo} = 1.500€/(5 \text{ años} * 12 \text{ meses}) = 25€/mes \tag{2.8}$$

En el Cuadro 2.2 se puede observar en detalle el coste total de todos los recursos.

Recursos	Coste mensual
Humanos	2.766€
Portátil de trabajo	19€ (coste amortizado)
Servidor de la Base de Datos	112€ (coste amortizado)
IntellIJ	42€ (coste amortizado)
Curso Flutter	3€ (coste amortizado)
Luz y agua	150€
Espacio de trabajo	8€ (alquiler mensual)
Otros recursos de trabajo (sillas, mesas, pizarras, etc)	25€ (coste amortizado)
Total	3.125€

Cuadro 2.2: Coste total del proyecto por mes.

En resumen, considerando el coste de 3.125€/mes por los recursos utilizados y que la estancia de prácticas ha sido de casi dos meses, el coste total del proyecto asciende a 6.250€ aproximadamente.

2.4. Riesgos y restricciones

Para asegurar el éxito del proyecto ha sido muy importante definir una buena gestión de riesgos. Para ello, en primer lugar se identificaron una serie de riesgos potenciales que podían influir en el desarrollo tanto del proyecto como de la aplicación. En el Cuadro 2.3 se muestran todos los riesgos identificados.

Identificador	Riesgo	Tipo de Riesgo
R01	Fecha de entrega inamovible	Proyecto
R02	Programador inexperto en algunas tecnologías	Proyecto
R03	Diseño e implementación erróneo	Aplicación
R04	Interfaz gráfica no usable para el paciente	Aplicación
R05	Cambios en las historias de usuario	Proyecto
R06	Nuevas historias de usuario	Proyecto

Cuadro 2.3: Identificación de los riesgos.

Una vez identificados los riesgos, en los Cuadros 2.4, 2.5, 2.6, 2.7, 2.8 y 2.9 se describen con mayor detalle cada uno de ellos. Concretamente, se muestra su magnitud y nivel de impacto, así como los planes de prevención y contingencia que se establecieron.

Riesgo	R01. Fecha de entrega inamovible.
Descripción	La fecha de entrega está fijada desde el inicio del proyecto.
Magnitud	Alta.
Impacto	La planificación debe ajustarse a la fecha de entrega.
Plan de prevención	Realizar el trabajo de planificación con un margen de ajuste del 10 %.
Plan de contingencia	Aumentar las horas de trabajo del programador de forma excepcional para solventar los desajustes.

Cuadro 2.4: Descripción y análisis del riesgo R01.

Riesgo	R02. Programador inexperto en algunas tecnologías
Descripción	El programador no tiene conocimientos de la tecnología Flutter necesaria para desarrollar la aplicación móvil.
Magnitud	Alta.
Impacto	El programador requiere de más tiempo para desarrollar las historias de usuario que hacen referencia a la aplicación móvil.
Plan de prevención	Realizar un curso web sobre la tecnología en cuestión proporcionado por la empresa.
Plan de contingencia	Preguntar al supervisor por un experto en Flutter para solucionar las posibles dudas.

Cuadro 2.5: Descripción y análisis del riesgo R02.

Riesgo	R03. Diseño e implementación erróneo
Descripción	Diseño o implementación de una historia de usuario de manera errónea.
Magnitud	Depende de la aparición. Si sucede al principio del proyecto, la magnitud será baja. Sin embargo, si el riesgo aparece en la parte final del proyecto la magnitud será alta.
Impacto	Puede suponer realizar una cantidad de cambios muy grande. Además, dependiendo de la historia de usuario, puede significar el fracaso del proyecto.
Plan de prevención	Definir las historias de usuario, previamente acordadas con el cliente. Además, realizar reuniones periódicas con el cliente para recibir su retroalimentación.
Plan de contingencia	Valorar el coste que supone no incluir o modificar la funcionalidad requerida en la historia de usuario.

Cuadro 2.6: Descripción y análisis del riesgo R03.

Riesgo	R04. Interfaz gráfica no usable para el paciente
Descripción	La interfaz gráfica no es intuitiva o usable para el paciente.
Magnitud	Depende de la aparición. Si sucede al realizar los prototipos, la magnitud será baja. Sin embargo, si el riesgo aparece en la parte final del proyecto la magnitud será alta.
Impacto	Puede suponer realizar una gran cantidad de cambios en la interfaz gráfica si se detecta al final del desarrollo.
Plan de prevención	Presentación de prototipos al cliente. Además, ofrecer una versión de prueba de forma periódica al cliente para recibir su retroalimentación.
Plan de contingencia	Dependiendo de la aparición del riesgo. Si sucede en la fase de prototipado se generará una nueva versión. Por el contrario, si el riesgo aparece al final del proyecto se realizarán los cambios mínimos posibles para lograr la satisfacción del cliente.

Cuadro 2.7: Descripción y análisis del riesgo R04.

Riesgo	R05. Cambios en las historias de usuario
Descripción	Se realizan cambios en las historias de usuario en mitad de un sprint.
Magnitud	Dependiendo de los cambios puede ser media o alta.
Impacto	Según el tipo de cambio puede suponer no finalizar la historia de usuario en cuestión. Además puede suponer un retraso en el resto de tareas del sprint.
Plan de prevención	Siguiendo la metodología Scrum, realizar reuniones diarias para solucionar dudas y estar constantemente informado de los posibles cambios.
Plan de contingencia	Si no se finaliza la historia de usuario en el sprint en el que se manifiesta el riesgo, se moverá al siguiente sprint. En caso de ser el último sprint, los cambios no se aplicarán.

Cuadro 2.8: Descripción y análisis del riesgo R05.

Riesgo	R06. Nuevas historias de usuario
Descripción	Se añaden nuevas historias de usuario a la pila del producto en mitad del desarrollo.
Magnitud	Alta.
Impacto	Si la estimación de la historia de usuario es muy alta, la implementación de la misma puede repercutir en el desarrollo de otras historias de usuario.
Plan de prevención	Realizar reuniones diarias y de planificación para poder mantener una mejor organización del trabajo y estar preparado para alguna posible nueva historia de usuario.
Plan de contingencia	Si la historia de usuario supone un conflicto con otra historia de usuario, se realizará la de mayor prioridad.

Cuadro 2.9: Descripción y análisis del riesgo R06.

Por último, en el Cuadro 2.10 se muestra la matriz de riesgos generada para la ayuda al desarrollo del proyecto. Esta matriz ha sido utilizada como guía en las reuniones de planificación del sprint. Además del identificador y el nombre del riesgo, también aparece la probabilidad de ocurrencia (PO), la evaluación del riesgo (ER) cuyo valor es el producto del nivel de impacto (I) y la probabilidad de ocurrencia, las acciones de prevención (AP) y las acciones de contención (AC).

Id.	Nombre	PO	I	ER	AP	AC
R01	Fecha de entrega inamovible	90	5	4,5	Margen de ajuste del 10 %	Horas extra
R02	Programador inexperto en algunas tecnologías	80	4	3,2	Formación previa	Contacto con experto
R03	Diseño e implementación erróneo	40	4	1,6	Comunicación contínua con el cliente	Valorar costes y reajustar
R04	Interfaz gráfica no usable para el cliente	40	4	1,6	Prototipos	Cambios mínimos con la colaboración del cliente
R05	Cambios en las historias de usuario	90	4	3,6	Reuniones diarias	Mover la historia al siguiente sprint
R06	Nuevas historias de usuario	40	4	1,6	Reuniones diarias y de planificación	Ordenar por prioridad

Cuadro 2.10: Matriz de riesgos del subsistema desarrollado.

En cuanto a las restricciones, la principal restricción ha sido la temporal, ya que este proyecto forma parte de una estancia en prácticas de una asignatura del grado y tiene un límite de 300 horas de trabajo. Dado que la fecha inicial de las prácticas fue el 1 de febrero y que se acordó que la jornada laboral fuera de 6 horas, el día 11 de abril supuso el fin de la estancia y por lo tanto, el fin del proyecto. Por otra parte, ha existido una restricción tecnológica debido a que la empresa exigió que el desarrollo de la aplicación móvil se desarrollara en Flutter.

2.5. Seguimiento del proyecto

En esta sección se detalla el seguimiento del proyecto durante los cinco sprints realizados.

2.5.1. Sprint 1

El principal objetivo de este sprint ha sido realizar el curso de introducción a Flutter, ya que era necesario para poder desarrollar la aplicación móvil. Además, también se han fijado como objetivo la generación del proyecto del servidor en Quarkus y de la aplicación móvil en Flutter. Como se puede observar en las Figuras 2.3 y 2.4, al finalizar el primer sprint del proyecto se han completado cuatro de las seis tareas incluidas en el sprint. La razón principal ha sido que una de las tareas completadas, concretamente la del curso de introducción a Flutter, ha necesitado

prácticamente de la totalidad del tiempo para ser realizada. En paralelo se han conseguido realizar tres tareas más. Las otras dos tareas se han mantenido en progreso y se han movido al siguiente sprint.

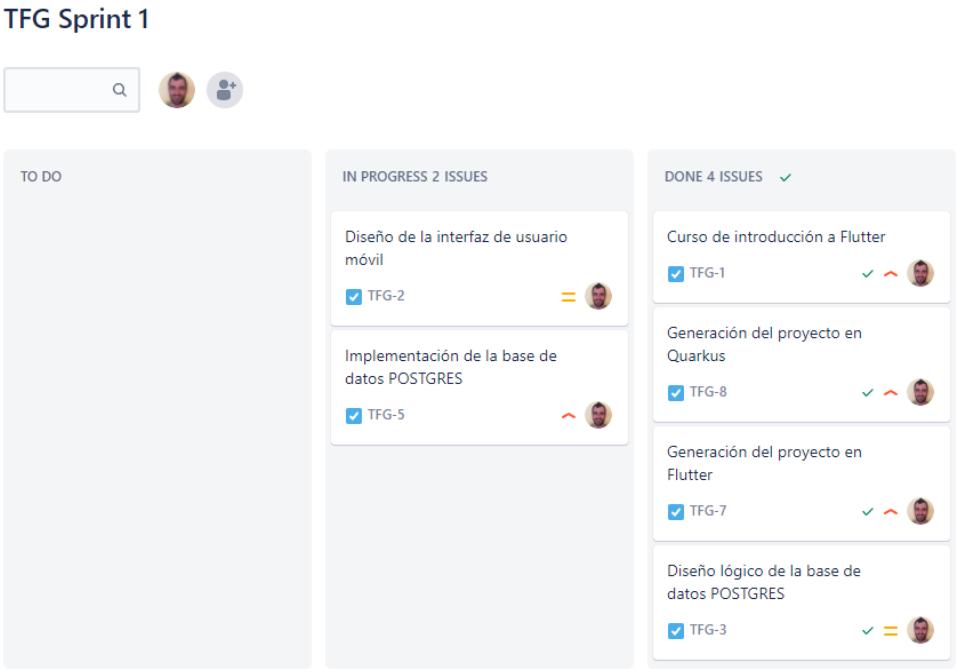

Figura 2.3: Tablero Kanban al finalizar el primer sprint.

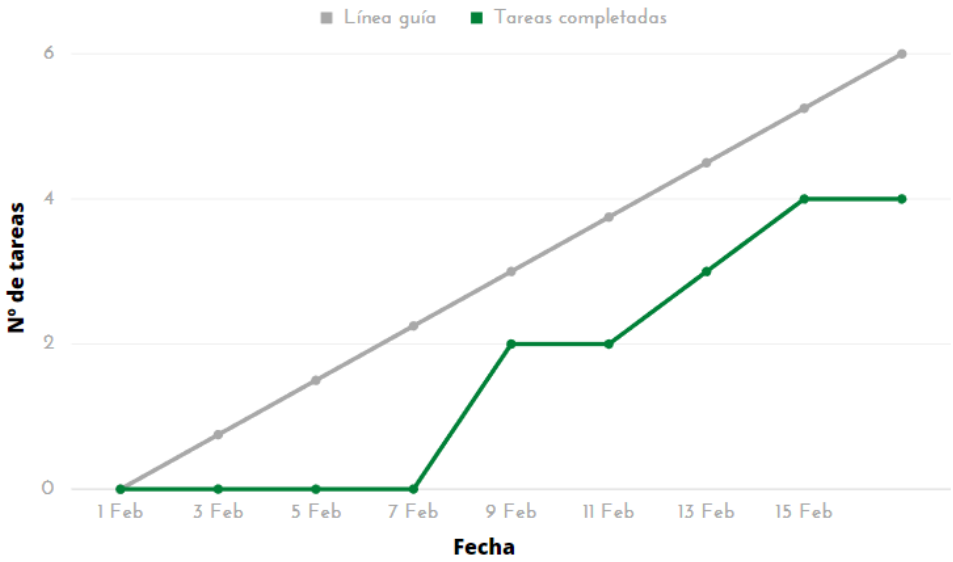

Figura 2.4: Informe sobre el progreso del primer sprint.

2.5.2. Sprint 2

La pila del producto al inicio del segundo sprint se puede ver en la Figura 2.5.

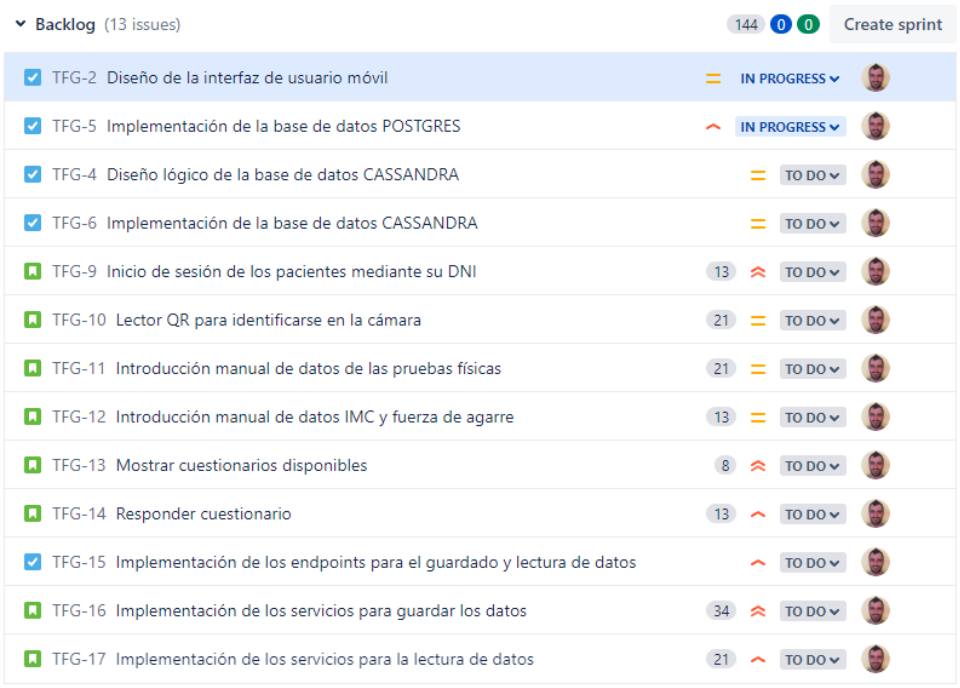

Figura 2.5: Pila del producto al inicio del segundo sprint.

Para el segundo sprint se ha decidido añadir, además de las dos tareas inacabadas en el primer sprint, las tareas restantes referentes al diseño e implementación de la base de datos. Además, se ha añadido la historia de usuario que hace referencia al inicio de sesión de los pacientes. El principal objetivo de este sprint ha sido preparar el sistema de gestión de base de datos necesario para la persistencia de los datos. Como se puede observar en la Figura 2.6 esta historia de usuario se ha dividido en siete subtareas.

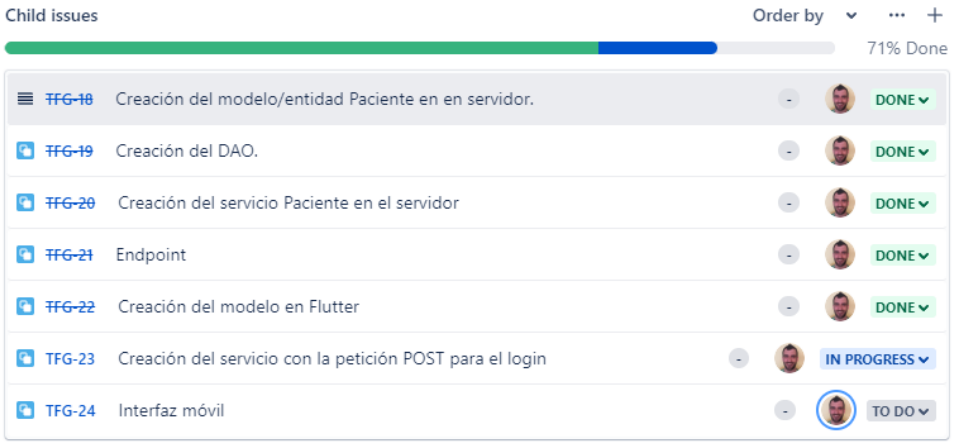

Figura 2.6: Subtareas añadidas a la historia de usuario de inicio de sesión del paciente.

La decisión de crear estas subtareas dentro de la historia de usuario de inicio de sesión en lugar de crearlas como tareas generales y añadirlas a la pila del producto ha sido una decisión consensuada entre el autor del documento y su supervisor. La razón principal ha sido

que se quiere alterar lo mínimo posible el sprint durante la realización del mismo. Así pues, el modelado de la entidad que representa al paciente, su objeto de acceso a datos (DAO), servicio y *endpoint* han sido añadidos como una subtarea del inicio de sesión del paciente. Esta decisión ha aumentado considerablemente la estimación de la historia de usuario y como se puede observar en la Figura 2.7, no ha podido ser completada al finalizar el sprint. Además, en la Figura 2.8 se aprecia el informe sobre el progreso del segundo sprint.

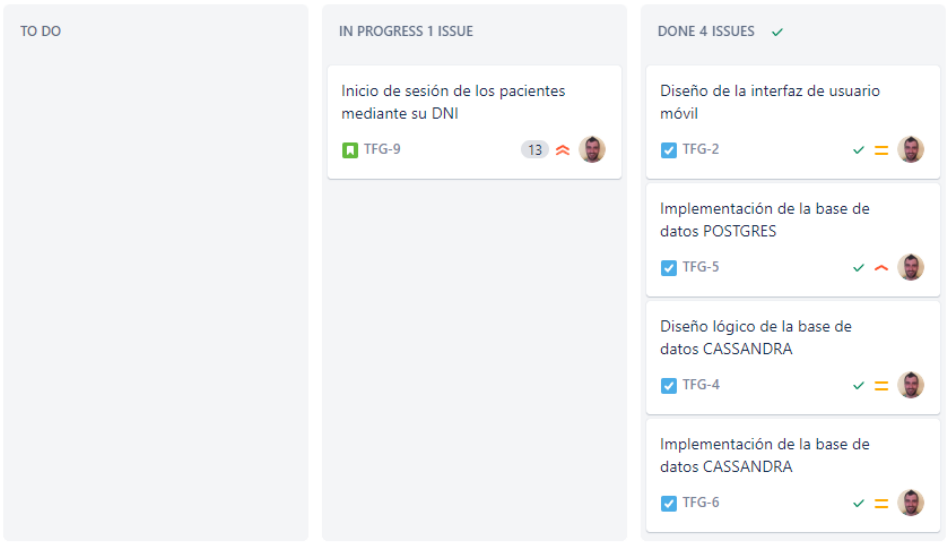

Figura 2.7: Tablero Kanban al finalizar el segundo sprint.

