Grado en Ingeniería Informática

Trabajo de Final de Grado

Digitalización y automatización de pruebas físicas para la detección de fragilidad en ancianos

Autor:
Miguel Pardo Navarro

Supervisor:
Sergio Aguado González
Tutor académico:
Reyes Grangel Seguer

Fecha de lectura: 14 de Julio de 2022
Curso académico 2021/2022

Resumen

En este documento se presenta el Trabajo Final de Grado realizado por Miguel Pardo Navarro tras la estancia en prácticas en la empresa Soluciones Cuatroochenta.

El proyecto consiste en el desarrollo de un sistema capaz de guiar y asistir a personas de edad avanzada para realizar tres pruebas físicas para la detección de fragilidad mediante una cámara 3D. La fragilidad hace referencia al estado en el que una persona presenta un mayor riesgo ante eventos adversos para la salud como caídas, delirios o discapacidades. El objetivo es digitalizar y automatizar la toma de resultados de las pruebas para poder realizar un seguimiento sobre el estado de salud de los pacientes a lo largo del tiempo.

La aplicación se ha implementado con el lenguaje de programación C# y con el *framework* WPF (*Windows Presentation Fundation*) y, además, forma parte de un sistema denominado *Frailty Tests Assistance System* (FTAS) o Sistema de Asistencia a las Pruebas de Fragilidad. Para acoplarse con los distintos subsistemas que forman FTAS, la aplicación hace uso de la tecnología MQTT y del protocolo de comunicación HTTP.

El desarrollo del proyecto se ha completado siguiendo la metodología ágil Scrum con cinco *sprints* de una duración de dos semanas cada uno.

Palabras clave

Fragilidad física, cámara 3D, Kinect V2, metodología Scrum, MQTT, WPF, C#.

Keywords

Physical frailty, 3D camera, Kinect V2, Scrum methodology, MQTT, WPF, C#.

Índice general

1. Introducción **11**

1.1. Contexto y motivación del proyecto 11

1.2. Objetivos del proyecto 13

1.3. Descripción del proyecto 13

1.4. Estructura de la memoria 16

2. Planificación del proyecto **17**

2.1. Metodología 17

2.2. Planificación 18

2.3. Estimación de recursos y costes del proyecto 19

2.3.1. Recursos del proyecto 19

2.3.2. Costes del proyecto 20

2.4. Gestión de riesgos 21

2.5. Seguimiento del proyecto 25

2.5.1. Primer *Sprint* 26

2.5.2. Segundo *Sprint* 27

2.5.3. Tercer *Sprint* 29

2.5.4. Cuarto *Sprint* 30

2.5.5. Quinto *Sprint* 31

3. Análisis y diseño del sistema **33**

3.1. Análisis del sistema 33

3.2. Diseño de la arquitectura del sistema 40

3.3. Diseño de la interfaz 45

3.3.1. Prototipado 46

3.3.2. Diseño final 49

4. Implementación y pruebas **53**

4.1. Detalles de implementación 53

4.1.1. Estructura y organización del proyecto 53

4.1.2. Patrones de diseño 55

4.1.3. Algoritmos para la detección de fragilidad 56

4.1.4. Cálculo de los pasos durante la prueba de velocidad de la marcha 64

4.1.5. Cálculo de la altura durante la prueba de sentadillas 65

4.1.6. Comunicación y acoplamiento de los subsistemas 66

4.2. Verificación y validación 69

4.2.1. Verificación de las pruebas físicas 69

4.2.2. Pruebas de sistema 69

4.2.3. Análisis estático del código 69

5. Conclusiones **71**

Índice de figuras

1.1. Subsistemas que componen FTAS 12

1.2. Articulaciones detectadas por la cámara Kinect 15

2.1. *Product backlog* inicial del proyecto 19

2.2. *Backlog* del primer *sprint* del proyecto 26

2.3. Gráficos *Burndown* del primer *sprint* 27

2.4. *Backlog* del segundo *sprint* del proyecto 27

2.5. Descripción y *Definition of Done* de HU02 27

2.6. Definición de los escenarios de HU02 28

2.7. Gráficos *Burndown* del segundo *sprint* 28

2.8. *Backlog* del tercer *sprint* del proyecto 29

2.9. Gráficos *Burndown* del tercer *sprint* 30

2.10. *Backlog* del cuarto *sprint* del proyecto 30

2.11. Gráficos *Burndown* del cuarto *sprint* 31

2.12. *Backlog* del quinto y último *sprint* del proyecto 31

2.13. Gráficos *Burndown* del quinto y último *sprint* 32

2.14. *Velocity report* del proyecto 32

3.1. Diagrama de casos de uso del proyecto 33

3.2. Modelo conceptual del sistema 37

3.3. Diagrama de actividades del proyecto 39
3.4. Modelo-Vista-Controlador 40
3.5. *Code behind* en WPF 40
3.6. Diagrama de clases final del proyecto 41
3.7. DC bloque 1. Estructuras básicas y contenedores de texto 42
3.8. DC bloque 2. Controladores de la vista y gestores principales 43
3.9. DC bloque 3. Lógica de las pruebas físicas 44
3.10. DC bloque 4. Modelo y lógica de los resultados 45
3.11. Prototipos de las pantallas de inicio y del menú principal 46
3.12. Prototipos de las pantallas de información previa a cada prueba 46
3.13. Prototipos de las pantallas durante la prueba de velocidad de la marcha 47
3.14. Prototipos de las pantallas durante el transcurso de la prueba de sentadillas . . . 48
3.15. Prototipos de las pantallas durante el transcurso de la prueba de equilibrio . . . 48
3.16. Pantallas de inicio y del menú principal 49
3.17. Pantallas de información previa a cada prueba 49
3.18. Pantallas durante el transcurso de la prueba de velocidad de la marcha 50
3.19. Pantallas durante el transcurso de la prueba de sentadillas 50
3.20. Pantallas durante el transcurso de la prueba de equilibrio 51

4.1. Estructura y organización del proyecto 54
4.2. Envío de los resultados de las pruebas con el patrón DTO 56
4.3. Digitalización de la prueba de velocidad de la marcha 59
4.4. Articulaciones que proporciona el sensor Kinect 59
4.5. Evaluación de la profundidad de los tobillos durante el paseo 64
4.6. Evaluación de la altura durante la prueba de sentadillas 65

4.7. Primera fase del proceso de identificación con el código QR 66

4.8. Segunda fase del proceso de identificación con el código QR 67

4.9. Tercera fase del proceso de identificación con el código QR 67

4.10. Cuarta fase del proceso de identificación con el código QR 68

4.11. Quinta fase del proceso de identificación con el código QR 68

4.12. Métricas de calidad y complejidad del código implementado 70

Índice de cuadros

1.1. Puntuación de la prueba de velocidad de la marcha 14

1.2. Puntuación de la prueba de sentadillas 14

1.3. Puntuación de la prueba de equilibrio en posición de tándem 15

2.1. Pila inicial del proyecto 18

2.2. Costes del proyecto 20

2.3. Identificación de los riesgos del proyecto 21

2.4. Análisis, prevención y contingencia del riesgo R01 21

2.5. Análisis, prevención y contingencia del riesgo R02 22

2.6. Análisis, prevención y contingencia del riesgo R03 22

2.7. Análisis, prevención y contingencia del riesgo R04 23

2.8. Análisis, prevención y contingencia del riesgo R05 24

2.9. Análisis, prevención y contingencia del riesgo R06 24

2.10. Análisis, prevención y contingencia del riesgo R07 25

2.11. Análisis, prevención y contingencia del riesgo R08 25

2.12. *Sprints* realizados a lo largo del proyecto 26

3.1. Historias de usuario del proyecto 34

3.2. Especificación de la historia de usuario HU01 35

3.3. Requisito de datos RD01 36

3.4. Requisito de datos RD02 36

3.5. Requisito de datos RD03 36

3.6. Requisito de datos RD04 37

3.7. Requisito de calidad RC01 38

3.8. Requisito de calidad RC02 38

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Según el Instituto Nacional de Estadística (INE), España se encuentra en un momento de envejecimiento demográfico. En las próximas décadas, se espera un incremento de más del 10 por ciento de la proporción de personas con edades de 70 años o más. De hecho, si se mantiene la tendencia demográfica actual, el grupo de edad más numeroso en el año 2050 serán los nacidos en los años 1970-1979 con edades entre 70 y 79 años [8].

Esta proyección reaviva la necesidad de desarrollar soluciones centradas en mejorar la calidad de vida de las personas de edad avanzada. Por ello nace la línea de trabajo e investigación para detectar y prevenir la fragilidad en ancianos y retrasar la aparición de dicho estado en el tiempo. En el ámbito de la salud, la fragilidad es el estado en el que una persona se ve más gravemente afectada y presenta un mayor riesgo ante eventos adversos para su salud como por ejemplo caídas, delirios o discapacidades [3].

Existen multitud de formas de evaluar la fragilidad física en ancianos como por ejemplo el test de velocidad de la marcha, que consiste en medir el tiempo que requiere una persona en recorrer un distancia de tres o cuatro metros caminando a velocidad normal. Tradicionalmente, este tipo de pruebas son guiadas y controladas por un profesional del ámbito de la salud, generalmente un fisioterapeuta geriátrico, que recoge las marcas de forma manual con un cronómetro.

Dado lo expuesto anteriormente, la empresa Soluciones Cuatroochenta ha propuesto desarrollar un innovador sistema para digitalizar y automatizar la detección de la fragilidad en personas de edad avanzada. Cuatroochenta es una empresa de base tecnológica nacida en 2011 y especializada en el desarrollo de soluciones digitales *cloud* y en ciberseguridad. En los últimos años, la compañía ha impulsado el desarrollo de soluciones relacionadas con la salud y el bienestar, siendo este proyecto parte de la reciente línea de trabajo de la empresa [4].

El sistema informático que plantea Cuatroochenta se ha denominado *Frailty Tests Assistance System* (FTAS) o Sistema de Asistencia a las Pruebas de Fragilidad que, a grandes rasgos, se puede dividir en tres subsistemas presentados en distintas plataformas y con funcionalidades

claramente diferenciadas.

El primer subsistema es una aplicación web que pretende permitir a los médicos o profesionales de la salud gestionar y consultar la información médica de sus pacientes. En concreto, el médico debe poder consultar las pruebas físicas realizadas por cada paciente, su estado de fragilidad y su evolución según los resultados de las pruebas a lo largo del tiempo. Además, el médico debe poder dar de alta a nuevos pacientes e introducir manualmente nuevos resultados de pruebas realizadas por alguno de sus pacientes. Ya que el ordenador es el entorno habitual de trabajo de un profesional de la salud, toda esta funcionalidad debe llevarse a cabo a través de una aplicación web.

El segundo subsistema se presenta como una aplicación móvil que debe permitir a las personas de edad avanzada consultar cierta información de las pruebas realizadas e introducir de forma manual nuevos resultados. En este segundo subsistema también se incluye la funcionalidad de añadir valores de peso y altura ya que puede tratarse de información relevante para el profesional de la salud. Toda la información que registre el paciente podrá ser consultada por el médico asociado desde su plataforma correspondiente.

Finalmente, el tercer subsistema debe permitir a los pacientes realizar un conjunto de pruebas físicas de forma autónoma sin necesidad de disponer de un especialista para guiar o asistir a la persona de edad avanzada durante el transcurso de la prueba. La plataforma en la que se realiza este conjunto de pruebas para la detección de fragilidad es una aplicación de escritorio para sistemas Windows con hardware específico. Dicha aplicación de escritorio requiere una cámara Microsoft Kinect v2 [9] que permite el reconocimiento y seguimiento del cuerpo humano en tiempo real para, de esta forma, digitalizar y automatizar las pruebas.

En la Figura 1.1 se muestra un diagrama de los subsistemas descritos que componen FTAS desde una perspectiva de alto nivel.