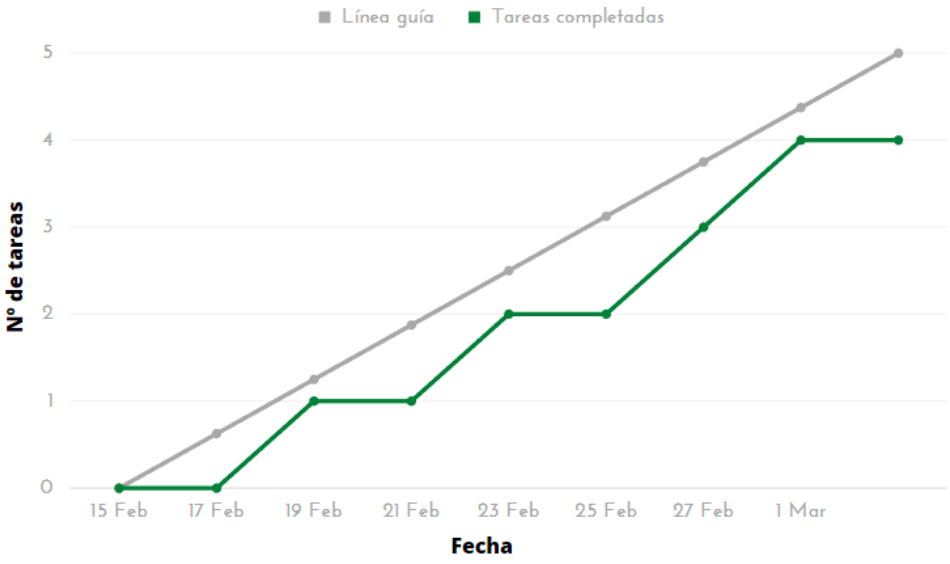

Figura 2.8: Informe sobre el progreso del segundo sprint.

2.5.3. Sprint 3

En este sprint se ha fijado el objetivo de intentar implementar todos los modelos, servicios y rutas necesarias para realizar el tratamiento de datos en el servidor. Las Figuras 2.9 y 2.10 muestran el estado de la pila del producto al inicio del tercer sprint y el tablero Kanban con las tareas e historias de usuario seleccionadas en ese sprint respectivamente.

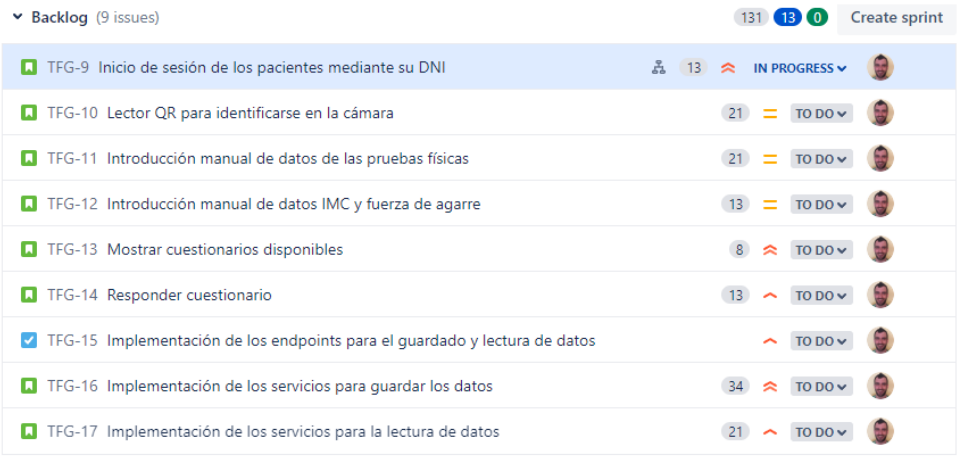

Figura 2.9: Pila del producto al inicio del tercer sprint.

Como se puede advertir en la figura 2.10, entre las tareas seleccionadas para el tercer sprint se encuentran las de la implementación de los servicios tanto para la lectura de datos como para su persistencia.

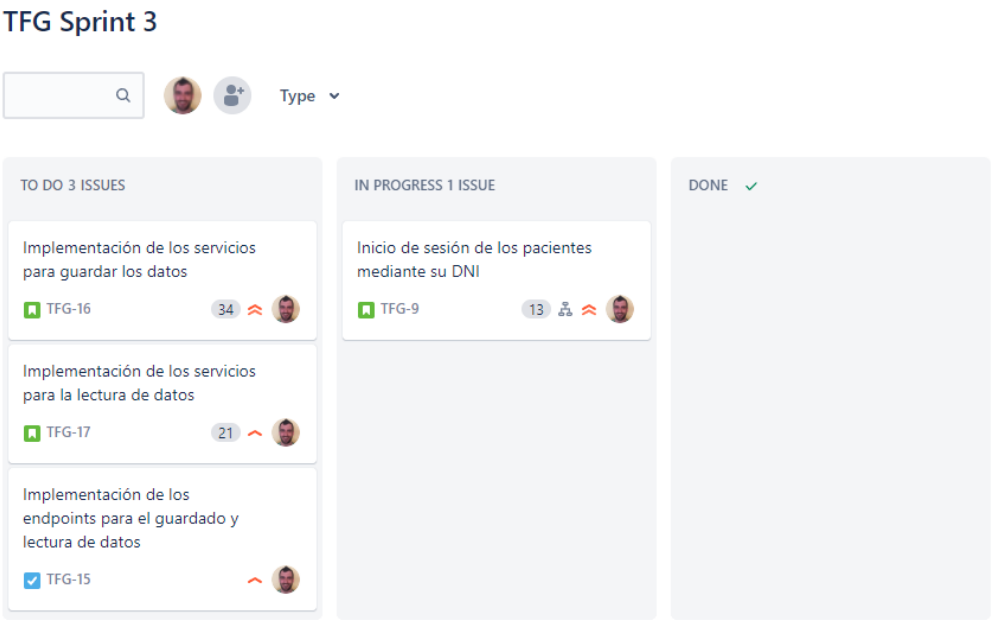

Figura 2.10: Tablero Kanban al inicio del tercer sprint.

En este punto, el autor de este documento vio la necesidad de crear de nuevo algunas subtareas para separar la creación de todas las entidades necesarias para llevar a cabo el proyecto. En esta ocasión, el supervisor aconsejó no crear varias subtareas sino una general que fue añadida a la pila del producto y al sprint que estaba en marcha. Cabe destacar que tras haber realizado la implementación del modelo del paciente en el segundo sprint, la implementación del resto de los modelos ha sido un proceso bastante más rápido. Sin embargo, el modelo de datos necesario para guardar las entradas manuales de los pacientes se ha modificado en dos ocasiones durante el desarrollo del sprint. La primera vez ha sido al inicio del tercer sprint, donde el cliente ha decidido añadir la posibilidad de registrar la altura del paciente. Este cambio ha implicado cambiar el modelo inicial que se había desarrollado. La segunda vez que se ha modificado el modelo ha sido al final del sprint, donde de nuevo se ha decidido añadir un nuevo tipo de dato. Estos cambios han modificado la estimación del trabajo del sprint y como se puede ver en la Figura 2.11, al finalizar el tercer sprint ha quedado una tarea sin realizar referente a las rutas para la lectura de datos.

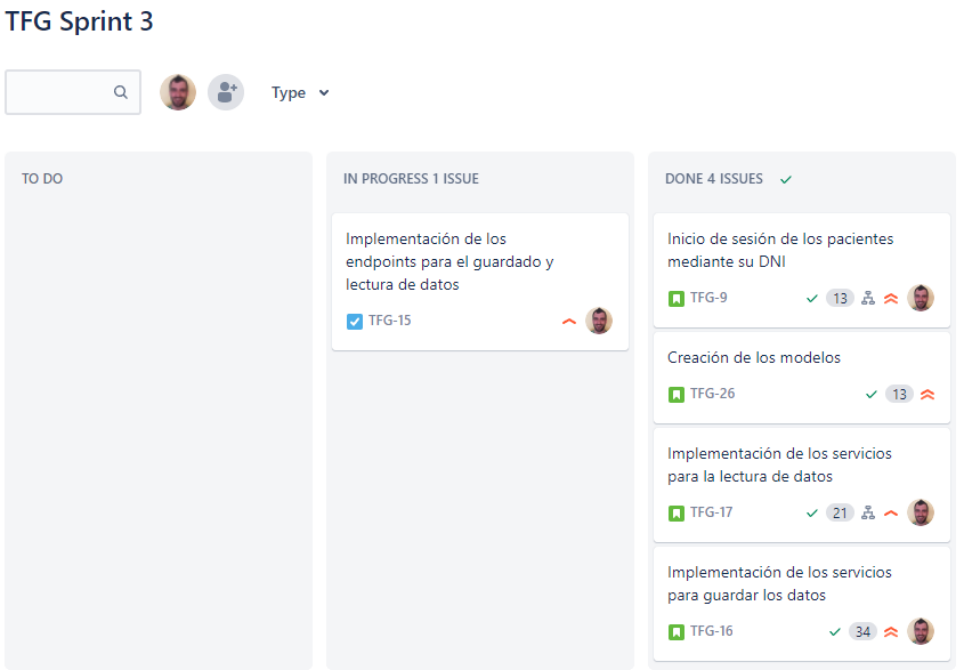

Figura 2.11: Tablero Kanban al finalizar el tercer sprint.

Esto también se puede observar en la Figura 2.12 donde se muestra el progreso del trabajo realizado en el tercer sprint.

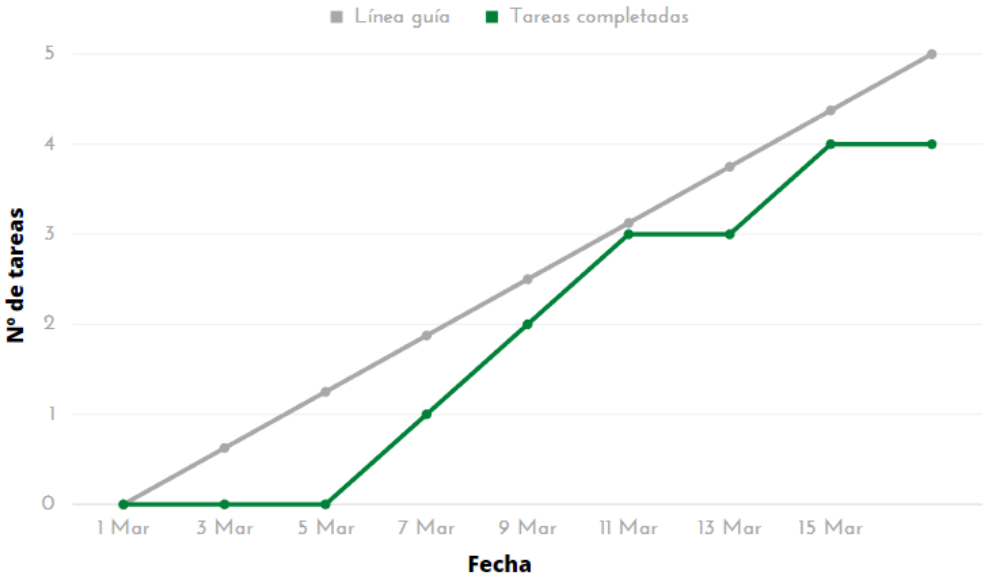

Figura 2.12: Informe sobre el progreso del tercer sprint.

2.5.4. Sprint 4

Tras la finalización del tercer sprint se ha terminado con casi toda la implementación en la parte del servidor. Así pues, en este sprint se ha puesto como objetivo principal, además de terminar la tarea referente al servidor, implementar las interfaces de la aplicación móvil relativas a la entrada de datos. Como se aprecia en la Figura 2.13, una única tarea pertenece a la parte del servidor mientras que el resto forman parte de la aplicación móvil.

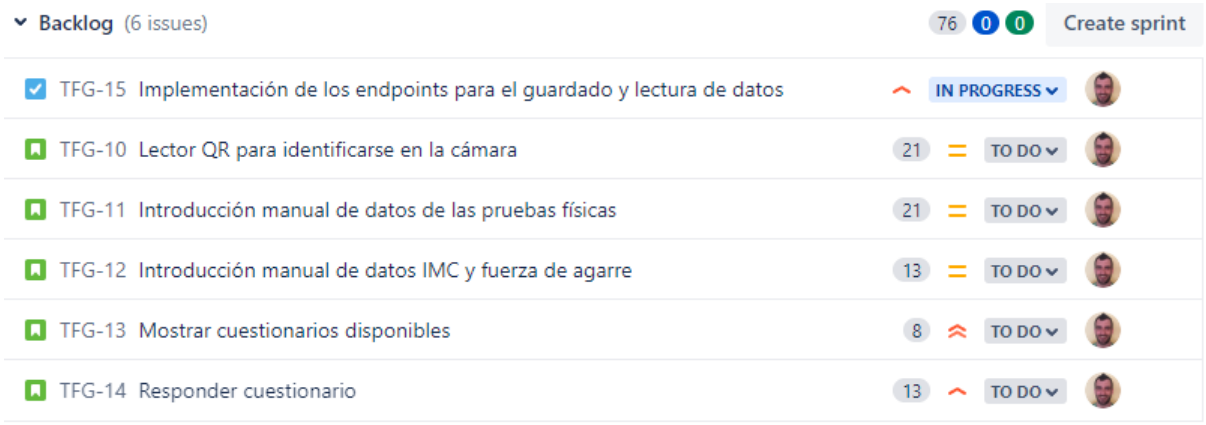

Figura 2.13: Pila del producto al inicio del cuarto sprint.

Para el cuarto sprint se han seleccionado todas las tareas excepto las que hacen referencia a la creación de los cuestionarios. En este sprint, se han conseguido finalizar todas las tareas a tiempo. La realización del curso en Flutter en el primer sprint ha optimizado el tiempo empleado en implementar el diseño de la aplicación. En la Figura 2.14 se refleja el estado del

tablero Kanban al finalizar el cuarto sprint. Por otra parte, en la Figura 2.15 se muestra el progreso de trabajo del sprint finalizado.

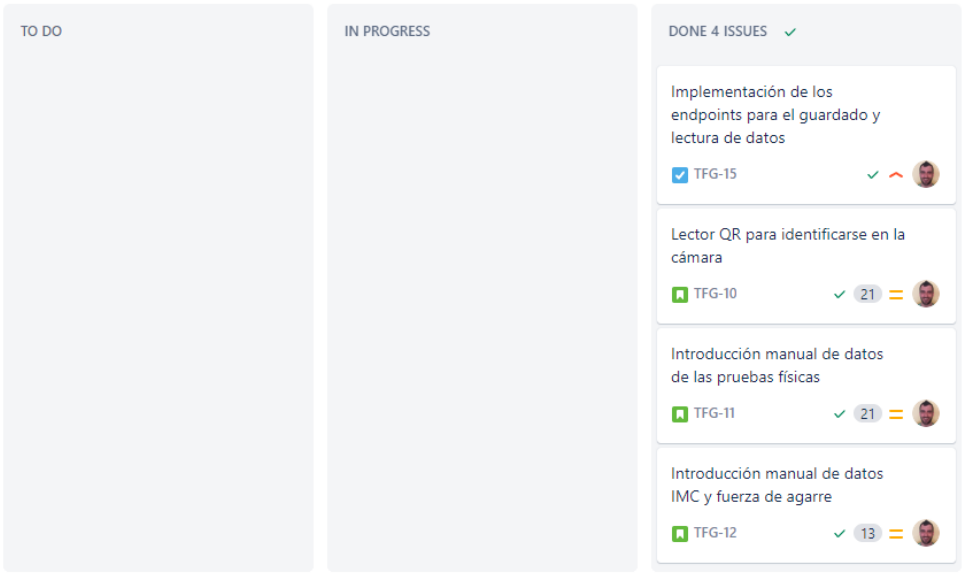

Figura 2.14: Tablero Kanban al finalizar el cuarto sprint.

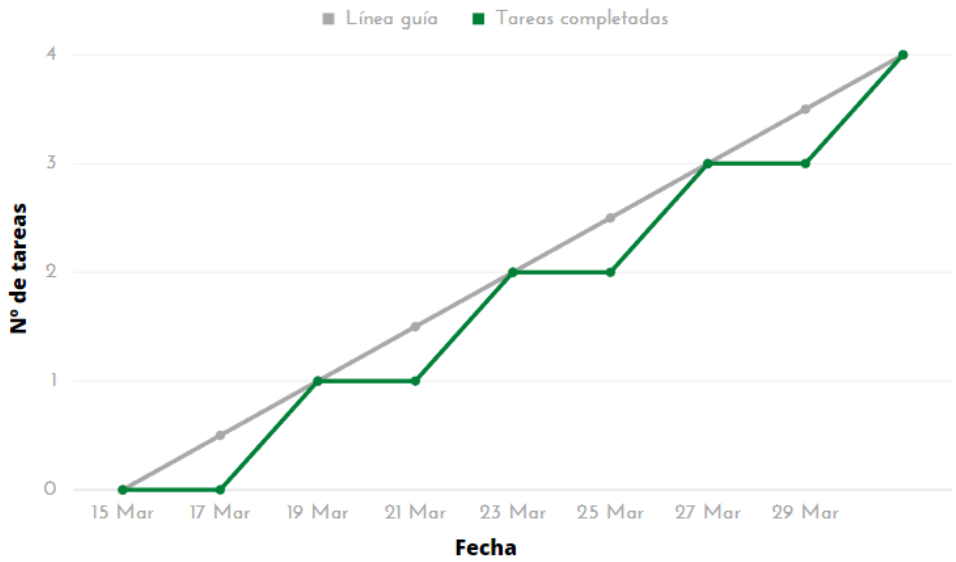

Figura 2.15: Informe sobre el progreso del cuarto sprint.

2.5.5. Sprint 5

Por último, en el quinto sprint se han seleccionado las tareas restantes que hacen referencia a la posibilidad de responder diferentes cuestionarios. La finalidad de este sprint ha sido principalmente acabar con las dos últimas tareas del proyecto que hacen referencia a la contestación de cuestionarios por parte del paciente. Como se muestra en la Figura 2.16, se ha decidido

añadir una tarea extra dedicada a la solución de problemas en la parte visual de la aplicación móvil. Esta tarea se ha añadido debido a que el resto han sido completadas unos días antes de terminar el sprint. En la Figura 2.17 se presenta el progreso del trabajo realizado en el quinto y último sprint.

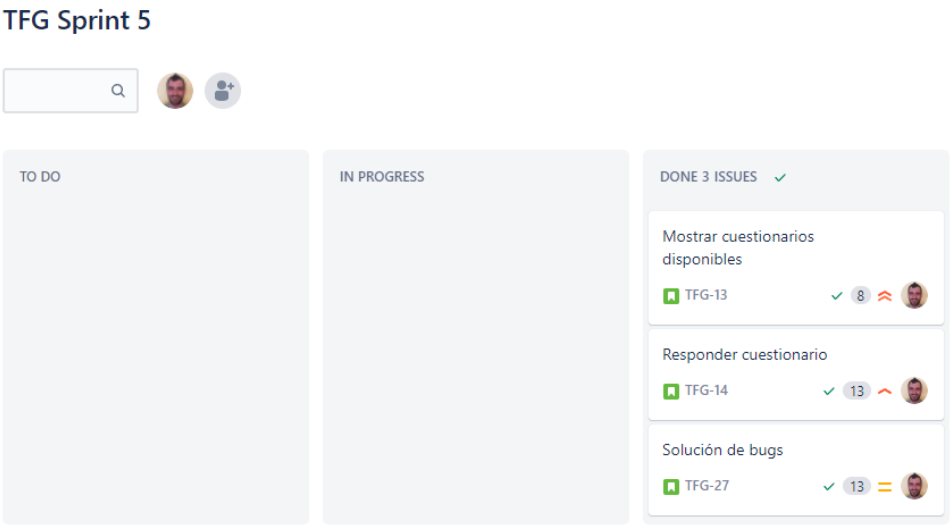

Figura 2.16: Tablero Kanban al finalizar el quinto sprint.