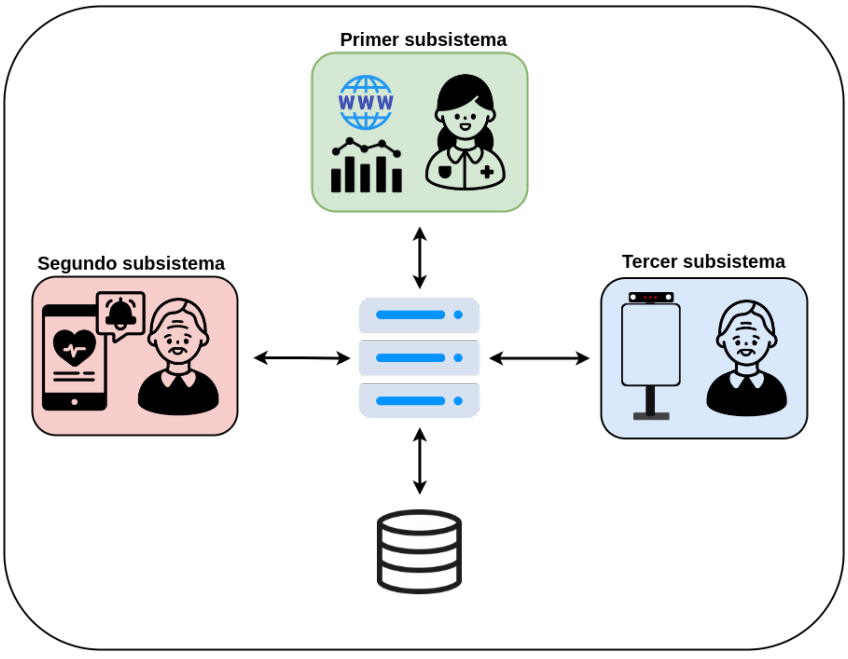


Figura 1.1: Subsistemas que componen FTAS

Este proyecto gira entorno al desarrollo del último subsistema, quedando fuera del alcance todo lo relacionado con la aplicación móvil del paciente y con la aplicación web del profesional de la salud. Es decir, este proyecto se centra exclusivamente en la digitalización y automatización de tres pruebas físicas para la detección de fragilidad con la cámara Microsoft Kinect v2 y en el desarrollo de la aplicación de escritorio para guiar y asistir al anciano que lleva a cabo las pruebas.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es el desarrollo de una aplicación que permita digitalizar y automatizar tres pruebas físicas para la detección de fragilidad en personas de edad avanzada. Este objetivo principal se puede desglosar en los siguientes subobjetivos:

- Ofrecer la posibilidad de realizar tres pruebas físicas para la detección de fragilidad de forma autónoma sin necesidad de disponer de un profesional de la salud durante el transcurso de la prueba.
- Digitalizar el proceso de la toma de datos y asegurar el registro y persistencia de los resultados.
- Proporcionar un método alternativo al tradicional, y de mayor precisión, para la estimación del tiempo que ha requerido realizar cada prueba.
- Reducir la brecha digital con el desarrollo de un sistema que acerque a los mayores las nuevas tecnologías.
- Facilitar el seguimiento del estado de salud para tomar medidas preventivas o correctivas en caso de detectar estados de fragilidad o prefragilidad y mejorar la calidad de vida de este grupo de la población.
- Proporcionar información relevante para analizar la evolución del estado físico de los pacientes a lo largo del tiempo y, de esta forma, poder llevar un control más detallado sobre la salud de los ancianos.

1.3. Descripción del proyecto

Una vez descrito el contexto del subsistema, a continuación se detalla el alcance funcional, organizativo e informático del producto que se describe en este documento.

En relación al **alcance funcional**:

- El sistema debe permitir a los pacientes identificarse mediante su DNI o, de forma complementaria, mediante el escaneo de un código QR para facilitar la interacción con el sistema.

- El sistema debe permitir a los pacientes realizar tres pruebas físicas para la detección de fragilidad, digitalizar la toma de datos y registrar el tiempo que requiere al paciente realizar cada una de las pruebas. En concreto, las pruebas físicas que el sistema debe permitir completar son:

  - La prueba de velocidad de la marcha, que consiste en caminar una distancia de tres metros a velocidad normal.
  - La prueba de sentadillas, que consiste en levantarse y sentarse cinco veces de una silla.
  - La prueba de equilibrio, que consiste en colocar los pies en línea (posición de tándem) y mantener el equilibrio el tiempo que sea posible o hasta superar los diez segundos.

- El sistema debe asignar una puntuación en base al tiempo que le ha requerido al paciente completar cada prueba según lo expresado en el Cuadro 1.1, 1.2 y 1.3.

- El sistema debe permitir realizar las tres pruebas anteriores de forma continuada y con la menor interacción posible con el sistema.

La digitalización de la toma de datos implica obtener información relevante durante el transcurso del test que pueda ser necesaria para su posterior análisis y diagnóstico por parte del profesional de la salud, como por ejemplo el número de pasos en la prueba de velocidad de la marcha o la amplitud de movimiento de las sentadillas en la segunda prueba.

Puntuación	**Tiempo**
0	Incapaz de realizar la prueba
1	Más de 6,52 segundos
2	Entre 4,66 y 6,52 segundos
3	Entre 3,62 y 4,65 segundos
4	Menos de 3,62 segundos

Cuadro 1.1: Puntuación de la prueba de velocidad de la marcha

Puntuación	**Tiempo**
0	Incapaz de completar la prueba o más de 60 segundos
1	Más de 16,70 segundos
2	Entre 13,70 y 16,69 segundos
3	Entre 11,20 y 13,69 segundos
4	Menos de 11,20 segundos

Cuadro 1.2: Puntuación de la prueba de sentadillas

Puntuación	Tiempo
0	Incapaz de mantener el equilibrio o menos de 3 segundos
1	Entre 3 y 9,99 segundos
2	10 segundos o más

Cuadro 1.3: Puntuación de la prueba de equilibrio en posición de tándem

En relación al **alcance organizativo**, la aplicación implementada va dirigida a las personas de edad avanzada que se encuentran en residencias, o centros de salud, que quieran disponer de esta tecnología para realizar un seguimiento del estado de salud de sus pacientes.

Finalmente, a continuación se detalla el **alcance informático** y los aspectos de carácter tecnológico más relevantes que abarca el desarrollo del producto.

En primer lugar, el proyecto incluye la cámara Microsoft Kinect v2. Se trata de un sensor que estima la ubicación en tres dimensiones de las articulaciones del cuerpo humano en tiempo real a partir de imágenes en dos dimensiones y con datos de profundidad mediante algoritmos de aprendizaje automático. Hoy día, este dispositivo desarrollado por Microsoft es ampliamente utilizado en el ámbito de la investigación por haber demostrado gran precisión y fiabilidad a la hora de estimar y calcular la posición y cinemática del cuerpo humano [2]. El sensor proporciona la posición de 25 articulaciones de hasta seis cuerpos al mismo tiempo en distancias de 0,5 a 4,5 metros respecto a la cámara. La Figura 1.2 muestra visualmente los puntos que el dispositivo reconoce como articulaciones y que, posteriormente, se pueden dibujar junto con la unión de los mismos representando lo que formaría el esqueleto simplificado del cuerpo humano. La cámara también proporciona imagen de vídeo de alta calidad, reconocimiento y seguimiento del rostro, funciones de infrarrojos y captura de audio entre otros.

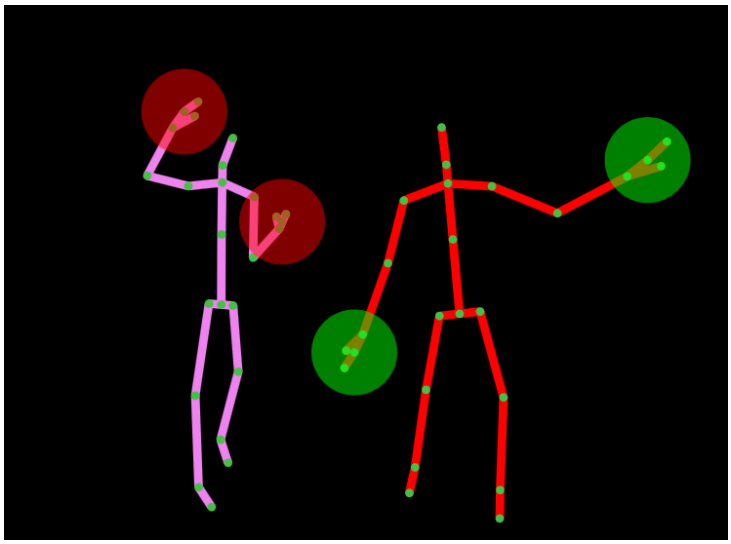


Figura 1.2: Articulaciones detectadas por la cámara Kinect

Obtener y tratar la información que proporciona la cámara Microsoft Kinect v2 requiere el desarrollo de software empleando el lenguaje de programación C#. Para el desarrollo de la aplicación se ha propuesto la tecnología denominada Windows Presentation Foundation (WPF)

ya que permite desarrollar aplicaciones de escritorio para sistemas Windows en C# con unos acabados modernos y visualmente atractivos. Este *framework* permite el desarrollo de aplicaciones siguiendo el conocido patrón arquitectónico Modelo-Vista-Controlador (MVC), o alguno de sus derivados como el Modelo-Vista-VistaModelo (MVVM) [17].

Para validar que se trata de un paciente registrado en el sistema y para guardar los resultados de las pruebas, es necesario interactuar con un servidor mediante el protocolo de comunicación HTTP. El desarrollo e implementación del servidor en cuestión queda fuera del alcance del proyecto.

Por otro lado, para realizar la identificación del usuario mediante el escaneo del código QR se utiliza MQTT. Esta tecnología permite establecer una comunicación bidireccional entre el dispositivo móvil (desde donde se escanea el código QR) y el sistema que aloja el sensor Kinect a través del servidor. Se trata de un protocolo de comunicación sencillo y ligero basado en la arquitectura de publicación/suscripción que permite intercambiar información en redes con alta latencia o bajo ancho de banda. Actualmente, MQTT está definido como un estándar OASIS para la conectividad IoT [11].

1.4. Estructura de la memoria

Para terminar con este primer capítulo, a continuación se presenta la estructura de la memoria. En el Capítulo 2 se detalla la metodología de desarrollo utilizada, así como también el seguimiento y control realizado durante el transcurso del proyecto. En el Capítulo 3 se muestra el análisis del sistema, incluyendo la especificación de requisitos, el diseño de la arquitectura y el diseño de la interfaz gráfica de usuario. En el Capítulo 4 se presentan los detalles de la implementación y cómo el sistema ha sido verificado y validado. Finalmente, el Capítulo 5 presenta las conclusiones en lo que se refiere al ámbito formativo, profesional y personal de este proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Para el desarrollo del proyecto, se ha seguido la metodología de desarrollo ágil Scrum [14]. Esta decisión se debe principalmente a que los requisitos no eran claros ni llegaban a estar completamente definidos. Además, las tecnologías utilizadas eran desconocidas y resultaba difícil estimar el coste de las tareas y su viabilidad.

La duración definida para los *sprints* fue de dos semanas o diez días laborables. Dado que las horas de trabajo acordadas para la estancia en prácticas fueron de 30 horas a la semana (seis horas al día), la duración de un *sprint* suponía un trabajo aproximado de 60 horas.

Por otra parte, las reuniones diarias se llevaron a cabo tres días a la semana: lunes, miércoles y jueves. En estas reuniones participaba, entre otros, Arturo Gascó, el desarrollador de la aplicación móvil de los pacientes (el segundo subsistema) y del servidor con el que interactúa la aplicación definida en este proyecto. La comunicación con Arturo durante las reuniones ha sido un factor clave para llevar a cabo el proyecto de forma satisfactoria y resolver los problemas asociados con la integración de los subsistemas. En caso de inicio de un nuevo *sprint*, el primer lunes se realizaba una reunión de planificación para abordar los principales objetivos y seleccionar las historias de usuario y tareas a llevar a cabo. Finalmente, al término de cada *sprint* se llevaba a cabo una reunión de revisión para comentar las mejoras del producto y atender la evolución del proyecto.