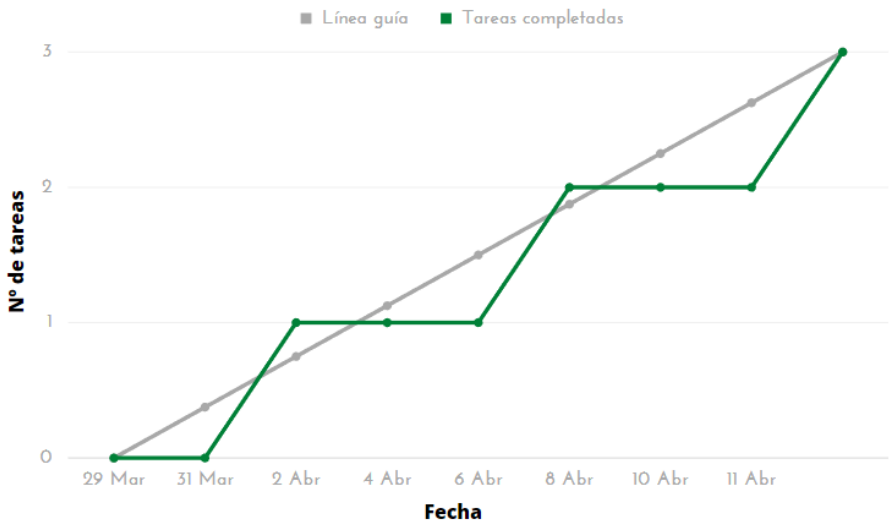

Figura 2.17: Informe sobre el progreso del quinto sprint.

En resumen, aunque no se han añadido nuevas historias de usuario a la pila del producto inicial, el desarrollo del mismo ha sufrido pequeños retrasos en los primeros sprints debido a una definición poco concreta de las tareas que han sido desarrolladas y a los cambios realizados en el modelo de datos en dos ocasiones, lo cual sí que ha modificado las tareas y subtareas de las historias de usuario incluidas en la pila del producto. Este retraso se ha conseguido estabilizar en el cuarto sprint, donde se han podido terminar todas las tareas asignadas. Esto ha permitido que en el quinto sprint se hayan terminado las tareas antes de tiempo y se han podido dedicar los últimos días del sprint a solucionar algunos errores visuales de la aplicación.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En esta sección se especifican el diagrama de casos de uso con las funcionalidades que puede realizar el paciente mediante la aplicación desarrollada, las historias de usuario implementadas durante la realización del proyecto, los requisitos de datos del segundo subsistema FTAS y el diagrama de clases diseñado para modelar el sistema.

3.1.1. Diagrama de casos de uso

Aunque la metodología ágil no requiere del diagrama de casos de uso, el autor de este documento ha creído conveniente su inclusión para ayudar al lector a visualizar la funcionalidad que se ha pretendido alcanzar durante el desarrollo del proyecto.

Como se puede observar en la Figura 3.1, el diagrama cubre todas las funcionalidades que se han definido en las historias de usuario. El CU01 está relacionado con la HU01 que permite el inicio de sesión en la aplicación por parte del paciente. El CU02 representa la HU02 en la cual el paciente debe ser identificado en la aplicación con la cámara del tercer subsistema FTAS a través de un código QR. El CU03 está relacionado con la HU05 donde el paciente debe poder visualizar y seleccionar un cuestionario. El CU04 comparte la funcionalidad representada en la HU06 donde el paciente puede responder a un cuestionario. El CU05 posee la funcionalidad de la HU03 sobre la inserción manual de datos relacionados con las pruebas físicas. Del mismo modo, el CU06 y la HU04 comparten la funcionalidad para la inserción de datos generales sobre la salud del paciente. Por último, el CU07 se corresponde a la HU08, donde el paciente puede ver la evolución del tiempo empleado en realizar las distintas pruebas físicas. La HU07 no está representada en el diagrama de casos de uso debido a que está implícita en todos los casos de uso existentes en el diagrama, ya que en todos se necesita una consulta o envío de datos al servidor.

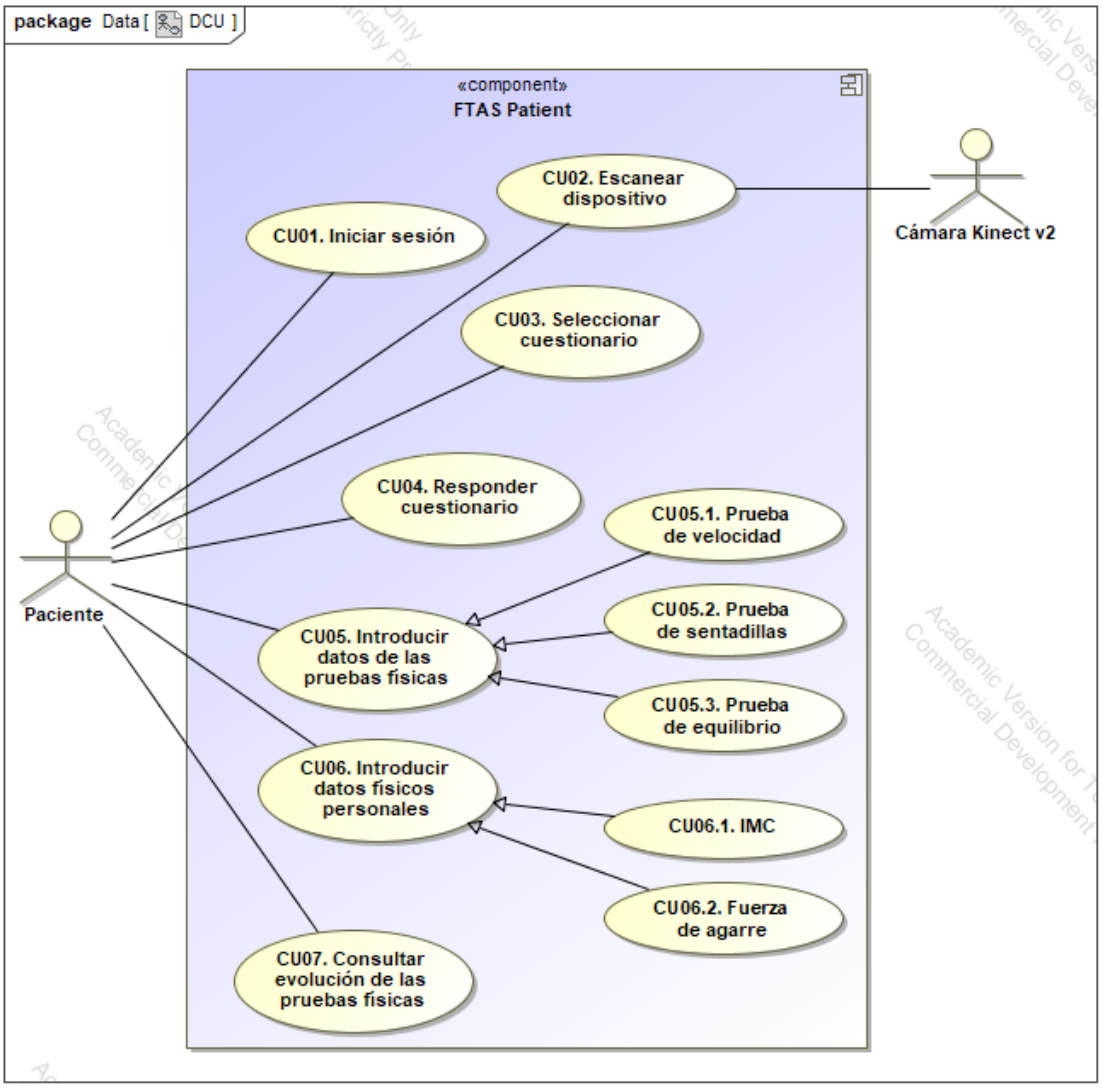

Figura 3.1: Diagrama de casos de uso del segundo subsistema FTAS.

3.1.2. Especificación de las historias de usuario

A continuación, se presenta la descripción y los escenarios de una de las historias de usuario definidas en el proyecto como un ejemplo para ilustrar al lector la forma en que se ha realizado la definición de las historias de usuario. En el Apéndice B se adjunta la definición del resto de historias de usuario. Los escenarios han sido definidos usando la estructura *given-when-then*.

HU06

Descripción: Como paciente quiero responder a los diferentes cuestionarios psicológicos disponibles en el sistema para aportar datos que ayuden a diagnosticar mi estado de fragilidad.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación y ha accedido al listado de cuestionarios.

AND El sistema está disponible y existe un cuestionario llamado 'Test de Pfiffer'.

When El paciente selecciona el cuestionario disponible.

Then La aplicación abre una nueva ventana con un mensaje de bienvenida al nuevo cuestionario.

Escenario 2

Given El paciente ha iniciado sesión en la aplicación y se encuentra en la ventana de bienvenida a un cuestionario.

When El paciente selecciona empezar el cuestionario.

Then La aplicación muestra la primera pregunta del cuestionario junto con las posibles respuestas.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación y se encuentra respondiendo a la primera pregunta de un cuestionario: '¿Qué día es hoy?'.

When El paciente escribe la respuesta 'Lunes tres de mayo' y decide continuar con la siguiente pregunta.

Then La aplicación guarda el valor introducido por el paciente y muestra la siguiente pregunta.

Escenario 4

Given El paciente ha iniciado sesión en la aplicación y se encuentra respondiendo a la segunda pregunta de un cuestionario: '¿Se ha encontrado bien durante la última semana?'

AND Se muestran dos respuestas posibles: 'Sí' y 'No'.

When El paciente selecciona la opción 'Sí' y decide continuar con la tercera pregunta.

Then La aplicación guarda el valor introducido por el paciente y muestra la siguiente pregunta.

Escenario 5

Given El paciente ha iniciado sesión en la aplicación y se encuentra respondiendo a la tercera pregunta de un cuestionario: 'Seleccione el tipo de dolor que ha sentido durante la última semana'. Se muestran cinco respuestas posibles: 'Cabeza', 'Brazos', 'Estómago', 'Cadera' y 'Piernas'.

When El paciente selecciona las opciones 'Cabeza' y 'Estómago', y decide continuar con la cuarta pregunta.

Then La aplicación guarda los valores introducidos por el paciente y muestra la siguiente pregunta.

Escenario 6

Given El paciente ha iniciado sesión en la aplicación y se encuentra respondiendo a la tercera pregunta de un cuestionario.

When El paciente selecciona la opción correspondiente para volver a la pregunta anterior para poder modificar la respuesta.

Then La aplicación muestra de nuevo la pregunta anterior.

Escenario 7

Given El paciente ha iniciado sesión en la aplicación y se encuentra respondiendo a la tercera pregunta de un cuestionario.

When El paciente decide salir del cuestionario sin finalizarlo y presiona sobre el botón correspondiente.

Then La aplicación vuelve a mostrar el listado de cuestionarios disponibles sin realizar el envío de datos al servidor.

Escenario 8

Given El paciente ha iniciado sesión en la aplicación ha respondido todas las preguntas de un cuestionario.

AND El sistema está disponible.

When El paciente selecciona la opción correspondiente para finalizar el cuestionario.

Then La aplicación envía los datos al servidor y muestra al paciente un mensaje indicando el éxito en el guardado de los datos.

Escenario 9

Given El paciente ha iniciado sesión en la aplicación ha respondido todas las preguntas de un cuestionario.

AND El sistema no está disponible.

When El paciente selecciona la opción correspondiente para finalizar el cuestionario.

Then La aplicación intenta enviar los datos al servidor y tras el error, muestra al paciente un mensaje indicando el fallo temporal del servidor.

3.1.3. Requisitos de datos

A continuación, el Cuadro 3.1 muestra todos los requisitos de datos que el sistema necesita para realizar las respectivas acciones en la aplicación.

Código	Nombre	Tipo
RD01	Datos del paciente	Entrada
RD02	Datos de la prueba de velocidad	Entrada, registro y salida
RD03	Datos de la prueba de sentadillas	Entrada, registro y salida
RD04	Datos de la prueba de equilibrio	Entrada, registro y salida
RD05	Datos físicos personales	Entrada y registro
RD06	Datos de los cuestionarios	Salida
RD07	Datos de las respuestas a los cuestionarios	Entrada y registro

Cuadro 3.1: Requisitos de datos del subsistema.

3.1.4. Diagrama de clases

En esta sección se muestra y se describe el modelo UML diseñado para el proyecto descrito en este documento. En la Figura 3.2 se presenta el modelo conceptual del subsistema representado mediante un diagrama de clases.

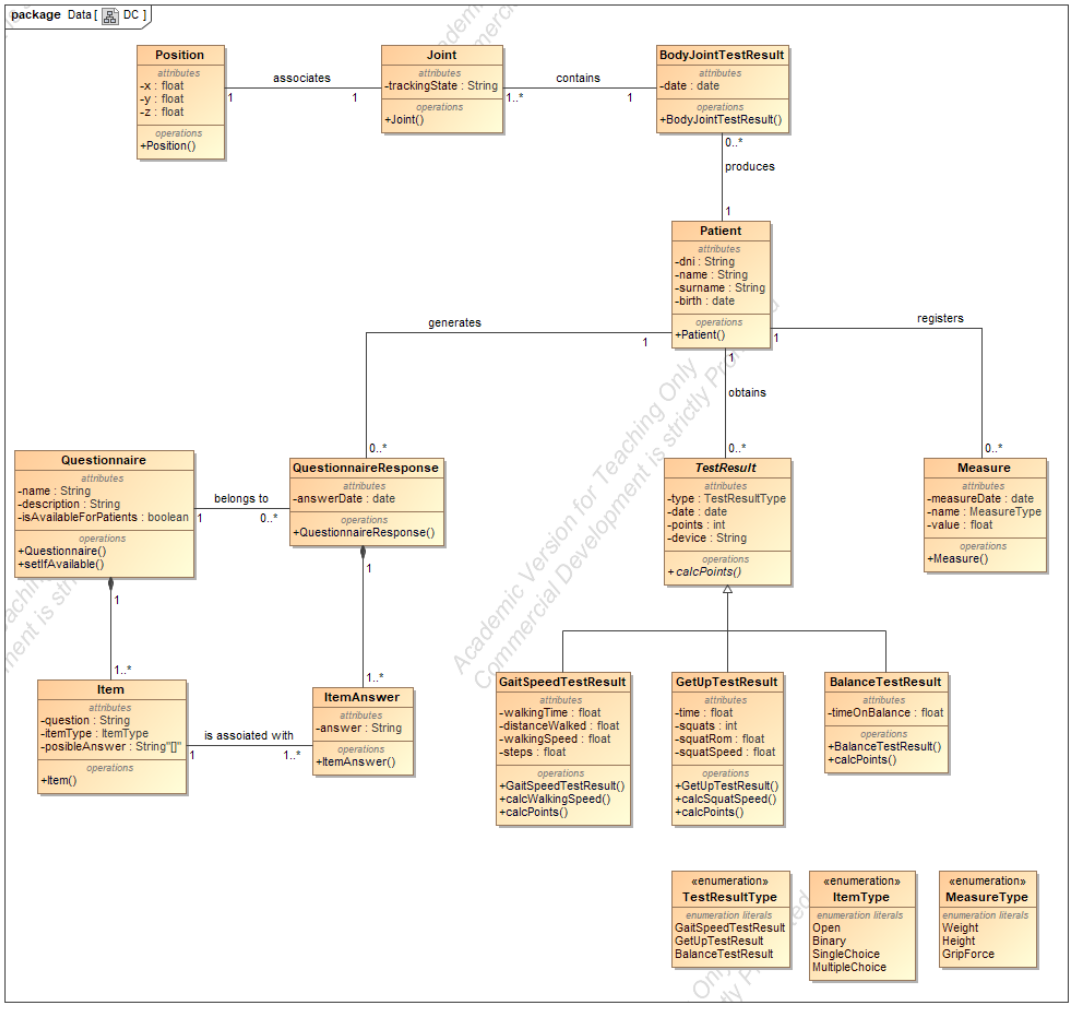

Figura 3.2: Diagrama de clases del segundo subsistema FTAS.

Como se puede observar, la clase principal del diagrama UML es *Patient*. Esta clase representa los datos personales del paciente y está conectada con el resto de clases.

En la parte superior de la figura, la clase *BodyJointTestResult* representa el movimiento realizado por el paciente en el espacio durante la realización de las pruebas físicas. Para ello, además de la fecha en la cual se realizó la prueba, se mantiene una serie de puntos en el espacio que simbolizan la posición de una de las articulaciones del paciente en el tiempo y que es representado por la clase *Joint*. Para controlar la posición exacta de un punto o articulación se utiliza la clase *Position*. Además, como se puede observar, la clase *Joint* posee un atributo que controla la fiabilidad de los valores almacenados. Este dato viene dado por la aplicación de la cámara Kinect.

En la derecha de la figura, la clase *Measure* representa la toma de una medida de carácter personal para el paciente y mantiene tanto el tipo de medida como su valor.

En el centro de la figura, la clase *TestResult* simboliza los datos obtenidos en las pruebas físicas, ya sea de manera automática mediante el tercer subsistema FTAS o de manera manual con la aplicación móvil descrita en este documento. Esta clase está definida como abstracta, de manera que su función es la de controlar los datos comunes a las tres pruebas físicas. Para gestionar los datos de cada una de las pruebas, se han definido las clases *GaitSpeedTestResult*, *GetUpTestResult* y *BalanceTestResult* que representan los datos de las pruebas de velocidad, sentadillas y equilibrio respectivamente. Si el lector se fija en estas tres clases puede observar que todas administran un valor que hace referencia al tiempo empleado en realizar la prueba. Este atributo no se ha incluido en la clase *TestResult* para no ser considerado un atributo común. La razón principal es que en cada prueba física, este atributo tiene una consideración diferente dependiendo del tipo de prueba.

En la parte izquierda de la figura se puede observar el diseño realizado para la administración de las respuestas de los cuestionarios por parte del paciente. Por una lado, la clase *Questionnaire* representa los datos básicos de un cuestionario y posee un atributo para controlar si está visible para los pacientes. Cada pregunta del cuestionario y sus posibles respuestas por defecto, si las tiene, están representadas por la clase *Item*. Esta clase también tiene en cuenta si el paciente puede seleccionar una o más respuestas a la pregunta. Por otro lado, la clase *QuestionnaireResponse* representa las respuestas introducidas por el paciente. Para ello, hace uso de la clase *ItemAnswer* que está relacionada con la clase *Item* para saber a qué pregunta está respondiendo el paciente.

Por último, también se han definido las siguientes tres enumeraciones para gestionar mejor los tipos de datos. La clase *TestResultType* representa los tres tipos de pruebas físicas, la clase *ItemType* simboliza el tipo de pregunta en un cuestionario y la clase *MeasureType* representa el tipo de medida de carácter personal que puede registrar un paciente.

3.2. Diseño de la arquitectura del sistema

En esta sección se analizan los componentes y las relaciones que forman parte del servidor y la aplicación móvil. En primer lugar se describe cada uno de ellos de forma separada y a

continuación se muestra la relación entre ambos. Además, también se presenta un esquema para entender mejor el lugar que ocupa el subsistema desarrollado dentro del sistema Frailty Tests Assistance System (FTAS). Por último, se presenta el diseño lógico utilizado para crear la base de datos.

3.2.1. Servidor

La arquitectura elegida para el desarrollo del programa servidor ha sido la clásica Modelo-Vista-Controlador (MVC). Sin embargo, se ha añadido una capa de servicios que se encarga de realizar la lógica de negocio separándola del controlador. En la Figura 3.3 se muestra un esquema que muestra todas las capas de la arquitectura.