En lo que se refiere a los distintos roles que define la metodología Scrum, tanto el rol de propietario del producto, o *product owner* en inglés, como el rol de *scrum master* lo ha tomado Sergio Aguado siendo el principal interesado y supervisor del proyecto.

2.2. Planificación

El Cuadro 2.1 muestra la primera versión de la pila del producto donde se presentan las historias de usuario definidas inicialmente para el desarrollo del proyecto. En dicha tabla aparecen las historias de usuario estimadas en puntos de historia y ordenadas por la prioridad establecida por el propietario del producto.

Id.	**Descripción**	**Puntos de historia**
HU01	Como paciente quiero poder realizar la prueba de velocidad de la marcha sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad.	21
HU02	Como paciente quiero poder realizar la prueba de levantarse y sentarse de la silla sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad.	13
HU03	Como paciente quiero poder realizar la prueba de equilibrio sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad.	34
HU04	Como paciente quiero poder identificarme en la aplicación mediante el DNI para realizar las pruebas físicas para la detección de fragilidad física.	8
HU05	Como paciente quiero poder identificarme en la aplicación mediante el escaneo de un código QR presentado en la pantalla para facilitar la interacción con el sistema.	21
HU06	Como paciente quiero que el sistema envíe los resultados de las pruebas a un servidor para que pueden ser guardados y consultados posteriormente.	8
HU07	Como paciente quiero que el sistema permita realizar la prueba de velocidad de la marcha, la de levantarse y sentarse de la silla y la de equilibrio de forma continuada y autónoma sin emplear el teclado o ratón para facilitar la interacción con el sistema.	21

Cuadro 2.1: Pila inicial del proyecto

Las historias de usuario detalladas en el Cuadro 2.1 también pueden apreciarse en la pila inicial del proyecto (o *product backlog*) registrada en Jira [1] y presentada en la Figura 2.1 junto con un grupo de tareas identificadas para llevar a cabo el proyecto. Dichas tareas tienen que ver con la realización de cursos de formación para conocer las nuevas tecnologías, la configuración inicial del proyecto y el diseño de la interfaz de usuario de la aplicación.

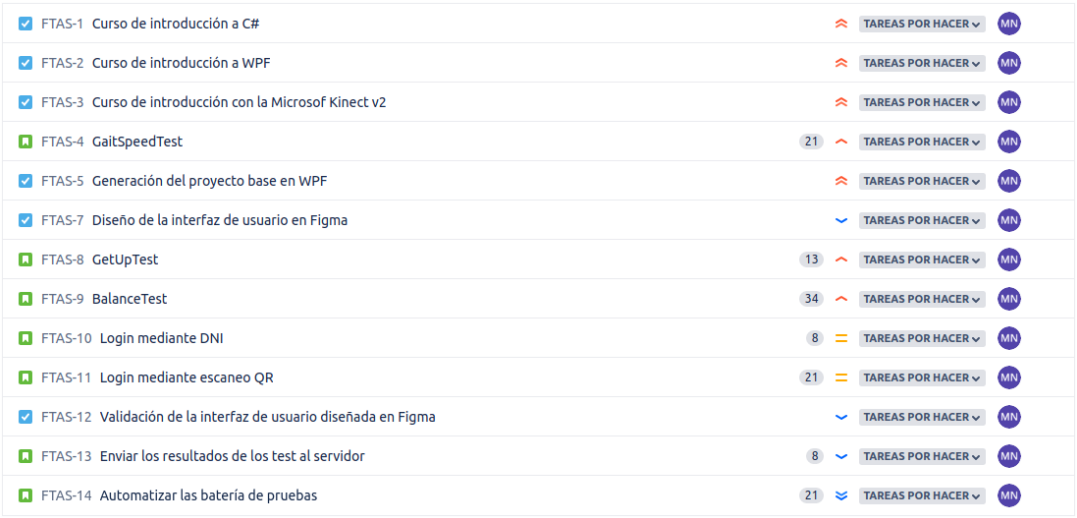


Figura 2.1: *Product backlog* inicial del proyecto

2.3. Estimación de recursos y costes del proyecto

En esta sección se detallan los aspectos más relevantes asociados a los costes y recursos necesarios para llevar a cabo el proyecto.

2.3.1. Recursos del proyecto

Por un lado, en lo que respecta a los recursos hardware, para desarrollar el proyecto ha sido necesario contar con la cámara Microsoft Kinect v2, un televisor Samsung, un teclado, un ratón y un computador DELL con 8GB de RAM, un procesador de 64 bits con cuatro núcleos con frecuencias de 3,2 GHz y con varios conectores USB 3.0.

Por otro lado, en lo que respecta a los recursos software, se ha utilizado el entorno de desarrollo Microsoft Visual Studio (versión *Community*) ya que proporciona facilidades a la hora de desarrollar aplicaciones WPF [10]. Para el diseño de las pantallas se propuso el uso de Figma, una aplicación Web que permite el desarrollo de *mockups* y prototipos de mayor fidelidad de forma sencilla e intuitiva [5]. Por otro lado, para facilitar las pruebas de comunicación mediante el protocolo HTTP con el servidor se ha utilizado Postman. Esta herramienta permite definir la cabecera y cuerpo de las peticiones HTTP y analizar las respuestas de dichas peticiones [12]. Por último, como ya se ha mencionado, también se utiliza Jira para realizar el seguimiento del proyecto y aplicar la metodología Scrum [1].

Finalmente, en lo que se refiere a los recursos humanos, la metodología empleada ha requerido asignar personal a los siguientes tres roles: propietario del producto, *scrum master* y desarrollador. Como se ha mencionado previamente, el rol de propietario del producto y el rol de *scrum master* lo ha tomado Sergio Aguado. El rol de desarrollador lo ha ejercido el autor de este documento.