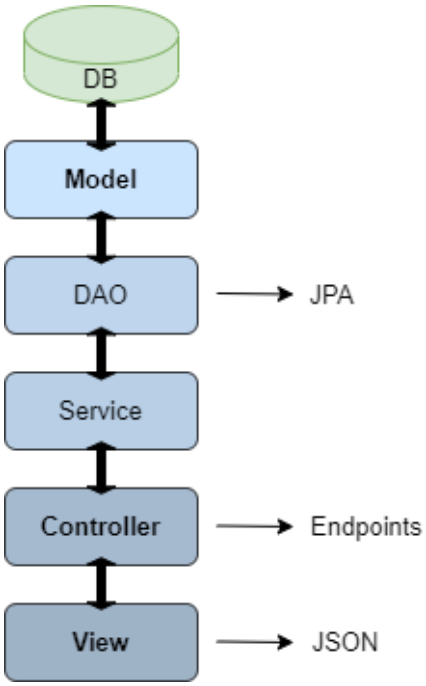

Figura 3.3: MVC de la aplicación servidor.

Cada una de las capas cumple una función distinta descrita a continuación. En primer lugar, la capa de la vista, que típicamente representa un archivo .xml o .html utilizado para mostrar un contenido web, es en este caso una cadena en formato .json con la información adquirida de la capa anterior, el controlador. En segundo lugar, la capa del controlador está dedicada a gestionar las rutas para recibir y enviar información a través de la capa de la vista y de la capa de servicio. En tercer lugar y como se ha comentado anteriormente, la capa de servicio pretende ser una capa intermedia entre la capa del controlador y el modelo de datos. Su función es la de separar la lógica de negocio del resto de capas. Por último, la capa del modelo representa los datos que gestiona el sistema, mientras que la capa DAO es la encargada de realizar las operaciones CRUD (Create, Read, Update and Delete) en la base de datos. En este caso, la capa DAO ha sido implementada mediante el ORM (Object-Relational Mapping) de Java conocido como JPA (Java Persistence Mapping).

3.2.2. Aplicación móvil

Como se ha comentado en diversas ocasiones, la herramienta utilizada para el desarrollo de la aplicación móvil ha sido Flutter. Esta tecnología está basada en la arquitectura Modelo-Vista-Vista de Modelo (MVVM). El esquema de esta arquitectura se presenta en la Figura 3.4. El patrón MVVM es una variación del patrón MVC que se caracteriza principalmente por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de negocio.

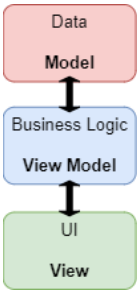

Figura 3.4: Esquema de la arquitectura MVVM.

La herramienta Flutter permite distintas maneras de mantener y actualizar el estado de la aplicación, es decir, controlar los cambios en los datos que se visualizan en la interfaz. De forma nativa, la herramienta proporciona una forma de mantener este estado. Sin embargo, tras realizar una búsqueda exhaustiva, el autor del documento llegó a la conclusión de que había mejores formas de realizar este mantenimiento usando librerías de terceros. La que se ha elegido para esta aplicación ha sido *provider*. Algunas de las ventajas de usar este software de terceros en lugar del nativo son las siguientes. En primer lugar, realiza una asignación de recursos más simple. En segundo lugar, tiene una mayor capacidad de escalabilidad debido a un mecanismo de escucha que observa los cambios realizados en la interfaz. Por último, dispone de la carga perezosa para proporcionar un mayor rendimiento en listas de datos muy largas. En la Figura 3.5 se muestra un ejemplo del flujo que sigue la aplicación cuando el paciente realiza una acción en la interfaz gráfica que conlleva una llamada al servidor.

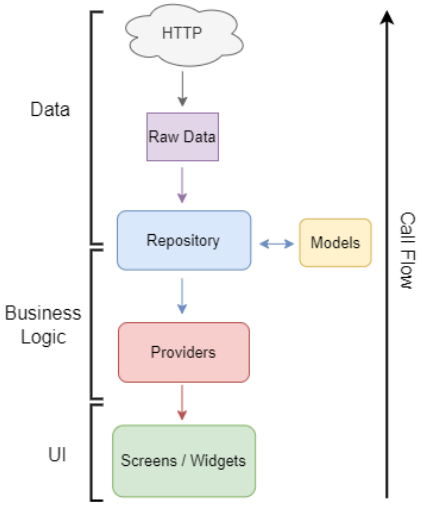

Figura 3.5: Flujo de ejemplo de una acción del paciente en la interfaz gráfica.

Cuando el paciente realiza una petición de envío o consulta de datos desde la interfaz, el *provider* asociado es notificado del cambio de estado que se produce y realiza las operaciones oportunas para satisfacer la acción del usuario. Para ello se ayuda de la capa de repositorio que transforma los datos de los modelos afectados y por último, se realiza la llamada HTTP. Cuando se recibe una respuesta del servidor, el proceso es el contrario. En este caso, los datos en bruto son procesados por la capa repositorio y se notifica a la capa *provider* para que actualice la interfaz.

3.2.3. Sistema FTAS

Para ayudar al lector a comprender mejor el sistema FTAS completo que pretende desarrollar la empresa Cuatroochenta, la Figura 3.6 presenta un esquema donde se pueden visualizar los tres subsistemas descritos en el Capítulo 1 y cómo se relacionan entre ellos.

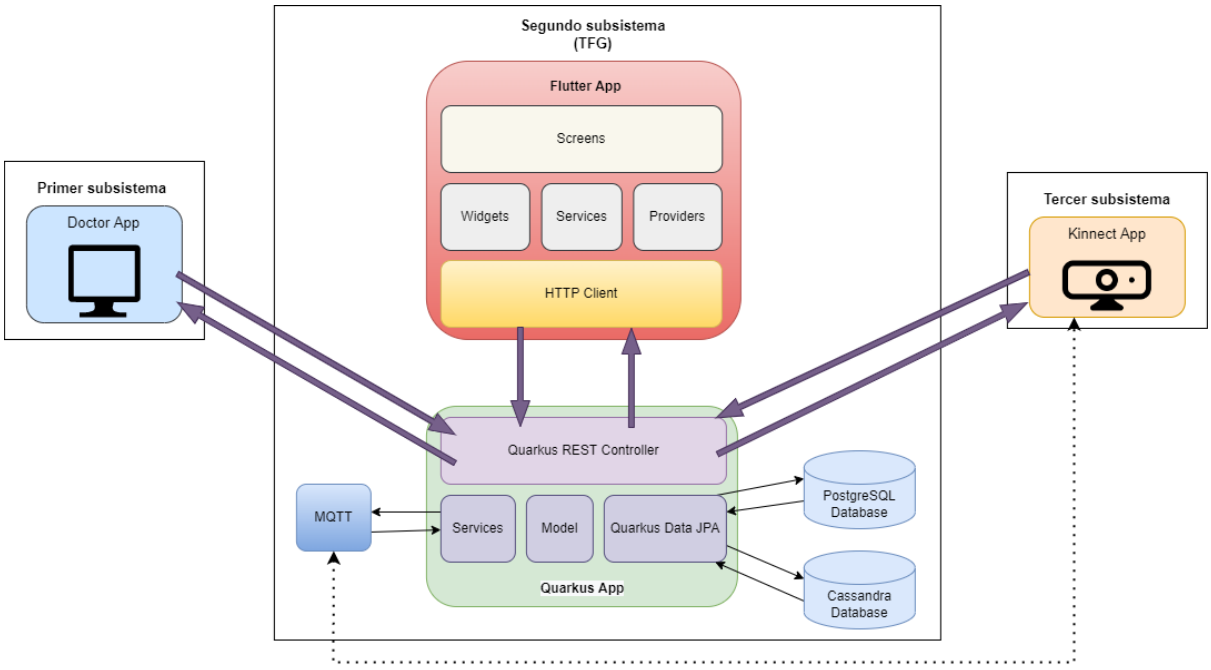

Figura 3.6: Esquema general del sistema FTAS.

La funcionalidad principal de cada uno de los subsistemas es la siguiente. Por una parte, el primer subsistema contiene la aplicación web diseñada para el profesional médico. Esta aplicación permite el registro de nuevos pacientes y realizar el seguimiento y evolución de las pruebas físicas de los pacientes, los datos de carácter físico introducidos de forma manual desde la aplicación de Flutter y las respuestas de los cuestionarios realizados por los pacientes. Todas estas operaciones se realizan mediante peticiones HTTP al servidor desarrollado en el segundo subsistema. Por otra parte, el segundo subsistema es el que trata este documento y está compuesto tanto por la aplicación móvil como por el programa servidor. Por último, el tercer subsistema contiene la aplicación de escritorio que controla la cámara Kinnect. Esta aplicación permite la identificación del paciente mediante un código QR que contiene el identificador de la cámara y que es usado para iniciar sesión en el dispositivo usando el protocolo MQTT. Cuando

el paciente ha iniciado sesión en el dispositivo, realiza como mínimo una de las pruebas físicas. Al finalizar cada una de las pruebas, se envían los datos recogidos al servidor desarrollado en el segundo subsistema mediante el protocolo HTTP.

3.2.4. Diseño de la base de datos

A continuación, se presenta el diseño lógico de la base de datos implementada en el gestor de base de datos PostgreSQL. Para realizar este diseño lógico se ha partido del diseño conceptual presentado en el diagrama de clases de la sección 3.1.4. El esquema completo de la base de datos se puede visualizar en la Figura 3.7.

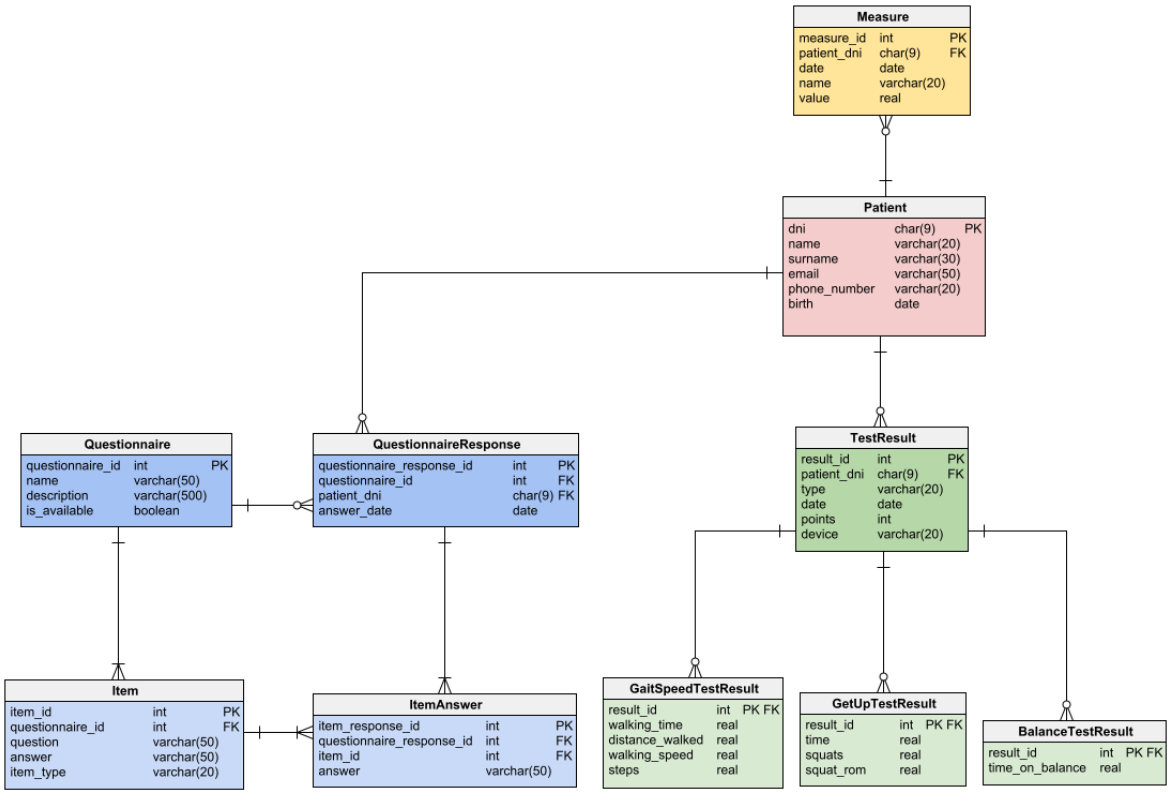

Figura 3.7: Diseño lógico de la base de datos.

Si se compara el diagrama de clases presentado en la sección anterior con la Figura 3.7, se puede observar que las clases *BodyJointTestResult*, *Joint* y *Position* no se muestran en el modelo de la base de datos PostgreSQL. La razón es que se espera que la cantidad de instancias de estas clases sea muy elevada, y PostgreSQL no es la mejor opción para este tipo de casos. Por ello, se ha elegido el gestor de base de datos Cassandra, ya que está preparado para gestionar una gran cantidad de datos. Por lo tanto, la base de datos Cassandra almacena las instancias de las clases *BodyJointTestResult*, *Joint* y *Position*. A continuación, se describe de manera más detallada el diseño lógico de la base de datos PostgreSQL.

Patient (dni, name, surname, birth)

- **Tipo de datos:**
  dni: string
  name: string
  surname: string
  birth: date

Measure (measure_id, patient_dni, date, name, value)

- patient_dni clave ajena a Patient
  Measure → Patient
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  measure_id: int
  patient_dni: string
  date: date
  name: string
  value: float

TestResult (result_id, patient_dni, type, date, points, device)

- patient_dni clave ajena a Patient
  TestResult → Patient
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  result_id: int
  patient_dni: string
  type: string (GaitSpeedTestResult, GetUpTestResult o BalanceTestResult)
  date: date
  points: int
  device: string

GaitSpeedTestResult (result_id, walking_time, distance_walked, walking_speed, steps)

- result_id clave ajena a TestResult
  GaitSpeedTestResult → TestResult
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  result_id: int
  walking_time: float
  distance_walked: float
  walking_speed: float
  steps: float

GetUpTestResult (result_id, time, squats, squat_rom)

- result_id clave ajena a TestResult
  GetUpTestResult → TestResult
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  result_id: int
  time: float
  squats: float
  squat_rom: float

BalanceTestResult (result_id, time_on_balance)

- result_id clave ajena a TestResult
  BalanceTestResult → TestResult
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  result_id: int
  time_on_balance: float

Questionnaire (questionnaire_id, name, description)

- **Tipo de datos:**
  questionnaire_id: int
  name: string
  description: string

Item (item_id, questionnaire_id, question, answers, is_multiple)

- questionnaire_id clave ajena a Questionnaire
  Item → Questionnaire
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  item_id: int
  questionnaire_id: int
  question: string
  answers: string
  is_multiple: boolean

QuestionnaireResponse (questionnaire_response_id, questionnaire_id, patient_dni, answer_date)

- questionnaire_id clave ajena a Questionnaire
  QuestionnaireResponse → Questionnaire
  Null: NO, Borrado: R, Modificación: P

- patient_dni clave ajena a Patient
  QuestionnaireResponse → Patient
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  questionnaire_response_id: int
  questionnaire_id: int
  patient_dni: string
  answer_date: date

ItemAnswer (item__answer_id, questionnaire_response_id, answers)

- questionnaire_response_id clave ajena a QuestionnaireResponse
  ItemAnswer → QuestionnaireResponse
  Null: NO, Borrado: R, Modificación: P

- **Tipo de datos:**
  item__answer_id: int
  questionnaire_response_id: int
  answers: string

3.3. Diseño de la interfaz

En esta sección se describen los criterios elegidos para diseñar la interfaz de usuario y se presentan los prototipos que el autor del documento realizó previamente al desarrollo, así como el resultado final después de la implementación.

Los usuarios finales de la aplicación desarrollada son pacientes con una edad avanzada. Eso significa que, en general, son personas que no tienden a utilizar el teléfono móvil porque les parece demasiado complicado, no entienden la información que se les muestra en pantalla y les resulta más complicado recordar la navegación del sitio o aplicación. Teniendo esto en cuenta, los tres criterios fundamentales que se han seguido para realizar el diseño de interfaces han sido los siguientes. Por una parte, se ha utilizado un tamaño de letra grande para obtener una mejor visibilidad en la información mostrada en la pantalla. Por otra parte, como se puede ver en la Figura 3.8 la navegación de la aplicación es lo más simple posible para ayudar a los pacientes a aprender y recordar de manera más sencilla toda la funcionalidad de la aplicación. Por último, se han intentado utilizar unos colores sencillos pero con claras diferencias entre sí para ayudar a diferenciar de forma clara entre las distintas opciones que aparecen en las ventanas.

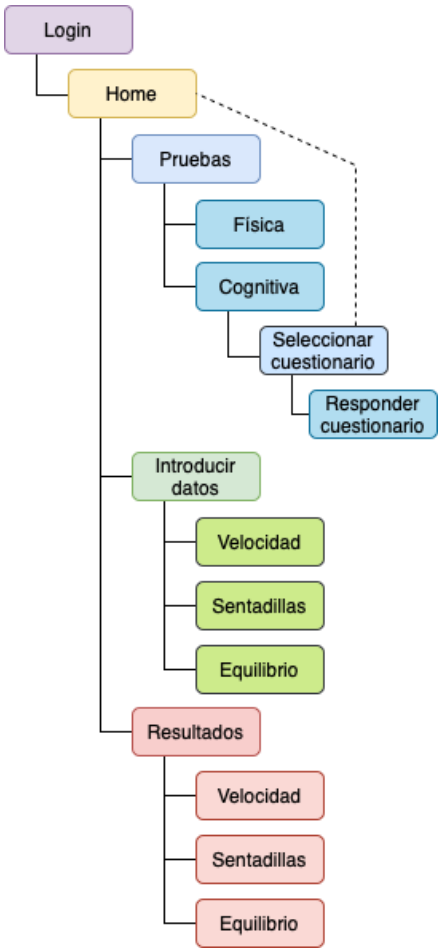

Figura 3.8: Mapa de navegación de la aplicación móvil desarrollada.

Para realizar el diseño de las interfaces se ha procurado seguir las reglas de oro estudiadas durante el desarrollo del grado. Se ha intentado mantener un alto grado de consistencia, con retroalimentación continua para el paciente informándole del estado de la aplicación. Además, se han incorporado atajos en algunas ventanas para evitar la saturación del usuario en la navegación. También se ha tenido en cuenta la posible voluntad del paciente de querer revertir alguna de las acciones ya realizadas en la aplicación. Finalmente, se ha intentado reducir la carga de memoria a corto plazo.

3.3.1. Hoja de estilos

Marca. El proyecto pretende ayudar a las personas mayores a prevenir o paliar el estado de fragilidad.

Tipografía. En la Figura 3.9 se muestra la tipografía utilizada en la aplicación desarrollada en Flutter.