2.3.2. Costes del proyecto

Expuestos los recursos necesarios para llevar a cabo el proyecto, el coste que requiere el desarrollo del sistema se detalla a continuación. En primer lugar, en lo que respecta a los recursos hardware, se debe tener en cuenta la cámara Kinect, el ordenador y los periféricos. En segundo lugar, en lo que respecta a los recursos software, los costes asociados son cero ya que todas las herramientas utilizadas son *open source* o de licencia gratuita. En tercer lugar, en relación a los recursos humanos, el salario medio de un programador en España es de 14,10 euros la hora [15] y, el coste de los recursos humanos asociados al desarrollo del proyecto se puede obtener a partir de las horas que implica la estancia en prácticas por el salario medio por hora de un programador en España:

$$CosteB = HorasEstancia \cdot SalarioMedio = 300 \cdot 14,10 = 4.230 €$$

Cabe destacar que el valor obtenido no supone el coste real que implica a la empresa mantener un trabajador. Es decir, el valor expresado equivale al salario bruto que recibe el trabajador y, por tanto, se deben añadir también 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 la siguiente expresión la que determina el valor real del coste asociado a los recursos humanos:

$$CosteH = CosteB + (0,3 \cdot CosteB) = 4.230 + (0,3 \cdot 4.230) = 5.499 €$$

Por último, también es necesario tener en cuenta otros gastos y costes indirectos como la luz, el alquiler del espacio de trabajo y material de oficina para el desarrollo del proyecto. El cómputo total y detalle de todos estos costes se muestra en el Cuadro 2.2.

Recurso	**Tipo de recurso**	**Coste en euros**
Microsoft Kinect V2	Hardware	260,00
Pantalla y soporte	Hardware	280,00
Ordenador	Hardware	350,00
Teclado y ratón	Hardware	30,00
Entornos de desarrollo u otros programas	Software	0,00
Desarrollador	Humano	5.499,00
Luz y agua	Indirecto	150,00
Espacio de trabajo	Indirecto	600,00
Material de oficina	Indirecto	200,00
Total	**-**	**7.369,00**

Cuadro 2.2: Costes del proyecto

2.4. Gestión de riesgos

La identificación y gestión de los riesgos supone un aspecto fundamental para asegurar el éxito del proyecto. En esta sección se presentan los principales riesgos identificados para los que, posteriormente, se presenta un plan de prevención y contingencia para mitigar su impacto en caso de que aparezcan. El Cuadro 2.3 muestra ocho riesgos identificados y el Cuadro 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10 y 2.11 los planes de prevención y contingencia para cada uno de ellos.

Id.	**Riesgo**	**Tipo de Riesgo**
R01	Fecha de entrega establecida e inamovible	Del proyecto
R02	Tecnologías desconocidas	Del proyecto
R03	Aspectos teóricos relacionados con la fragilidad física desconocidos	Del proyecto
R04	Diseño o implementación inadecuada de la interfaz gráfica	Del producto
R05	Diseño o implementación errónea de la lógica de las pruebas	Del producto
R06	Imposibilidad de acoplar los subsistemas	Del producto
R07	Falta de comunicación con el propietario del producto	Del proyecto
R08	Baja o ausencia de algún miembro del equipo	Del proyecto

Cuadro 2.3: Identificación de los riesgos del proyecto

Riesgo	**R01. Fecha de entrega establecida e inamovible**
Descripción	El proyecto se realiza durante la estancia en prácticas con un determinado número de horas finitas
Magnitud	Alta
Impacto	Se debe tener en cuenta la fecha de entrega durante todas las reuniones de planificación especialmente en lo que respecta a la priorización de tareas
Indicadores	No procede
Plan de prevención	Tener en cuenta durante todas las reuniones de planificación las prioridades del propietario del producto y atender la evolución del proyecto en base a los requisitos iniciales
Plan de contingencia	Realizar la priorización de requisitos con el propietario del producto y descartar aquellas de menor relevancia para cubrir las más prioritarias

Cuadro 2.4: Análisis, prevención y contingencia del riesgo R01

Riesgo	**R02. Tecnologías desconocidas**
Descripción	El programador desconoce todas las tecnologías propuestas para el desarrollo del proyecto
Magnitud	Alta
Impacto	Se debe tener en cuenta a la hora de estimar el coste de las tareas y contar con un periodo de formación
Indicadores	El desarrollador tiene poca solvencia frente a nuevos problemas y la implementación es más lenta de lo esperado
Plan de prevención	Dedicar un periodo de formación y familiarización con las nuevas tecnologías en el primer *sprint*
Plan de contingencia	Detallar los problemas detectados en las reuniones diarias y contactar con miembros de la empresa que conozcan estas tecnologías en caso de no conseguir solventar algún problema

Cuadro 2.5: Análisis, prevención y contingencia del riesgo R02

Riesgo	**R03. Aspectos teóricos relacionados con la fragilidad física desconocidos**
Descripción	El proyecto se basa en la fragilidad física. Se trata de un concepto del área de la salud y psicología que es inicialmente poco conocido por el desarrollador y autor de este documento
Magnitud	Media
Impacto	La especificación de requisitos e historias de usuario puede no estar bien definida o haber malinterpretaciones por parte del desarrollador
Indicadores	Diseño o desarrollo erróneo de la lógica de negocio o definición ambigua de los requisitos o historias de usuario
Plan de prevención	Realizar una correcta especificación y definición de requisitos y validar con el propietario del producto
Plan de contingencia	Contactar con el propietario del producto para aclarar las dudas y especificar los requisitos

Cuadro 2.6: Análisis, prevención y contingencia del riesgo R03

Riesgo	**R04. Diseño o implementación inadecuada de la interfaz gráfica**
Descripción	La aplicación va dirigida a un grupo de población muy concreto (las personas de edad avanzada) lo que implica la posibilidad de realizar un diseño inadecuado o poco usable
Magnitud	Según el momento en el que se detecte. En caso de detectarse durante la fase inicial del diseño, se trata de un riesgo de magnitud leve. En cambio, si se detecta en las últimas fases de desarrollo puede suponer un riesgo de magnitud alta
Impacto	Se debe tener en cuenta durante todo el diseño y desarrollo de la interfaz las dificultades y carencias visuales que presentan las personas de edad avanzada
Indicadores	Los usuarios no tienen una buena experiencia con el sistema, les cuesta seguir las indicaciones y no consiguen completar las pruebas
Plan de prevención	Tener en cuenta durante todo el diseño y desarrollo el tamaño de la letra, los colores y la usabilidad de la interfaz. Por ello será vital mostrar el diseño en las primeras fases al propietario del producto y, si es posible, a personas de edad avanzada para validar el proceso de creación de la interfaz gráfica
Plan de contingencia	En caso de que se detecte un problema durante el desarrollo de los prototipos, se modificará el diseño junto con la colaboración de los usuarios o del propietario del producto. Si el problema se detecta al final del desarrollo, se aplicarán los mínimos cambios necesarios para conseguir una usabilidad aceptada por los usuarios o por el propietario del producto

Cuadro 2.7: Análisis, prevención y contingencia del riesgo R04

Riesgo	**R05. Diseño o implementación errónea de la lógica de las pruebas**
Descripción	El desarrollo de las pruebas físicas presenta cierto nivel de complejidad, lo que puede dar lugar a una implementación equivocada de la lógica de las pruebas
Magnitud	Según el momento en el que se detecte. En caso de detectarse durante la fase inicial de desarrollo, se trata de un riesgo de magnitud media. En cambio, si se detecta en las últimas fases de desarrollo puede suponer un riesgo de magnitud alta
Impacto	Puede suponer realizar cambios a gran escala debido a una documentación ambigua o poco especificada.
Indicadores	El propietario del producto no está satisfecho con la funcionalidad del sistema o los usuarios no son capaces de realizar las pruebas
Plan de prevención	Especificar y detallar los requisitos e historias de usuario con el propietario del producto en cada reunión de planificación. Por supuesto, también es vital la demostración de la nueva funcionalidad en las reuniones de revisión al término de cada *sprint* para validar las pruebas y la funcionalidad del sistema
Plan de contingencia	Valorar los costes que supone el cambio de funcionalidad y priorizar las tareas o historias de usuario pendientes por desarrollar para, en caso de necesidad, descartar las de menor relevancia

Cuadro 2.8: Análisis, prevención y contingencia del riesgo R05

Riesgo	**R06. Imposibilidad de acoplar los subsistemas**
Descripción	Existe la posibilidad de que las tecnologías utilizadas para el desarrollo del proyecto imposibiliten o dificulten en gran medida el acoplamiento con los otros dos subsistemas que forman FTAS
Magnitud	Alta
Impacto	Puede suponer rediseñar el sistema e incluso el fracaso del proyecto
Indicadores	Los subsistemas que forman FTAS no se pueden comunicar o el acoplamiento está suponiendo un coste muy elevado en el desarrollo del proyecto
Plan de prevención	Realizar una formación en la primera fase del proyecto (en la primera iteración) sobre la tecnología MQTT con el lenguaje de programación C#
Plan de contingencia	Valorar los costes que supone elegir una tecnología alternativa. En caso de que el desarrollo se encuentre en una fase avanzada, se deberán priorizar las tareas más relevantes para llevarse a cabo frente a las de menor relevancia

Cuadro 2.9: Análisis, prevención y contingencia del riesgo R06

Riesgo	**R07. Falta de comunicación con el *product owner***
Descripción	El proyecto se realiza siguiendo la metodología de desarrollo ágil Scrum y la comunicación con el propietario del producto supone un factor clave para el desarrollo del proyecto
Magnitud	Alta
Impacto	La falta de comunicación con el propietario del producto puede suponer el fracaso del proyecto
Indicadores	El propietario del producto no acude con frecuencia a las reuniones diarias, de revisión y de planificación del *sprint*
Plan de prevención	Establecer una comunicación con el propietario del producto desde las fases iniciales del proyecto
Plan de contingencia	Contactar con el propietario del producto o, en el peor caso, contactar con las coordinadoras del grado

Cuadro 2.10: Análisis, prevención y contingencia del riesgo R07

Riesgo	**R08. Baja o ausencia de algún miembro del equipo**
Descripción	El único desarrollador del proyecto y autor de este documento pueda estar de baja durante la estancia en prácticas
Magnitud	Alta
Impacto	En el peor caso, si la baja se alarga durante un considerable espacio de tiempo puede suponer el fracaso del proyecto
Indicadores	Falta o ausencia del desarrollador
Plan de prevención	Tener en cuenta durante todas las reuniones de planificación las prioridades del propietario del producto y atender la evolución del proyecto en base a los requisitos iniciales
Plan de contingencia	Realizar la priorización de requisitos con el propietario del producto y descartar aquellas de menor relevancia para cubrir las más prioritarias

Cuadro 2.11: Análisis, prevención y contingencia del riesgo R08

2.5. Seguimiento del proyecto

En esta sección se detalla, para cada *sprint*, los principales objetivos definidos, los problemas detectados y cómo estos han sido abordados y han afectado al desarrollo del proyecto. También se incluye la información más relevante sobre las reuniones de revisión realizadas al término de cada iteración. El desarrollo del proyecto se ha realizado en un total de cinco iteraciones, cuyas fechas de inicio y término se presentan en el Cuadro 2.12

Sprint	Fecha inicio	Fecha fin
1	01/02/2022	14/02/2022
2	14/02/2022	28/02/2022
3	28/02/2022	14/03/2022
4	14/03/2022	28/03/2022
5	28/03/2022	11/04/2022

Cuadro 2.12: *Sprints* realizados a lo largo del proyecto

2.5.1. Primer *Sprint*

Para el primer *sprint* se propuso seleccionar las tareas relacionadas con la formación y configuración del proyecto base y la primera historia de usuario descrita en el Cuadro 2.1 tal y como se presenta en la Figura 2.2. El objetivo de este primer *sprint* era familiarizarse con las nuevas tecnologías y generar el proyecto base.