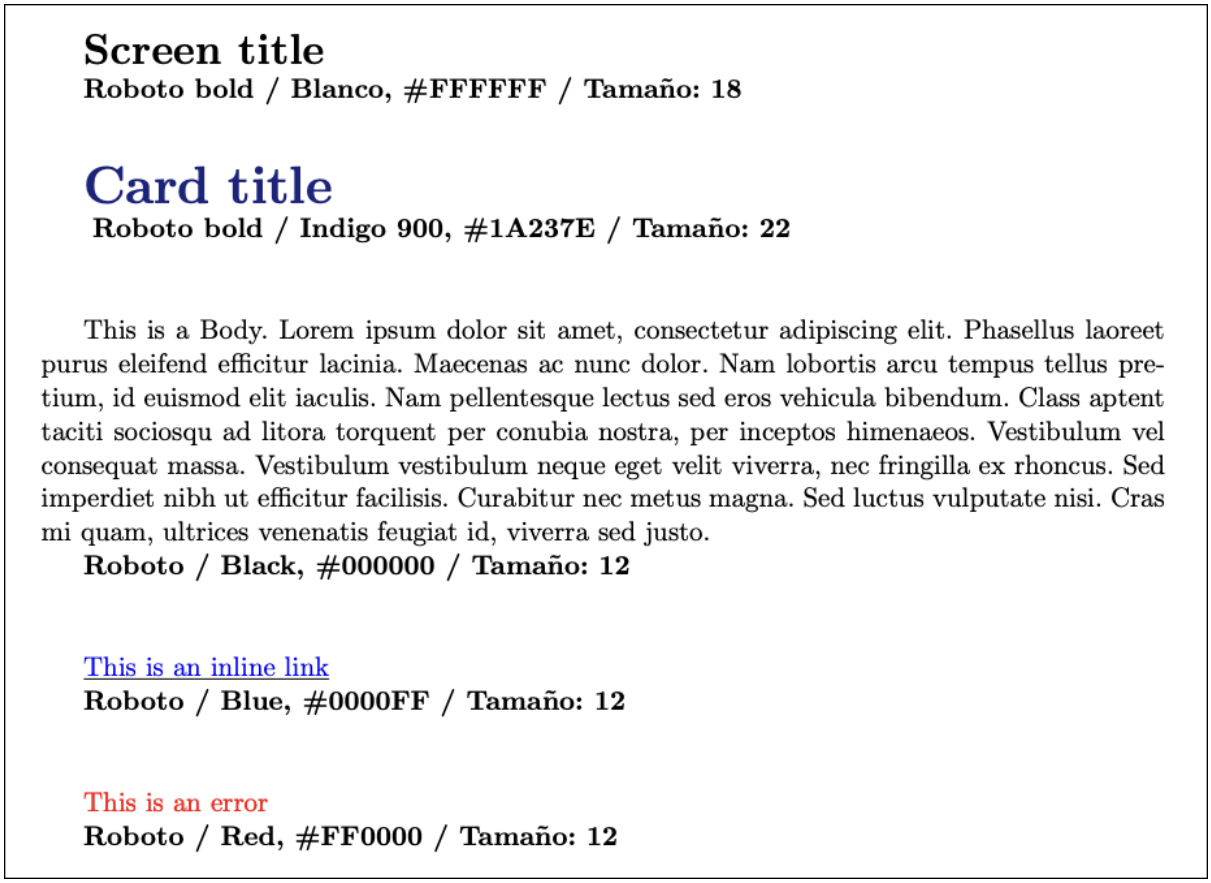

Figura 3.9: Tipografía utilizada en la aplicación móvil.

Paleta de colores. En la Figura 3.10 se pueden ver los colores principales utilizados en la interfaz móvil.

Figura 3.10: Paleta de colores de la aplicación móvil.

Voz. Se ha utilizado la 3ª persona formal del singular. Algunos ejemplos se pueden ver a continuación.

"Introduzca una medida"
"Inténtelo de nuevo en unos instantes"

Iconografía. Se utilizan iconos para mostrar información de guía al paciente en cada uno de los botones que requieran su interacción. Los iconos pertenecen a la librería Material de Google. Las Figuras 3.11, 3.12 y 3.13 muestran algunos iconos usados en el diseño final. Sin embargo, el color usado en la aplicación ha sido el blanco.

Figura 3.11: Icono para la prueba de velocidad.

Figura 3.12: Icono para la prueba de sentadillas.

Figura 3.13: Icono para la prueba de equilibrio.

Formas. Se especifica información detallada sobre los distintos errores y mensajes de la aplicación. Los mensajes destinados al paciente se muestran en una ventana emergente o alerta, especificando en cada caso los pasos a seguir. Se prioriza una retroalimentación completa para el usuario.

El estilo de los botones para toda la aplicación es el siguiente. Las esquinas están redondeadas con un valor de 10 píxeles y el color utilizado para mostrar su estado normal es índigo. Si el botón está desactivado se utiliza el color gris. Además, para dar una sensación de elevación se utilizan 5 píxeles por debajo del botón con el objetivo de dar la sensación de sombreado.

Botones. Los botones que se han usado han sido proporcionados por Flutter, que utiliza la librería Material [9]. Principalmente se caracterizan por su estilo minimalista y su curvada forma. Se han usado los colores establecidos en la paleta de colores y la forma especificada.

Espaciado. Los títulos de las ventanas tiene 5 píxeles de margen por la parte superior e inferior. Los márgenes derecho e izquierdo dependen de los elementos que existan en la ventana en cuestión. El título se adapta para quedar centrado y por lo tanto, el espaciado horizontal es dinámico.

Las tarjetas tienen un margen vertical de 30 píxeles para separar una tarjeta de otra. El contenido que hay en el interior de ellas tiene un acolchado o *padding* de 20 píxeles tanto vertical como horizontalmente.

Los componentes utilizados para la entrada de datos manual por parte del paciente poseen un *padding* de 5 píxeles en vertical y horizontalmente. De esta manera, se consigue una mejor visibilidad a la hora de comprobar los datos que se están insertando manualmente.

Todas las ventanas tienen un *padding* de 10 píxeles en todas las direcciones para intentar, en la medida de lo posible, dar una mayor sensación de amplitud.

3.3.2. Prototipos

A continuación, las Figuras 3.14 y 3.15 muestran los prototipos diseñados para el inicio de sesión del paciente y su menú principal.

Figura 3.14: Login.

Figura 3.15: Menú principal.

Seguidamente, en las Figuras 3.16, 3.17, 3.18 y 3.19 se presentan las distintas interfaces diseñadas para cumplir con la funcionalidad que hace referencia al escaneo de un dispositivo y su interacción.

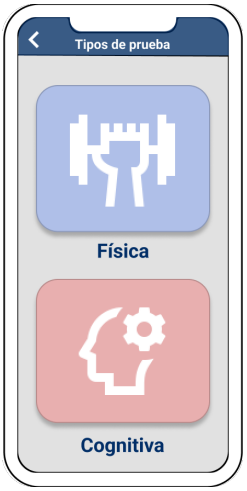

Figura 3.16: Tipos de prueba.

Figura 3.17: Escanear dispositivo.

Figura 3.18: Éxito al escanear.

Figura 3.19: Fallo al escanear.

De la misma forma, en las Figuras 3.20 y 3.21 se pueden observar los diseños realizados para facilitar al paciente la contestación de los cuestionarios cognitivos.

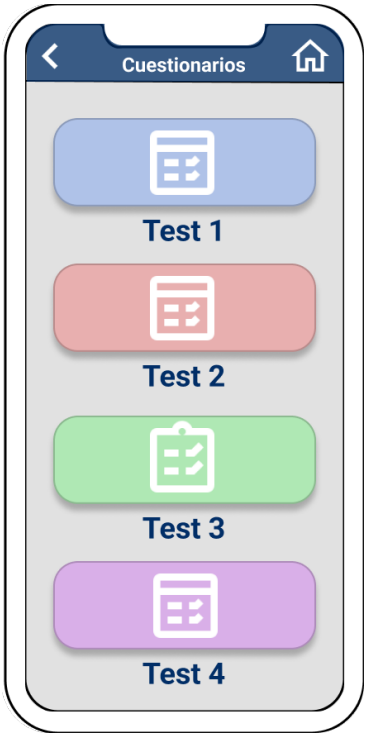

Figura 3.20: Lista de cuestionarios.

Figura 3.21: Respuesta a un cuestionario.

Por otra parte, en las Figuras 3.22, 3.23, 3.24 y 3.25 se presentan los diseños de algunas de las ventanas necesarias para que el paciente pueda introducir datos de manera manual.

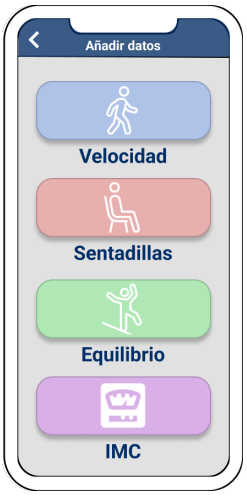

Figura 3.22: Introducir datos manualmente.

Figura 3.23: Introducir datos de la prueba de velocidad.

Figura 3.24: Introducir datos de la prueba de sentadillas.

Figura 3.25: Introducir datos IMC.

Por último, en la Figura 3.26 se muestra el prototipo diseñado para el listado de resultados, mientras que en la Figura 3.27 se presenta el prototipo diseñado para la presentación de los resultados obtenidos en las pruebas físicas.

Figura 3.26: Lista de pruebas con resultados disponibles.

Figura 3.27: Resultados prueba de velocidad.

Cabe mencionar que las interfaces mostradas forman parte del prototipo final aceptado para el desarrollo de la aplicación. En el Apéndice A se muestra otro diseño de prototipos que se realizó en la etapa de análisis. Aunque este diseño está pensado para una mayor eficiencia de uso y tiene prácticamente toda la funcionalidad de la aplicación en una única ventana, se decidió optar por el diseño presentado debido a que tiene un aspecto visual más amigable para las personas mayores. Aún así, el diseño alternativo no ha sido descartado por la empresa y se ha guardado como una posible opción de cambio cuando el sistema FTAS se implante en centros médicos.

3.3.3. Interfaz final

En este apartado se presentan las interfaces finales de la aplicación desarrollada en el mismo orden que en la sección anterior. Las imágenes están tomadas desde un iPhone SE. Las Figuras 3.28 y 3.29 muestran la interfaz implementada para las ventanas de inicio de sesión y el menú principal de la aplicación.

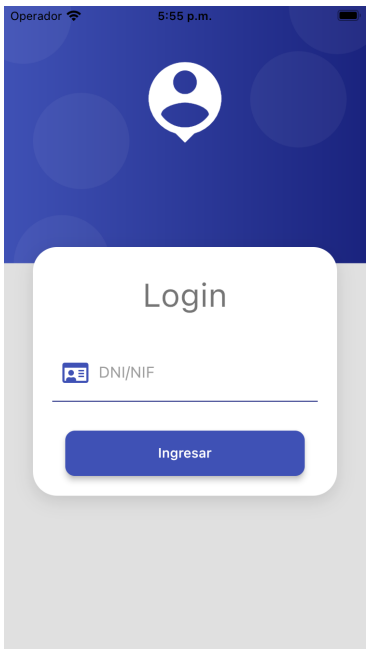

Figura 3.28: Login.

Figura 3.29: Menú principal.

En las Figuras 3.30 y 3.31 se muestran algunas de las interfaces relativas al escaneo de un dispositivo.

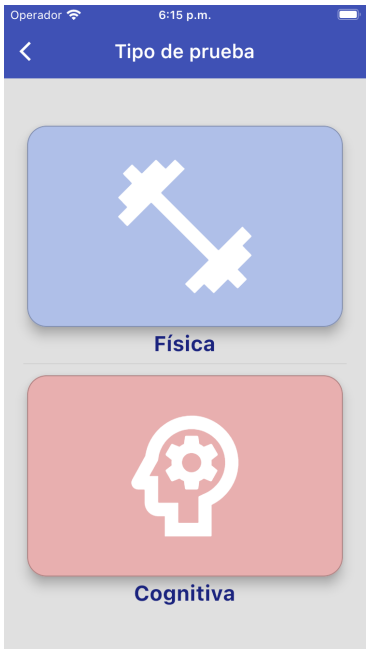

Figura 3.30: Tipos de pruebas.

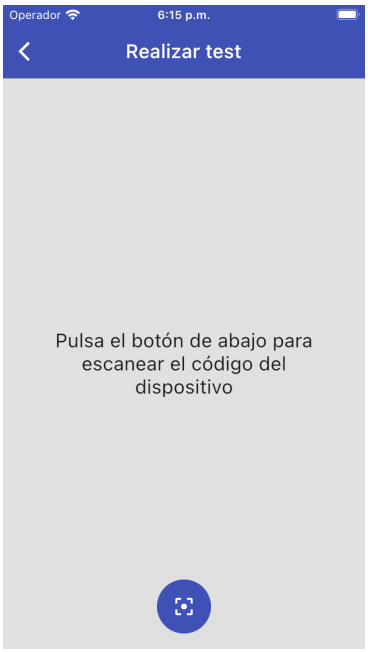

Figura 3.31: Escanear un dispositivo.

Respecto a la visualización y respuesta de cuestionarios, las Figuras 3.32, 3.33, 3.34 y 3.35 muestran el flujo que sigue el paciente cuando quiere responder a un cuestionario en particular.

Figura 3.32: Cuestionarios disponibles.

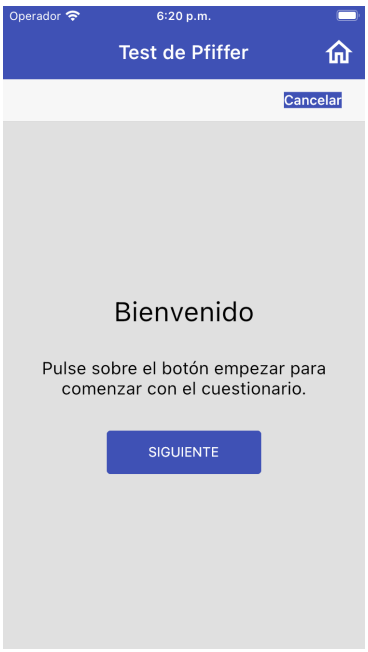

Figura 3.33: Ventana de introducción al cuestionario.

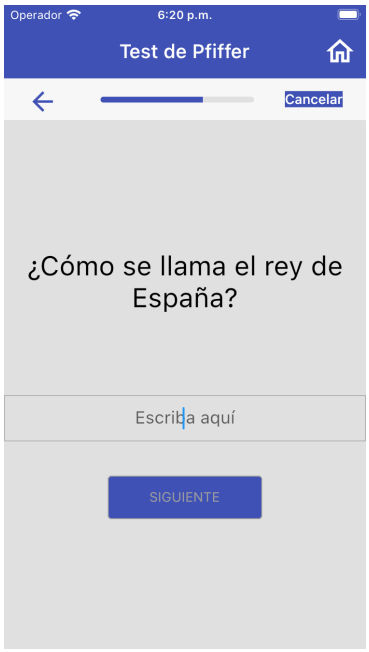

Figura 3.34: Pregunta del cuestionario seleccionado por el paciente.

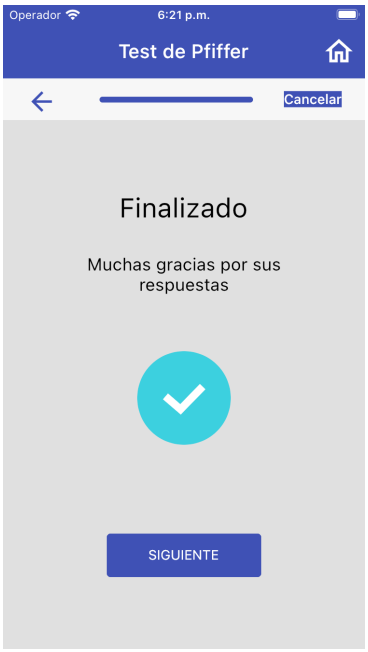

Figura 3.35: Fin del cuestionario.

Por lo que respecta a la introducción de datos manuales, en las Figuras 3.36, 3.37 y 3.38 se muestran las ventanas correspondientes al listado de posibles datos a introducir, introducción de datos de la prueba de velocidad y introducción de datos de carácter físico.

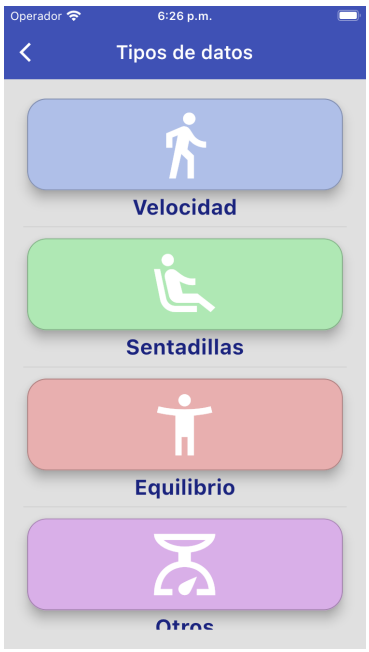

Figura 3.36: Tipos de datos que se pueden introducir.

Figura 3.37: Inserción de datos de la prueba de velocidad.

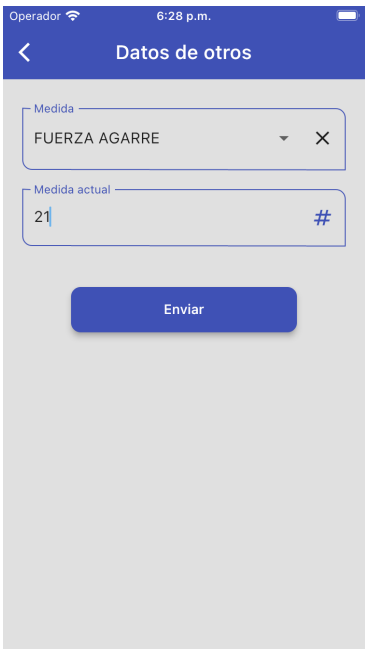

Figura 3.38: Inserción de datos físicos personales.

Por último, en relación a la posibilidad de visualizar la evolución de los datos de las pruebas, la Figura 3.39 muestra la interfaz final diseñada. Como se puede observar, se optó por mostrar los resultados mediante un gráfico en lugar de como se había especificado en los prototipos.

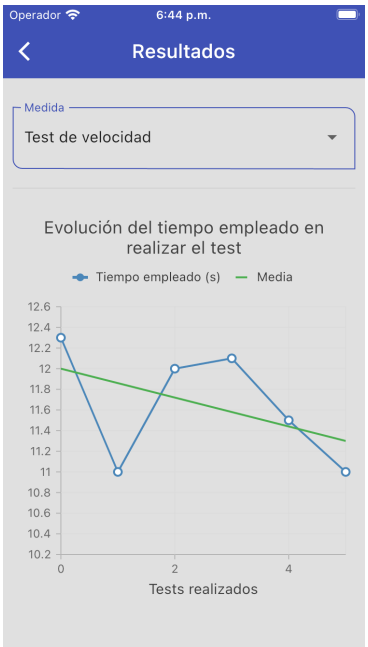

Figura 3.39: Tiempo empleado en realizar seis pruebas de velocidad en distintas fechas.

Capítulo 4

Implementación y pruebas

En este capítulo se explican la programación y las pruebas del proyecto, detallando cada parte, además de exponer todo lo relativo a la documentación del mismo.

4.1. Detalles de implementación

Antes de explicar en detalle todos los módulos que forman la parte del servidor y de la aplicación móvil, se va a exponer en la siguiente sección la configuración de las herramientas y tecnologías necesarias para que el sistema informático en explotación pueda funcionar.

4.1.1. Instalación y configuración de las herramientas

Por lo que respecta a la instalación, para poder almacenar los datos ha sido necesario instalar los dos gestores de bases de datos utilizados en este proyecto. Para poder realizar la instalación en un mayor número de sistemas operativos se ha decidido utilizar Docker. A continuación, se muestran los comandos utilizados para realizar cada una de las instalaciones. En primer lugar, para poder conectar todas las herramientas del servidor en Docker se ha creado una red de la siguiente manera.

```
$ docker create network frailty_network
```

Mediante esta red, todos los contenedores creados pueden operar e interactuar entre ellos sin ningún problema. Un contenedor Docker es un contenedor ejecutable, independiente y liviano que incluye todo lo necesario para ejecutar una aplicación, incluidas bibliotecas, herramientas del sistema, código y tiempo de ejecución. En este caso, a cada gestor de base de datos le corresponde un contenedor. Para crear el contenedor de PostgreSQL se han usado los siguientes comandos.

```
$ docker pull postgres
$ docker run --name frailty_postgres --network frailty_network -d
  -p 5432:5432 -e POSTGRES_PASSWORD=12345678 postgres
```

El primer comando descarga lo necesario para poder ejecutar el gestor de base de datos, mientras que el segundo comando crea el contenedor y lo conecta con la red creada anteriormente. Por otra parte, para crear el contenedor de Cassandra se han usado los siguientes comandos.

```
$ docker pull cassandra
$ docker run --name frailty_cassandra --network frailty_network -p
    7000:7000 -p 7001:7001 -p 7199:7199 -p 9042:9042 -p 9160:9160
  -p 9404:9404 -d cassandra
```

Como en el caso anterior, el primer comando descarga todo lo necesario para instalar Cassandra y el segundo comando crea el contenedor correspondiente. Para el caso de PostgreSQL, también ha sido necesario crear la base de datos, un usuario de acceso y su contraseña. Esto se ha realizado de la siguiente manera.

```
$ docker exec -it frailty_postgres bash
frailty_postgres $ psql postgres
frailty_postgres $ create database develop_postgres
frailty_postgres $ create user hibernate with password 'hibernate'
```

El primer comando permite entrar dentro del contenedor de PostgreSQL y ejecutar comandos. Una vez dentro del contenedor, se ha creado la base de datos y el usuario. Por otra parte, para poder usar el protocolo de mensajes MQTT, se ha instalado un contenedor siguiendo la lógica anterior mediante los siguientes comandos.

```
$ docker pull toke/mosquitto
$ docker run -ti --name mosqutto --network frailty_network -p
    1883:1883 -p 9001:9001 toke/mosquitto
```

Por último, para poder desplegar el contenedor que mantiene la aplicación del servidor, es necesario en primer lugar, construir el proyecto y empaquetar el archivo .jar resultante en el directorio *target*. Esto se consigue ejecutando el siguiente comando en el directorio raíz del proyecto.

```
$ ./mvnw clean package
```

Una vez obtenido el archivo ejecutable .jar, se puede utilizar Docker para construir el contenedor del servidor de manera similar a como se ha realizado anteriormente. Los siguientes comandos se deben ejecutar en la carpeta que contiene el proyecto.

```
$ docker build -f src/main/docker/Dockerfile.jvm -t quarkus/
    fragility-jvm .
$ docker run -i -d -p 8080:8080 --network frailty_network --name
    frailty_app quarkus/fragility-jvm
```

En relación a la configuración, la herramienta Quarkus utilizada para desarrollar el servidor utiliza por defecto Maven [1]. Este software es capaz de gestionar la construcción, los informes y la documentación de un proyecto desde un archivo de información central llamado *pom.xml*. En este archivo se pueden introducir, entre otras muchas cosas, las dependencias a librerías externas del proyecto. De esta manera, se consigue de una manera muy simple importar, por ejemplo, los drivers necesarios para poder conectarse y usar las bases de datos. En la Figura 4.1

se muestra un fragmento del archivo *pom.xml* del servidor. Como se puede observar, se importan los drivers necesarios para conectarse a las bases de datos PostgreSQL y Cassandra.

```xml
<!-- POSTGRES -->
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-jdbc-postgresql</artifactId>
</dependency>
<!-- Cassandra -->
<dependency>
  <groupId>com.datastax.oss.quarkus</groupId>
  <artifactId>cassandra-quarkus-client</artifactId>
  <version>1.1.1</version>
</dependency>
```

Figura 4.1: Dependencias necesarias para utilizar PostgreSQL y Cassandra.

De igual forma, se han añadido las dependencias necesarias para poder utilizar JPA, cuyo fin es el de ayudar a interactuar con la base de datos PostgreSQL utlizando el patrón de mapeo objeto-relacional (ORM). Además, también ha sido necesario añadir una dependencia para poder utilizar un cliente de MQTT para poder interactuar con la aplicación de la cámara Kinect.

Además del archivo *pom.xml*, Quarkus proporciona un archivo llamado *application.properties* que permite introducir toda la configuración necesaria para interactuar con las bases de datos, certificados de segurdidad SSL, opciones generales del proyecto, etc. En la Figura 4.2 se presenta la configuración necesaria para poder conectarse con las bases de datos pertenecientes al proyecto.

```
# POSTGRES
quarkus.datasource."develop_postgres".db-kind = postgresql
quarkus.datasource."develop_postgres".username = hibernate
quarkus.datasource."develop_postgres".password = hibernate
quarkus.datasource."develop_postgres".jdbc.url = jdbc:postgresql://frailty_postgres:5432/develop_postgres
quarkus.hibernate-orm."develop_postgres".database.generation=update
quarkus.hibernate-orm."develop_postgres".datasource=develop_postgres
quarkus.hibernate-orm."develop_postgres".packages=es.uji.model
quarkus.hibernate-orm."develop_postgres".sql-load-script=reloadAll.sql

# CASSANDRA
quarkus.cassandra.contact-points=localhost:9042
quarkus.cassandra.local-datacenter=datacenter1
quarkus.cassandra.keyspace=k1
```

Figura 4.2: Archivo de configuración *application.properties* del proyecto Quarkus.

En la Figura 4.2 se pueden visualizar dos bloques diferenciados. El bloque de configuración de PostgreSQL contiene la mayor parte de la configuración del archivo. Todas las líneas de configuración poseen la palabra *develop_postgres* entre comillas. Este palabra se corresponde con el nombre que se le ha dado a la base de datos PostgreSQL. Las primeras líneas del bloque indican el tipo de base de datos, el usuario y la contraseña de acceso. Además, también se especifica la dirección URL desde la cual se puede acceder y el tipo de inicio que debe tener

la base de datos cada vez que se ejecuta el programa. Por último, en las dos últimas líneas del bloque se especifica el paquete donde se encuentran las entidades que se usan en la base de datos y el nombre del archivo que se ejecuta cada vez que se reinicia y que contiene algunas consultas de inserción que sirven para añadir un paciente de prueba. Por otra parte, el bloque de Cassandra contiene el nombre de la base de datos, la dirección url de acceso y el nombre de un espacio clave que es propio de este gestor de base de datos. Como se puede apreciar, la configuración de Cassandra es mucho más sencilla. Esto se debe a que esta es una base de datos no relacional y no necesita de apenas información para guardar los datos.

Por lo que respecta a la aplicación móvil, la instalación realizada se obvia en este documento ya que el lector puede ir a la página oficial de Flutter [11] y seguir los pasos de instalación básica seguidos también por el autor del documento. En relación a la configuración, Flutter utiliza de manera similar a Quarkus un archivo de configuración llamado *pubspec.yaml*. Si el lector de este documento intentara ejecutar la aplicación, debería de ir a la carpeta raíz del proyecto y ejecutar el siguiente comando.

```
$ flutter pub get
```

Este comando utiliza el archivo *pubspec.yaml* para crear y ejecutar la aplicación. Su contenido más importante se puede ver en la Figura 4.3, donde se muestran las librerías externas utilizadas para el desarrollo del proyecto. Las más relevantes son *provider*, *survey_kit* y *syncfusion_flutter_charts*. La primera se ha utilizado para almacenar el estado de la aplicación como se ha mencionado en el Capítulo 3. La segunda ha sido de ayuda para implementar las funcionalidades y las interfaces de la historia de usuario referente a la contestación de cuestionarios. Por último, la tercera librería ha sido utilizada para generar los gráficos que muestran el tiempo empleado en completar las diferentes pruebas físicas.

```
cupertino_icons: ^1.0.2
font_awesome_flutter: ^9.2.0
http: ^0.13.4
provider: ^6.0.2
flutter_barcode_scanner: ^2.0.0
flutter_form_builder: ^7.1.1
form_builder_validators: ^7.6.1
form_builder_image_picker: ^2.0.0
shimmer: ^2.0.0
survey_kit: ^0.0.21
syncfusion_flutter_charts: ^19.4.56
```

Figura 4.3: Parte de la configuración de la aplicación móvil desarrollada en Flutter.

4.1.2. Módulos del servidor

A continuación, se procede a detallar cada uno de los módulos que componen la parte del código del servidor. Estos son *models*, *services* y *resources*. Primero, se mostrarán los módulos *models* y *services*, que son aquellos que tienen que ver con la base de datos. Por último, se describirá el módulo *resources* que contiene las rutas por las que se accede a los recursos.

Models. En este módulo se han definido los esquemas del modelo de datos que las bases de datos usan para persistir la información. Para ello, dentro del paquete principal del programa

se ha creado una carpeta *models*, y dentro de dicha carpeta se han creado todos los ficheros java relacionados con los modelos. Los ficheros *Patient*, *Measure*, *TestResult*, *GaitSpeedTestResult*, *GetUpTestResult*, *BalanceTestResult*, *Questionnaire*, *QuestionnaireResponse*, *Item* y *ItemAnswer* se corresponden con las entidades modeladas en el Capítulo 3 y contienen los atributos básicos que definen a cada una, los *getters* y las anotaciones JPA necesarias para relacionar cada atributo con una columna en la base de datos PostgreSQL. Además, los ficheros *ItemType*, *MeasureType* y *TestResultType* pertenecen a las enumeraciones también definidas anteriormente.

También se han creado los ficheros *Position*, *Joint* y *BodyJointTestResult*. Estos ficheros contiene los atributos básicos que los definen y una etiqueta con un nombre que los identifica para poder ser almacenados en la base de datos Cassandra. Como la base de datos es no relacional, el sistema no necesita de etiquetas particulares para cada atributo como en el caso de PostgreSQL.

Además, se ha creado un fichero de ayuda *PatientDeviceCredentials* para gestionar los datos que se utilizan en la comunicación entre la aplicación de la cámara Kinect y la aplicación móvil cuando se escanea el código QR. Este fichero contiene un atributo para guardar el DNI del paciente y otro para guardar el identificador del dispositivo escaneado.

Services. Este módulo corresponde a la capa de acceso a la base de datos. Cada vez que se quiere acceder a la base de datos se hace uso de los servicios que provee este módulo. Para ello, se ha creado dentro del directorio raíz un subdirectorio llamado *services*, y dentro de dicho subdirectorio se han creado los ficheros *IBodyJoints*, *IMeasure*, *IPatient*, *IQuestionnaire*, *IQuestionnaireResponse* y *ITestResult*, que son interfaces con las operaciones CRUD para la lectura y escritura de las entidades creadas en el módulo *models*. Para mostrar con un poco más de detalle estos archivos, a continuación se explican brevemente los métodos que posee uno de los archivos, concretamente *IQuestionnaire*:

- **Questionnaire create(Questionnaire questionnaire)**: guarda un cuestionario en la base de datos y lo devuelve.
- **Questionnaire retrieve(int id)**: devuelve un cuestionario dado su identificador.
- **Questionnaire update(Questionnaire questionnaire)**: actualiza un cuestionario existente y lo devuelve.
- **Questionnaire delete(int id)**: borra un cuestionario dado su identificador.
- **Collection<Questionnaire> retrieveAll()**: devuelve una lista con todos los cuestionarios guardados en el sistema.
- **Collection<Questionnaire> retrieveAvailableForPatients()**: devuelve todos los cuestionarios disponibles para los pacientes.
- **Collection<Questionnaire> retrieveNonAvailableForPatients()**: devuelve todos los cuestionarios no disponibles para el paciente.

El resto de interfaces tienen métodos muy similares, tan solo cambia el modelo con el que operan. Las interfaces definidas son implementadas por otros archivos pertenecientes a este

módulo. La nomenclatura que se ha seguido ha sido la de usar el nombre del modelo al que implica el servicio acompañado de la palabra DAO. Así pues, en este módulo también existen los archivos *BodyJointsDAO*, *MeasureDAO*, *PatientDAO*, *QuestionnaireDAO*, *QuestionnaireResponseDAO* y *TestResultDAO* que implementan los métodos a los que hace referencia su correspondiente interfaz.

Por último, en este módulo también se ha creado el archivo *MQTTService*. Este archivo está diseñado con el patrón *Singleton* y posee los métodos necesarios para crear el cliente MQTT, suscribirse a un *topic* para recibir mensajes, publicar mensajes en un *topic* y comprobar si el cliente está conectado. Al usar el patrón *Singleton*, este archivo es accesible desde cualquier otro archivo del servidor.

Resources. Dentro del directorio raíz del proyecto se ha creado un subdirectorio llamado *Resources*. Este módulo contiene los controladores de las rutas API REST del sistema. Al igual que con el módulo *Services*, se han creado varios archivos para controlar cada uno de los modelos del sistema de manera separada. Los archivos existentes en este subdirectorio son *BodyJointsResource*, *MeasureResource*, *PatientResource*, *QuestionnaireResource*, *QuestionnaireResponseResource* y *TestResultResource*. Cada uno de estos archivos realiza una inyección de dependencias del servicio correspondiente para poder realizar las operaciones necesarias en la base de datos. Como en el módulo anterior, a continuación se describen los métodos del archivo *QuestionnaireResource* como ejemplo de lo que contienen el resto de archivos:

- **Response getQuestionnaire(@PathParam('id') final int id)**: método llamado al acceder a la ruta '/questionnaire/retrieve/id'. Recibe un identificador que usa para devolver un cuestionario existente. Si el cuestionario existe, el método devuelve una respuesta con el cuestionario y un código de éxito 200. De lo contrario, devuelve una respuesta con el código de error 404.
- **Response getQuestionnaires()**: método llamado al acceder a la ruta '/questionnaire/retrieve/all'. Devuelve una lista con todos los cuestionarios almacenados en la base de datos y un código de éxito 200. Si no existen cuestionarios en el sistema, devuelve el mismo código de éxito y una lista vacía.
- **Response getAvailableQuestionnaires()**: método llamado al acceder a la ruta 'questionnaire/retrieve/available'. Devuelve una lista con todos los cuestionarios disponibles y un código de éxito 200. Si no existen cuestionarios disponibles en el sistema, devuelve el mismo código de éxito y una lista vacía.
- **Response getNoNBinaryQuestionnaires()**: método llamado al acceder a la ruta 'questionnaire/retrieve/non-available'. Devuelve una lista con todos los cuestionarios no disponibles para el paciente y un código de éxito 200. Si no existen cuestionarios no disponibles para el paciente en el sistema, devuelve el mismo código de éxito y una lista vacía.
- **Response saveQuestionnaire(Questionnaire questionnaire)**: método llamado al acceder a la ruta questionnaire/create'. Recibe un cuestionario para ser almacenado en la base de datos. Si no se produce ningún error se almacena el cuestionario y devuelve el identificador en la respuesta con un código de éxito 201. Si por el contrario se produce algún error se devuelve un mensaje de error 409.

En este módulo también se ha creado el archivo *MQTTResource* que contiene solo un método POST llamado *activate_device*. Este método es llamado cuando un paciente escanea el código QR de un dispositivo y envía a ese dispositivo un mensaje con el DNI del paciente para que pueda ser identificado.

4.1.3. Módulos de la aplicación móvil

En este apartado se presentan los módulos que componen la parte del código de la aplicación en Flutter. Estos son *models*, *providers*, *router*, *screens*, *services*, *themes*, *ui* y *widgets*. A continuación se detalla cada uno de ellos.

Models. En este módulo se han definido los esquemas del modelo de datos de igual forma que en el servidor. Para ello, se ha creado dentro del directorio *lib* un subdirectorio llamado *models*, y dentro de dicho subdirectorio se han creado los ficheros *balance_test_result*, *measure*, *patient*, *questionnaire*, *questionnaire_response*, *speed_test_result*, *squats_test_result* y *test_result*, cada uno para cada esquema. Cada archivo contiene los atributos necesarios del modelo correspondiente y métodos estáticos *toMap()* y *toJson()* para permitir operar con los modelos en diferentes escenarios.

Además, en este módulo se han creado otros dos archivos de ayuda para el paso de información entre ventanas. Por un lado, el archivo *default_card* representa un contenedor visual con un tamaño, color, título e icono que se repite en diferentes ventanas de la interfaz. Este contenedor actúa como un botón que cuando es presionado por el paciente, realiza una acción. Esta acción también se almacena en el archivo como un atributo más. Por otro lado, el archivo *menu_options* representa cada una de las ventanas existentes en la aplicación. Los atributos que se controlan en este archivo son la ruta de acceso, el nombre y el componente visual que despliega. Este archivo ha sido usado para controlar la navegación de la aplicación de una manera más sencilla.

Providers. En este módulo se han creado los archivos necesarios para gestionar el estado de la aplicación. Concretamente, los archivos controlan las actualizaciones de los datos que se muestran por pantalla en tiempo real. Esto significa que si se está mostrando una lista con tres cuestionarios y se añade otro, la lista se actualiza mostrando cuatro cuestionarios. Además, estos archivos también controlan los cambios que se producen desde la interfaz y que deben ser mantenidos en el sistema. Algo a destacar es que todos estos archivos son accesibles desde cualquier parte del sistema debido a que están implementados mediante el patrón *Singleton*. Los archivos que controlan el estado creados en este módulo son *user_provider*, *measure_provider*, *test_result_provider* y *questionnaire_provider*. Estos archivos se detallan brevemente a continuación:

- **user_provider**: este archivo controla el estado del campo DNI introducido en el formulario de inicio de sesión. Cada vez que se realiza un cambio es notificado. Una vez el paciente ha iniciado sesión, el archivo guarda el DNI y es accesible desde cualquier interfaz y se utiliza para realizar las peticiones HTTP necesarias.

- **measure_provider**: este archivo gestiona el estado de los valores introducidos por el paciente en el formulario correspondiente a la entrada manual de datos físicos personales.
- **test_result_provider**: este archivo controla el estado de los valores introducidos por el paciente en el formulario correspondiente a la entrada manual de datos relativos a las pruebas físicas.
- **questionnaire_provider**: este archivo gestiona el estado de la lista de cuestionarios disponibles para el paciente en el sistema. La primera vez que se accede a la interfaz, el archivo obtiene los cuestionarios disponibles mediante un servicio y los almacena para ser mostrados en la interfaz. Además, este archivo también controla el cuestionario que el paciente está respondiendo. De esta manera, cuando el paciente termina el cuestionario y presiona sobre el botón correspondiente para enviarlo, se utiliza este archivo para obtener el identificador del cuestionario.

Router. Este módulo contiene un único archivo llamado *app_routes*. Su función es la de indicar a la aplicación la ruta inicial de la interfaz que se debe mostrar. Además, mantiene una lista de *menu_option*, anteriormente descrito, para poder gestionar la navegación entre interfaces de una manera más cómoda. Al inicio del programa, se generan todas las rutas con sus correspondientes interfaces. Cuando un paciente realiza alguna acción que implica el cambio de ventana, se invoca este archivo y se accede al *menu_options* correspondiente para cambiar la interfaz mostrada.

Screens. Este módulo contiene un archivo por cada ventana que posee la aplicación. Concretamente, los archivos creados en este módulo son *cognitive_test_list_screen*, *cognitive_test_screen*, *form_test_data_input_screen*, *input_type_screen*, *patient_home_screen*, *patient_login_screen*, *results_screen*, *scan_device_screen* y *test_type_screen*. A continuación, se describe brevemente cada uno de ellos:

- **cognitive_test_list_screen**: interfaz que muestra una lista de cuestionarios disponibles para el paciente. Permite al paciente seleccionar un cuestionario.
- **cognitive_test_screen**: interfaz que muestra las preguntas de un cuestionario seleccionado por el paciente. Dependiendo del tipo de pregunta, permite al paciente escribir una respuesta o seleccionar una o varias respuestas entre todas las disponibles. Además, también permite al paciente volver a una pregunta anterior para cambiar la respuesta si lo considera necesario.
- **form_test_data_input_screen**: interfaz que muestra un formulario para que el paciente introduzca alguna clase de dato.
- **input_type_screen**: interfaz que muestra cuatro elementos que el paciente puede seleccionar y que hacen referencia a las tres pruebas físicas y a la entrada de datos físicos personales. Este interfaz sirve de menú al paciente para seleccionar qué tipo de dato quiere introducir.
- **patient_home_screen**: interfaz del menú principal del paciente.
- **patient_login_screen**: interfaz que muestra un formulario donde el paciente puede introducir su DNI para iniciar sesión en la aplicación.

- **results_screen**: interfaz que muestra un gráfico con la evolución del tiempo empleado en realizar cada una de las pruebas físicas. También muestra el tiempo medio empleado.

- **scan_device_screen**: interfaz que permite al paciente activar la cámara del teléfono móvil para poder escanear un código QR.

- **test_type_screen**: interfaz que permite al paciente seleccionar la opción de realizar una prueba física escaneando un dispositivo o una prueba cognitiva respondiendo un cuestionario.

Para ilustrar al lector con un ejemplo del código implementado en estos archivos, en la Figura 4.4 se presenta el contenido del archivo *test_type_screen*. Como se puede ver, el archivo es una clase que extiende de *StatelessWidget*. Esto indica a Flutter que la ventana no gestiona ningún estado. Para dibujar la interfaz, se debe sobrescribir el método *build*, que devuelve un componente o *widget*. En la línea 26 se puede observar que el *widget* que Flutter interpreta como una ventana es *Scaffold*. Este componente está compuesto por una barra de aplicación o *app bar* y de un cuerpo, que puede contener a su vez otros componentes más pequeños. En este caso, el cuerpo contiene un componente personalizado llamado *FixedSizeCardList* que se detalla más adelante.

```
5  class TestTypeScreen extends StatelessWidget {
6    const TestTypeScreen({Key? key}) : super(key: key);
7
8    @override
9    Widget build(BuildContext context) {
10     Size size = MediaQuery.of(context).size;
11
12     final options = [
13       DefaultCard(
14           color: const Color(0xffAFBFE8),
15           title: "Física",
16           route: "patient_scan_device",
17           icon: Icons.fitness_center_outlined,
18           height: size.height / 3.2), // DefaultCard
19       DefaultCard(
20           color: const Color(0xffE8AFAF),
21           title: "Cognitiva",
22           route: "patient_cognitive_test_list",
23           icon: Icons.psychology_outlined,
24           height: size.height / 3.2), // DefaultCard
25     ];
26     return Scaffold(
27       appBar: AppBar(
28         title: const Text("Tipo de prueba"),
29       ), // AppBar
30       body: FixedSizeCardList(options: options),
31     ); // Scaffold
32   }
33 }
```

Figura 4.4: Contenido del archivo *test_type_screen*.

Services. En este módulo se han creado los archivos necesarios para realizar las peticiones HTTP al servidor. Para cada modelo se ha creado un archivo de servicio distinto para separar el código de cada uno de ellos. Los archivos creados en este módulo han sido *balance_test_service*, *measure_service*, *patient_service*, *questionnaire_response_service*, *questionnaire_service*, *speed_test_service*, *squats_test_service* y *test_result_service*. Como ejemplo, en la Figura 4.5 se muestra el código empleado en el archivo *questionnaire_service* para recibir todos los cuestionarios disponibles para el paciente.

```
getQuestionnaires() async {
  _isLoading = true;
  final url = Uri.http(MyApp.SERVER_URL, 'questionnaire/retrieve/available');
  try {
    final http.Response resp = await http.get(url);
    if (resp.statusCode == 200) {
      final List<dynamic> jsonList = jsonDecode(resp.body);
      List<Questionnaire> questionnaireList = jsonList.map((data) {
        return Questionnaire.fromMap(data);
      }).toList();
      _isLoading = false;
      return questionnaireList;
    }
    _isLoading = false;
    return null;
  } catch (e) {
    _isLoading = false;
    return null;
  }
}
```

Figura 4.5: Llamada HTTP GET para obtener los cuestionarios disponibles para el paciente.

Además, en este módulo se ha creado el archivo *send_input_interface* que contiene un método *sendFormData* y que es implementado por los archivos *balance_test_service*, *speed_test_service*, *squats_test_service* y *measure_service*. La razón es que todos estos archivos comparten la misma funcionalidad, enviar al servidor un dato introducido por el paciente.

Themes. La razón de este módulo es la de mantener los distintos temas que se puedan crear para la aplicación. Aunque en esta aplicación solo existe un tema controlado por el archivo *app_theme*, otros temas podrían ser añadidos en este módulo y permitirían cambiar el aspecto visual de la aplicación con tan solo cambiar una línea en el archivo *main*. El archivo creado en este módulo gestiona el color principal de la aplicación, color del fondo de pantalla, decoración de los componentes que permiten la entrada de datos, bordes, etc. En resumen, todo lo referente al aspecto visual general de la aplicación. En la Figura 4.6 se muestra el código utilizado en este proyecto para diseñar el aspecto visual básico de la aplicación móvil.

```dart
static const Color primary = Colors.indigo;
static final Color? primary_900 = Colors.indigo[900];

static final ThemeData lightTheme = ThemeData.light().copyWith(
  scaffoldBackgroundColor: Colors.grey[300],
  appBarTheme: const AppBarTheme(color: primary, elevation: 0),
  inputDecorationTheme: const InputDecorationTheme(
    floatingLabelStyle: TextStyle(color: primary),
    enabledBorder: OutlineInputBorder(
      borderSide: BorderSide(color: primary),
      borderRadius: BorderRadius.only(
          bottomLeft: Radius.circular(10), topRight: Radius.circular(10)),
    ), // OutlineInputBorder
    focusedBorder: OutlineInputBorder(
      borderSide: BorderSide(color: primary),
      borderRadius: BorderRadius.only(
          bottomLeft: Radius.circular(10), topRight: Radius.circular(10)),
    ), // OutlineInputBorder
    border: OutlineInputBorder(
      borderRadius: BorderRadius.only(
          bottomLeft: Radius.circular(10), topLeft: Radius.circular(10)),
    ), // OutlineInputBorder
    iconColor: primary,
  ), // InputDecorationTheme
);
```

Figura 4.6: Contenido del archivo *app_theme*.

UI. Este módulo se ha creado como ayuda para gestionar los diálogos o ventanas emergentes que se utilizan en la aplicación para notificar al usuario del éxito o fracaso de algunas operaciones. Existe un único archivo llamado *alert_dialog* que contiene tres métodos, uno público accesible desde otros archivos y dos privados. El método público *showBasicAlert()* comprueba el sistema operativo donde se está ejecutando la aplicación y dependiendo del resultado ejecuta uno de los métodos privados *_displayBasicDialogAndroid()* o *_displayBasicDialogIOS()*. Aunque lo que se muestra en ambos sistemas operativos es lo mismo, cada uno tiene particularidades que es necesario controlar desde Flutter.

Widgets. En este módulo se han creado los archivos necesarios para reutilizar fragmentos de código iguales en diferentes interfaces o para separar un trozo de código grande dotándole de funcionalidad propia. Esta forma de separar el código en componentes más pequeños viene dada por Flutter y es una de las ventajas de usar esta herramienta. Los archivos creados en este módulo han sido *auth_background*, *card_container*, *custom_material_button*, *custom_number_input_field*, *fixed_size_card_list*, *icon_card*, *measures_dropdown*, *patient_survey_kit*, *result_test_dropdown*, *skeleton_container* y *test_result_linear_chart*. Cada uno de ellos se describe brevemente a continuación:

- **auth_background**: componente usado para el diseño del fondo de pantalla en la ventana de inicio de sesión del paciente.
- **card_container**: componente que representa un contenedor al que se le pasa por parámetro el ancho y el componente hijo que debe contener y por lo tanto, dibujar.

- **custom_material_button**: componente que dibuja un botón y que ha sido reutilizado para que todos los botones de la aplicación tengan el mismo aspecto.
- **custom_number_input_field**: componente que se reutiliza en todas las entradas de datos relacionadas con las pruebas físicas o con los datos de carácter personal. Este componente solo permite introducir números.
- **fixed_size_card_list**: componente que muestra una lista fija de *default_card*, explicado en el módulo *models*.
- **icon_card**: contenedor genérico con un icono en el centro. Recibe por parámetro el color del contenedor, el icono, la altura y el texto a mostrar. Este componente es reutilizado en todas las ventanas donde se muestran listas o menús.
- **measures_dropdown**: componente que permite al paciente seleccionar entre los diferentes tipos de medidas de carácter personal que se pueden introducir en la aplicación. Actualmente el paciente puede seleccionar peso, altura o fuerza de agarre.
- **patient_survey_kit**: componente que recibe un cuestionario y con la ayuda de la librería *survey_kit* dibuja y controla la interfaz relacionada con la visualización y respuesta de los cuestionarios.
- **result_test_dropdown**: componente que permite al paciente seleccionar entre los diferentes tipos de pruebas físicas para poder visualizar el gráfico de la evolución del tiempo empleado en realizar la prueba correspondiente.
- **skeleton_container**: componente que se muestra mientras los datos del servidor están siendo solicitados. Este componente se usa en todas las listas y menús donde se realizan peticiones GET al servidor para obtener los datos a mostrar.
- **test_result_linear_chart**: componente que dibuja una gráfico lineal con la ayuda de la librería *syncfusion_flutter_charts*. Este componente fue creado debido a que se reutiliza tres veces, una por cada tipo de prueba física.

4.2. Verificación y validación

Para verificar y validar la funcionalidad de ambos programas, el del servidor y el de la aplicación móvil, se han usado las técnicas estudiadas durante los cuatro años del grado.

Por lo que respecta a la verificación, se ha usado la técnica de análisis de código estático. Concretamente, al finalizar cada historia de usuario se ha realizado una lectura ordenada del código implementado. Para controlar el estado de las variables en cada momento se ha utilizado una hoja y una papel. De esta manera se ha seguido la traza del programa mientras se buscaban bugs, nombres de variables poco específicos, métodos demasiado largos, código muerto, etc. Para realizar este procedimiento se ha contado con la ayuda del supervisor de la empresa, que hacía el papel de moderador y en todo momento detectaba el posible fallo, no la solución. Este análisis del código para cada historia de usuario ha ido acompañado de sus correspondientes pruebas de validación, como se detalla a continuación.

En relación a las pruebas de validación, se han diseñado diferentes tests de caja negra y caja blanca para probar las distintas funcionalidades de las aplicaciones desarrolladas. Para ello se han creado diferentes escenarios y casos de prueba para algunas pruebas unitarias relacionadas con las historias de usuario, tanto del servidor como de la aplicación móvil. Como ejemplo, en el Cuadro 4.1 se presentan los escenarios de caja blanca diseñados para la prueba unitaria correspondiente al envío de datos de una prueba de velocidad introducida de forma manual. El resto de historias de usuario han sido probadas de manera similar. En esta prueba unitaria, se validan dos valores introducidos por parte del paciente para su posterior envío a la base de datos. La variable *DR* hace referencia a la distancia recorrida, mientras que la variable *TE* se refiere al tiempo empleado en realizar la prueba.

ID	Condición	Cuando True	Cuando False	Escenarios
EB01	DR != null	Cuando el paciente ha introducido algún carácter.	Cuando el paciente no ha introducido nada.	EB01T, EB01F
EB02	DR.isValidDigit == true	Cuando el dato introducido es un número mayor que 0.	Cuando el valor introducido no es un número mayor que 0.	EB02T, EB02F
EB03	TE != null	Cuando el paciente ha introducido algún carácter.	Cuando el paciente no ha introducido nada.	EB03T, EB03F
EB04	TE.isValidDigit == true	Cuando el dato introducido es un número mayor que 0.	Cuando el valor introducido no es un número mayor que 0.	EB04T, EB04F

Cuadro 4.1: Escenarios de caja blanca de un test unitario de la HU03.

De la misma manera, a continuación se presentan los escenarios diseñados para las pruebas de caja negra de la misma prueba unitaria. En el Cuadro 4.2 se detallan los escenarios definidos.

Dato	Clases válidas	Clases inválidas
in: DR	V1: DR.length mayor que 0, V2: DR.isDigit() == true, V3: DR mayor que 0	I1: DR.length == 0
in: TE	V4: TE.length mayor que 0, V5: TE.isDigit() == true, V6: TE mayor 0	I2: TE.length == 0
(DR, TE)	V7: V1 y V4, V8: V2 y V5, V9: V3 y V5	

Cuadro 4.2: Escenarios de caja negra de un test unitario de la HU03.

Los casos de prueba diseñados para los escenarios anteriores se pueden ver en el Cuadro 4.3. Como se puede observar, si la primera condición no se cumple, la segunda no llega a ser evaluada.

ID	Entrada	Salida	Clases representadas
CP01	DR = 3, TE = 4.3	True	EB01T, EB02T, EB03T, EB04T, V7, V8, V9
CP02	DR = null, TE = -2	False	EB01F
CP03	DR = 3, TE = null	False	EB03F, V1, V2
CP04	DR = -3, TE = 'algo'	False	EB02F
CP05	DR = 3, TE = 'algo'	False	EB04F, V1, V2
CP06	DR = 3, TE = ''	False	I1
CP07	DR = '', TE = 4.3	False	I2

Cuadro 4.3: Casos de prueba de un test unitario de la HU03.

En relación a las pruebas de integración, en este proyecto se han realizado las pruebas de integración siguiendo el método de integración no incremental o *Big-Bang*, donde cada componente ha sido probado por separado mediante las pruebas unitarias y posteriormente se han integrado todos de una vez. Para comprobar el buen funcionamiento de la integración se ha recurrido a los *logs* que proporciona Docker de forma nativa.

Para finalizar, se han realizado algunas pruebas de sistema de manera muy simple. Además de las pruebas funcionales para comprobar que todas las historias de usuario tienen implementada su funcionalidad, se ha realizado una prueba de rendimiento para comprobar que los tiempos de respuesta eran adecuados.

Capítulo 5

Conclusiones

Después de la realización de este proyecto, he de decir que estoy muy contento en general y muy agradecido por la oportunidad de haberlo llevado a cabo en mi estancia en prácticas. No solo a nivel personal, sino también a nivel profesional. Poder ser capaz de planificar, analizar, diseñar e implementar un proyecto tan completo por mí mismo resulta una sensación muy reconfortante de cara al futuro.

Por un lado, cabe mencionar que algunas de las tecnologías que se han usado en este proyecto han sido nuevas para mí, por lo que el interés y la motivación a la hora de decidirme a aprender a usarlas ha sido vital para que el proyecto saliese adelante. Por otro lado, aunque había tecnologías nuevas para mí, he podido hacer uso de metodologías y procedimientos aprendidos a lo largo de la carrera; por lo que podría decirse que la realización de este proyecto significa, en cierto sentido, un éxito académico.

Durante la realización de este proyecto, me he dado cuenta de las competencias adquiridas durante los cuatro años de grado. Obviamente, todas las asignaturas de programación y bases de datos me han ayudado a formarme para ser capaz de implementar el servidor y la aplicación de este proyecto. Sin embargo, la realización de este documento ha sido posible gracias a las asignaturas del itinerario en Ingeniería del Software, ya que me han ayudado a entender y estructurar este tipo de documento técnico. Además, también me gustaría destacar la utilidad de la asignatura Diseño e Implementación de Sistemas de la Información (EI1027), debido a que ha sido de gran ayuda para poder realizar un buen diseño de interfaces. De igual forma, la asignatura Fundamentos de la Ingeniería del Software (EI1023) ha resultado vital para poder realizar el análisis del sistema desarrollado.

A lo largo del desarrollo de este proyecto han surgido algunas dificultades. Algunas de ellas pude resolverlas con el supervisor, pero al ser también Flutter una tecnología relativamente nueva para él y el resto de la oficina, la mayoría de dificultades tuve que afrontarlas por mí mismo mediante la búsqueda de los errores que me iban apareciendo por internet. Además, hay que tener en cuenta que tras la pandemia del COVID-19, la mayoría de trabajadores estaban en casa realizando sus labores de manera virtual, por lo que preguntarles a ellos resultaba un poco complicado.

Las principales dificultades que he encontrado a la hora de realizar este proyecto han sido las siguientes. Por una lado, el diseño de la aplicación costó más de lo esperado debido a cambios constantes de opinión del cliente. Tras haber realizado el primer diseño, se pidió un cambio por completo. Tras realizar este cambio desde cero, se volvió al primer diseño. Este proceso tardó unas dos semanas, en las cuales pude ocupar mi tiempo realizando el curso de Flutter. Por otro lado, la falta de experiencia en Dart y Flutter ha provocado que algunos problemas sencillos como la conversión de datos del modelo para su posterior envío haya resultado costoso. Siempre solían aparecer problemas muy pequeños que me costaban varias horas de resolver. Por último, algo que me costó mucho de solucionar fue la implementación del servicio MQTT. Aunque el proceso de instalación del *broker* y la creación del clientes resultó sencillo, tras seguir la documentación concienzudamente, no funcionaba nada. Este problema resultó tener su origen en la conexión de internet que utilizaba. Como el espacio de trabajo de la empresa 480 se encuentra dentro de la universidad, la red que yo utilizaba era la de la propia UJI. Esta red no permite la comunicación por los puertos que utiliza MQTT. La solución a este problema fue conectarme a otra red independiente de la universidad. Aunque la solución escrita aquí es muy sencilla, no la obtuve hasta prácticamente el final del proyecto.

El buen ambiente en la oficina y la buena relación con mis compañeros ha hecho que mi estancia en prácticas fuera muy agradable y donde me he sentido uno más en todo momento, por lo que estoy muy agradecido.

Comparando los objetivos del proyecto con el subsistema final desarrollado, creo que he conseguido alcanzar todas las funcionalidades que se demandaban. Obviamente, el subsistema necesita ser probado en un ambiente con pacientes reales. Las pruebas que se han realizado para comprobar el buen funcionamiento de la aplicación están condicionadas a la cantidad de datos disponibles. Por ejemplo, para poder probar si la implementación relacionada con los cuestionarios funciona bien, he dispuesto de dos cuestionarios reales que son utilizados actualmente por los especialistas médicos. Por lo tanto, el subsistema podría necesitar de cambios futuros para poder ser adaptado a una mayor cantidad de cuestionarios. Sin embargo, durante el desarrollo del proyecto esto se ha tenido en cuenta, por lo que creo que sería relativamente fácil de implementar. Respecto a las pruebas físicas, la aplicación permite al paciente realizar todas las funcionalidades detalladas en los objetivos del proyecto. La inserción tanto de los datos relacionados con las tres pruebas físicas como los relacionados con el peso, altura y fuerza de agarre se puede realizar de manera sencilla como se pretendía. En relación al seguimiento de la evolución del rendimiento en las pruebas físicas, creo que aunque he alcanzado el objetivo que se planteaba, el resultado visual puede resultar demasiado sencillo. Esto es debido a que solo se muestra un gráfico lineal de una variable (tiempo) con una pequeña leyenda. Pensándolo ahora, podría haber implementado un gráfico con las tres pruebas juntas o otro tipo de gráfico más llamativo para el paciente. Finalmente, el sistema de inicio de sesión mediante el escáner del código QR ha quedado, bajo mi punto de vista, muy elegante. He podido probar el sistema con mi teléfono móvil personal y poder iniciar sesión con tan solo escanear la pantalla donde se encuentra el subsistema con la cámara resulta muy gratificante.

Por último, me gustaría destacar que durante la realización de este proyecto no solo he aprendido una nueva tecnología actual que pienso que me va a servir en el futuro. Para mí lo más importante ha sido el aprendizaje en el concepto de fragilidad y conocer como puede llegar a afectar a personas que tienen la edad de mi madre. Este trabajo me ha ayudado a ser consciente del problema y no descarto continuar investigando de alguna manera sobre este tema.

Bibliografía

[1] Apache. Welcome to Apache Maven. `https://maven.apache.org`, 2002. Acceso el 20-05-2022.

[2] Apache. Manage massive amounts of data, fast, without losing sleep. `https://cassandra.apache.org/_/index.html`, 2009. Acceso el 01-04-2022.

[3] Soluciones Cuatroochenta. 480. `https://cuatroochenta.com`, 2014. Acceso el 30-03-2022.

[4] Soluciones Cuatroochenta. Soluciones digitales cloud y ciberseguridad para afrontar los retos de tu organización. `https://cuatroochenta.com/sobre-cuatroochenta/`, 2014. Acceso el 30-03-2022.

[5] Ewing SK Ensrud KE. Comparison of 2 frailty indexes for prediction of falls, disability, fractures, and death in older women. *Arch Intern Med*, 168(4):382–9, 2008.

[6] Sieber C Fielding RA. Psychological frailty in the aging patient. *Quality of life research*, 83(1):45–53, 2015.

[7] van Assen M Gobbens R. The prediction of quality of life by physical, psychological and social components of frailty in community-dwelling older people. *Nestlé Nutrition Institute Workshop, Barcelona*, 23(1):2289–2300, 2014.

[8] Google. Paint your UI to life. `https://dart.dev`, 2011. Acceso el 07-04-2022.

[9] Google. Material Design. `https://material.io`, 2013. Acceso el 01-06-2022.

[10] Google. Orquestación de contenedores para producción. `https://kubernetes.io/es/`, 2014. Acceso el 07-04-2022.

[11] Google. Build apps for any screen. `https://flutter.dev`, 2017. Acceso el 07-04-2022.

[12] Red Hat. A Java framework tailored for deployment on Kubernetes. `https://quarkus.io`, 2019. Acceso el 01-04-2022.

[13] Solomon Hykes. Developers Love Docker. Businesses Trust It. `https://www.docker.com`, 2013. Acceso el 20-05-2022.

[14] Fried LP. Frailty in older adults: evidence for a phenotype. *J Gerontol A Biol Sci Med Sci*, 56(3):146–56, 2001.

[15] Tang Z Ma L, Sun F. Social frailty is associated with physical functioning, cognition, and depression, and predicts mortality. *J Nutr Health Aging*, 22(8):989–95, 2018.

[16] Microsoft. Kinect for Windows SDK 2.0. https://developer.microsoft.com/en-us/windows/kinect/, 2013. Acceso el 07-04-2022.

[17] Ikujiro Nonaka. Welcome to the Home of Scrum! ™. https://www.scrum.org, 1986. Acceso el 11-04-2022.

[18] OASIS. MQTT: The Standard for IoT Messaging. https://mqtt.org, 2022. Acceso el 01-04-2022.

[19] M Panda F, Lozupone. Different cognitive frailty models and health- and cognitive-related outcomes in older age: From epidemiology to prevention. *Journal of Alzheimer's Disease*, 62(3):993–1012, 2018.

[20] M Panda F, Lozupone. Searching for a frailty model to predict and prevent dementia. *Lancet Neurol*, 18(2):133–4, 2019.

[21] Michael Stonebraker. PostgreSQL: The World's Most Advanced Open Source Relational Database. https://www.postgresql.org, 1996. Acceso el 07-04-2022.

[22] Arai H Yamada M. Predictive value of frailty scores for healthy life expectancy in community-dwelling older japanese adults. *J Am Med Dir Assoc*, 16(11):1002.e7–11, 2015.

[23] Arai H Yamada M. Social frailty predicts incident disability and mortality among community-dwelling japanese older adults. *J Am Med Dir Assoc*, 19(12):1099–103, 2018.

Anexo A

Prototipo alternativo

En este capítulo se presenta el diseño del prototipo alternativo realizado para la aplicación móvil. En la Figura A.1 se muestra la ventana de inicio de sesión. En la Figura A.2 se puede observar la pantalla principal diseñada para el paciente. Como se puede apreciar, en este diseño alternativo, casi toda la funcionalidad de la aplicación es accesible desde esta ventana. En la Figura A.3 se muestra el diseño realizado para la selección del tipo de prueba a realizar. El cambio respecto al diseño original reside en la ventana emergente que se muestra. La Figura A.4 enseña el diseño realizado para el momento en el que un dispositivo es escaneado con éxito. En general, se puede apreciar como este diseño alternativo destacaba un mayor énfasis en las ventanas y alertas emergentes.

Figura A.1: Login alternativo.

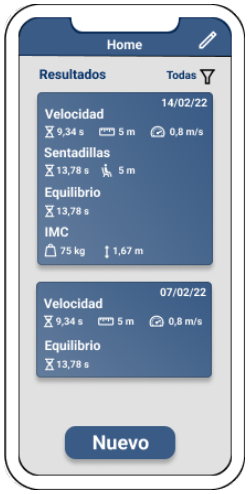

Figura A.2: Resultados generales.

Figura A.3: Seleccionar prueba.

Figura A.4: Éxito al escanear.

Respecto a la contestación de cuestionarios en las Figuras A.5, A.6, A.7 y A.8 se puede observar el diseño de las ventanas necesarias para la respuesta de los cuestionarios.

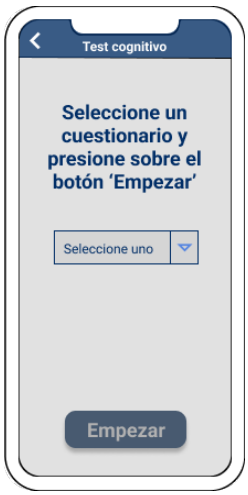

Figura A.5: Seleccionar cuestionario.

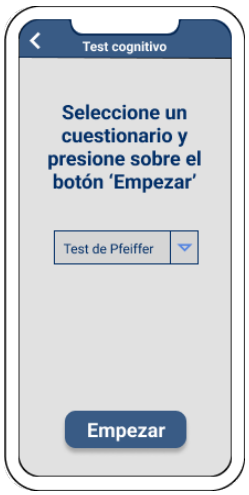

Figura A.6: Cuestionario seleccionado.

Figura A.7: Cuestionario de ejemplo.

Figura A.8: Cuestionario finalizado.

Por último, en las Figuras A.9, A.10, A.11 y A.12 se presentan los diseños realizados para permitir al paciente la inserción de datos manuales.

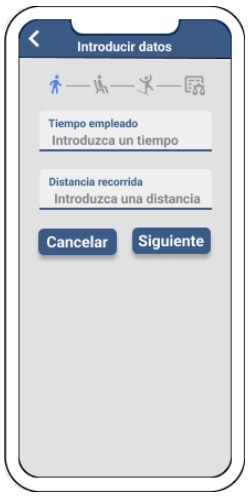

Figura A.9: Introducir datos de la prueba de velocidad.

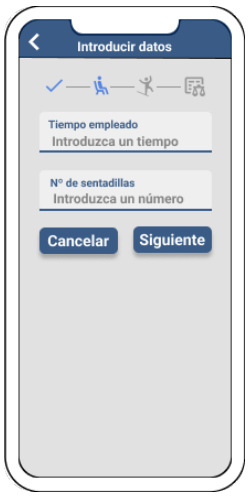

Figura A.10: Introducir datos de la prueba de sentadillas.

Figura A.11: Introducir datos de la prueba de equilibrio.

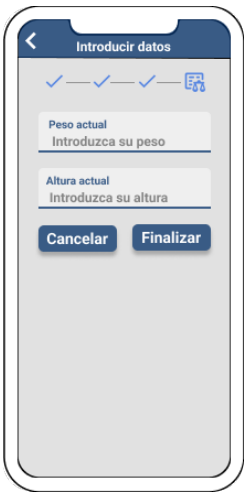

Figura A.12: Introducir datos IMC.

Anexo B

Especificación del resto de historias de usuario

En este capítulo se presenta la especificación de todas las historias de usuario definidas en el proyecto, exceptuando la HU06 ya descrita en el Capítulo 3.

HU01

Descripción: Como paciente quiero validar mi usuario mediante el DNI para acceder a la aplicación móvil.

Escenario 1

Given El profesional clínico ha registrado al paciente en el sistema.
When El paciente introduce su DNI correctamente.
Then La aplicación redirige al paciente a la ventana principal.

Escenario 2

Given El profesional clínico ha registrado al paciente en el sistema.
When El paciente introduce un texto que no se corresponde con el formato de DNI.
Then La aplicación muestra un mensaje de error al paciente indicándole que el texto introducido no se corresponde con el formato de un DNI.

Escenario 3

Given El profesional clínico no ha registrado al paciente en el sistema.
When El paciente introduce su DNI e intenta acceder a la aplicación.
Then La aplicación muestra un mensaje de error al paciente indicándole que no está registrado.

HU02

Descripción: Como paciente quiero ser identificado en la aplicación con la cámara mediante un código QR para facilitar el acceso al dispositivo.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación y el sistema está disponible.
When El paciente escanea el código QR de una de las cámaras.
Then La aplicación muestra un mensaje de éxito en la comunicación y le indica al paciente que debe seguir las instrucciones que se muestran en la aplicación de la cámara.

Escenario 2

Given El paciente se encuentra en el centro médico y ha iniciado sesión en la aplicación.
When El paciente escanea el código QR de una de las cámaras y el sistema no está disponible.
Then La aplicación muestra un mensaje indicando al paciente que el servicio está temporalmente no disponible.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación.
When El paciente escanea un código QR ajeno al sistema FTAS.
Then La aplicación muestra un mensaje de error indicando al paciente que el código escaneado no pertenece a ningún dispositivo conocido por el sistema.

HU03

Descripción: Como paciente quiero ser capaz de introducir manualmente los datos obtenidos al realizar las pruebas físicas para tener un forma alternativa de registrar los datos obtenidos.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación y desea introducir los datos de la prueba de velocidad.
AND El sistema está disponible.
When El paciente introduce un valor de 10 segundos para el tiempo empleado y un valor de 3 metros para la distancia recorrida.

Then La aplicación envía los datos al servidor y muestra al paciente un mensaje para indicar que los datos han sido almacenados con éxito.

Escenario 2

Given El paciente ha iniciado sesión en la aplicación y desea introducir los datos de la prueba de sentadillas.
AND El sistema está disponible.
When El paciente introduce un valor de 12 segundos para el tiempo empleado y un valor de 5 para indicar el número de sentadillas realizadas.
Then La aplicación envía los datos al servidor y muestra al paciente un mensaje para indicar que los datos han sido almacenados con éxito.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación y desea introducir los datos de la prueba de equilibrio.
AND El sistema está disponible.
When El paciente introduce un valor de 5 segundos para el tiempo en equilibrio.
Then La aplicación envía los datos al servidor y muestra al paciente un mensaje para indicar que los datos han sido almacenados con éxito.

Escenario 4

Given El paciente ha iniciado sesión en la aplicación y desea introducir los datos de la prueba de equilibrio.
AND El sistema no está disponible.
When El paciente introduce un valor de 5 segundos para el tiempo en equilibrio.
Then La aplicación intenta enviar los datos al servidor y tras el fallo en el envío, muestra al paciente un mensaje para indicar que los datos no han podido ser almacenados y le aconseja intentarlo de nuevo.

Escenario 5

Given El paciente ha iniciado sesión en la aplicación y desea introducir los datos de la prueba de equilibrio.
AND El sistema está disponible.
When El paciente intenta introducir un dato no numérico.
Then La aplicación muestra un mensaje al paciente indicando que el valor debe ser un número.

HU04

Descripción: Como paciente quiero ser capaz de introducir manualmente datos generales sobre mi salud, como el peso, la altura y la fuerza de agarre para poder generar datos que ayuden a diagnosticar mi estado de fragilidad.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación y desea introducir su peso en el sistema.
AND El sistema está disponible.
When El paciente selecciona la opción correspondiente al peso e introduce un valor de 78 kilogramos.
Then La aplicación envía el dato al servidor y muestra al paciente un mensaje para indicar que ha sido almacenado con éxito.

Escenario 2

Given El paciente ha iniciado sesión en la aplicación y desea introducir su altura en el sistema.
AND El sistema está disponible.
When El paciente selecciona la opción correspondiente a la altura e introduce un valor de 176 centímetros.
Then La aplicación envía el dato al servidor y muestra al paciente un mensaje para indicar que ha sido almacenado con éxito.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación y desea introducir su fuerza de agarre en el sistema.
AND El sistema está disponible.
When El paciente selecciona la opción correspondiente a la fuerza de agarre e introduce un valor de 10 kilogramos.
Then La aplicación envía el dato al servidor y muestra al paciente un mensaje para indicar que ha sido almacenado con éxito.

Escenario 4

Given El paciente ha iniciado sesión en la aplicación y desea introducir su altura en el sistema.
AND El sistema no está disponible.
When El paciente selecciona la opción correspondiente a la altura e introduce un valor de 176 centímetros.
Then La aplicación intenta enviar el dato al servidor y tras el fallo en el envío, muestra al paciente un mensaje para indicar que el dato no ha podido ser almacenado y le aconseja

intentarlo de nuevo.

HU05

Descripción: Como paciente quiero ver los cuestionarios disponibles en el sistema para poder seleccionar uno para responder.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación y ha accedido a la sección correspondiente para responder cuestionarios.
AND El sistema está disponible y existen 2 cuestionarios disponibles para el paciente.
When El paciente selecciona la opción correspondiente para responder un cuestionario.
Then La aplicación abre una nueva ventana y muestra una lista con 2 cuestionarios disponibles para el paciente por orden de inserción en la base de datos.

Escenario 2

Given El paciente ha iniciado sesión en la aplicación y ha accedido a la sección correspondiente para responder cuestionarios.
AND El sistema está disponible y no existen cuestionarios disponibles para el paciente.
When El paciente selecciona la opción correspondiente para responder un cuestionario.
Then La aplicación abre una nueva ventana con un mensaje indicando que no existen cuestionarios disponibles actualmente.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación y ha accedido a la sección correspondiente para responder cuestionarios.
AND El sistema no está disponible.
When El paciente selecciona la opción correspondiente para responder un cuestionario.
Then La aplicación muestra un indicador de carga durante un máximo de 5 segundos y muestra un mensaje al paciente indicando el fallo temporal en el servicio.

HU07

Descripción: Como paciente quiero que todos los datos introducidos manualmente sean enviados al servidor para su posterior almacenamiento.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación, ha seleccionado introducir su nuevo peso y ha introducido un valor de 78 kilogramos.

When El paciente presiona sobre el botón para enviar los datos.

Then La aplicación realiza una petición HTTP POST al servidor a la ruta correspondiente para guardar el peso del paciente.

Escenario 2

Given El paciente ha iniciado sesión en la aplicación, ha seleccionado introducir su nuevo peso y ha introducido un valor de 78 kilogramos. Además, ha enviado los datos.

AND El sistema está disponible.

When El sistema valida el formato de los datos y tras comprobar que no hay errores se envía una respuesta con el código 201.

Then La aplicación recibe la respuesta del servidor y muestra un mensaje indicando al paciente el éxito en el envío de datos.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación, ha seleccionado introducir su nuevo peso y ha introducido un valor de 78 kilogramos.

AND El paciente ha presionado sobre el botón para enviar los datos. El sistema no está disponible.

When La aplicación recibe una respuesta con el código de error 503 debido a que el servicio no está disponible.

Then La aplicación muestra un mensaje indicando al paciente el fallo temporal del servidor.

HU08

Descripción: Como paciente quiero consultar el tiempo medio empleado en realizar cada una de las pruebas físicas para poder ver mi evolución en el tiempo.

Escenario 1

Given El paciente ha iniciado sesión en la aplicación y tiene diversos registros almacenados en el sistema de las tres pruebas físicas.

AND El sistema está disponible.

When El paciente selecciona la opción correspondiente para ver sus resultados en las pruebas de velocidad.

Then La aplicación abre una nueva ventana y muestra un gráfico mostrando la evolución del tiempo empleado en realizar la prueba de velocidad.

Escenario 2

Given El paciente ha iniciado sesión en la aplicación y tiene diversos registros almacenados en el sistema de las tres pruebas físicas.

AND El sistema está disponible.

When El paciente selecciona la opción correspondiente para ver sus resultados en las pruebas de sentadillas.

Then La aplicación abre una nueva ventana y muestra un gráfico mostrando la evolución del tiempo empleado en realizar la prueba de sentadillas.

Escenario 3

Given El paciente ha iniciado sesión en la aplicación y tiene diversos registros almacenados en el sistema de las tres pruebas físicas.

AND El sistema está disponible.

When El paciente selecciona la opción correspondiente para ver sus resultados en las pruebas de equilibrio.

Then La aplicación abre una nueva ventana y muestra un gráfico mostrando la evolución del tiempo empleado en realizar la prueba de equilibrio.

Escenario 4

Given El paciente ha iniciado sesión en la aplicación y no tiene ningún registro de la prueba de velocidad.

AND El sistema está disponible.

When El paciente selecciona la opción correspondiente para ver sus resultados en las pruebas de velocidad.

Then La aplicación abre una nueva ventana y muestra un texto indicando que no existen registros en el sistema.

Escenario 5

Given El paciente ha iniciado sesión en la aplicación y no tiene ningún registro de la prueba de velocidad.

AND El sistema no está disponible.

When El paciente selecciona la opción correspondiente para ver sus resultados en las pruebas de velocidad.

Then La aplicación muestra un mensaje indicando el fallo temporal del servidor.

Anexo C

Cuestionarios cognitivos

En este anexo se presentan dos ejemplos de cuestionarios cognitivos utilizados en la aplicación. El primer cuestionario es utilizado para detectar la fragilidad social, mientras que el segundo cuestionario está destinado a detectar la fragilidad psicológica. Estos cuestionarios han sido proporcionados por la empresa.

C.1. Fragilidad Social (Escala Garre-Olmo)

A continuación, voy a hacerle 6 preguntas sobre sus relaciones sociales. Recuerde que no existen respuestas correctas o incorrectas. Tómese el tiempo que necesite para responder.

1. ¿Con quién vive usualmente? Respuesta abierta
2. ¿Tiene familiares o amigos a los que pudiera pedir ayuda si la necesitara? Sí / No
3. ¿Con qué frecuencia se reúne o habla con sus familiares más cercanos? Respuesta abierta
4. ¿Con qué frecuencia se reúne o habla con sus amigos y/o vecinos? Respuesta abierta
5. ¿Hay alguien especial (una pareja, amigo, familiar y/o vecino) en quién confiar y hablar sobre asuntos personales y sus sentimientos? Sí / No
6. En los últimos 3 meses, ¿ha conseguido ayuda de otras personas cuando la ha necesitado para comprar, preparar alimentos, limpiar la casa, planchar u otras actividades personales? Sí / No

C.2. Fragilidad psicológica (Escala Pfeiffer)

A continuación, voy a realizarle 10 preguntas que valoran varias funciones mentales cómo la orientación, la memoria y el cálculo. No se preocupe si no sabe la respuesta a alguna pregunta. Tómese el tiempo que necesite para responder.

1. ¿Qué fecha es hoy? (día, mes, año)
2. ¿Qué día de la semana es hoy?
3. ¿Cuál es su número de teléfono / móvil?
4. ¿Dónde estamos ahora?
5. ¿Qué edad tiene?
6. ¿Cuándo nació? (día, mes, año)
7. ¿Quién es el presidente del gobierno?
8. ¿Quién era el anterior presidente de gobierno?
9. ¿Cuál era el primer apellido de su madre?
10. Le voy a pedir que reste de 3 en 3 desde 20 (cualquier error hace la respuesta errónea).