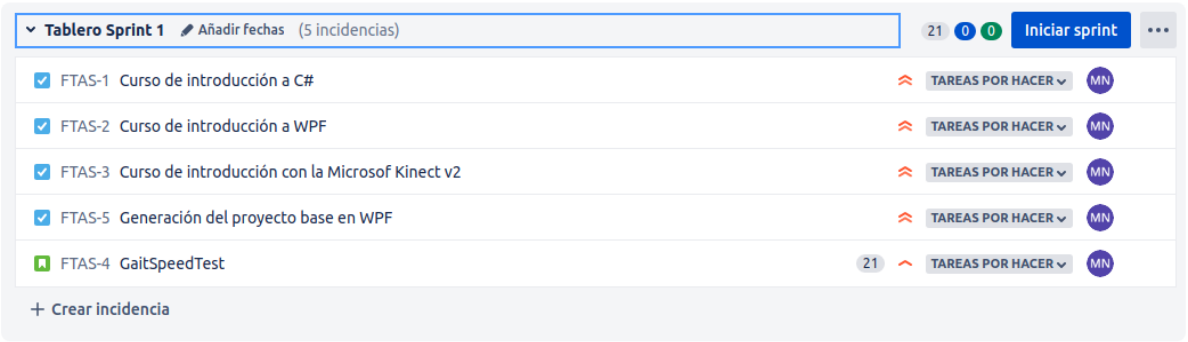
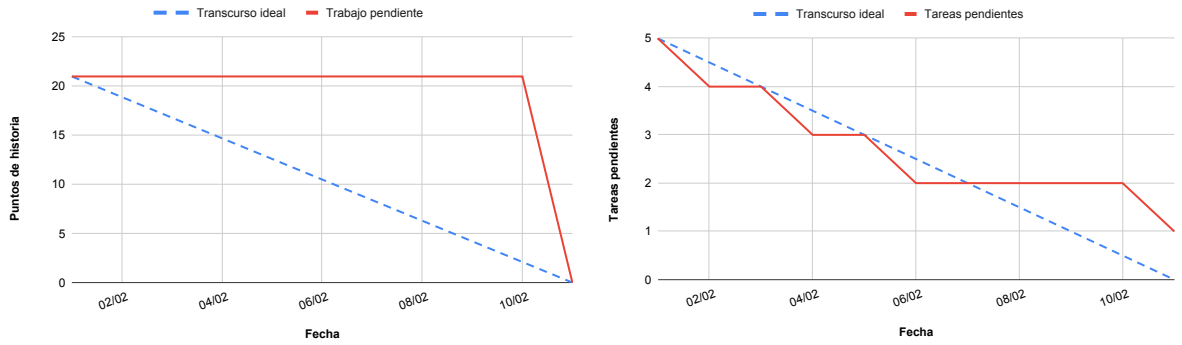


Figura 2.2: *Backlog* del primer *sprint* del proyecto

Transcurridas las dos semanas, el primer *sprint* planteó una serie de dificultades. La más destacable tenía que ver con la generación del proyecto base debido a que las herramientas y tecnologías de trabajo utilizadas eran desconocidas. Esta barrera obligó a desarrollar provisionalmente la lógica de la prueba de velocidad de la marcha (asociada a la historia de usuario HU01) en un proyecto base facilitado por el SDK de la cámara Kinect.

El resultado de esta primera iteración fue una formación completada para familiarizarse con las nuevas tecnologías y el desarrollo de una primera aproximación de la lógica de la prueba de velocidad de la marcha que se trabajó y mejoró en los posteriores *sprints*. También quedó pendiente para la siguiente iteración la generación del proyecto base. La Figura 2.3a y 2.3b muestra de forma visual el transcurso del *sprint* según los puntos de historia y las tareas completadas.



(a) Según los puntos de historia

(b) Según las tareas completadas

Figura 2.3: Gráficos *Burndown* del primer *sprint*

2.5.2. Segundo *Sprint*

El principal objetivo definido en la respectiva reunión de planificación de este segundo *sprint* fue desarrollar una aproximación de la lógica de las pruebas físicas y la generación del proyecto base. La Figura 2.4 presenta la tarea y las historias de usuario seleccionadas para esta segunda iteración. En un principio, solo se añadieron estas dos historias de usuario ya que se trata de una funcionalidad con bastante complejidad. Por ejemplo, la historia de usuario correspondiente a la prueba de levantarse y sentarse de la silla comprende una serie de escenarios y detalles descritos tal y como se aprecia en la Figura 2.5 y 2.6.

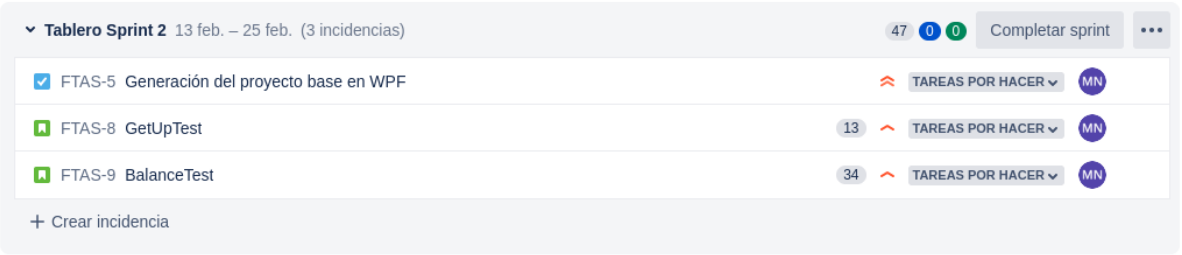


Figura 2.4: *Backlog* del segundo *sprint* del proyecto

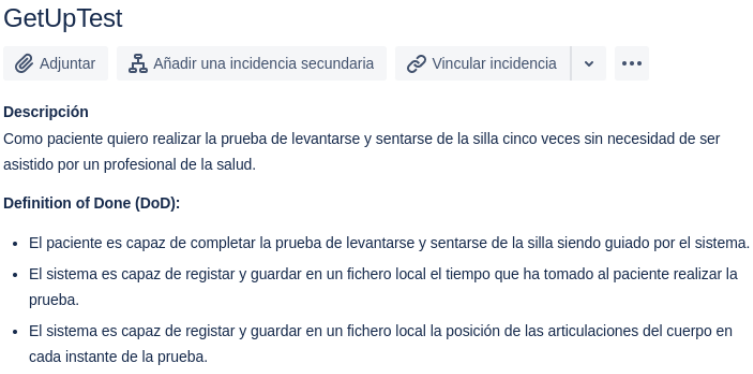
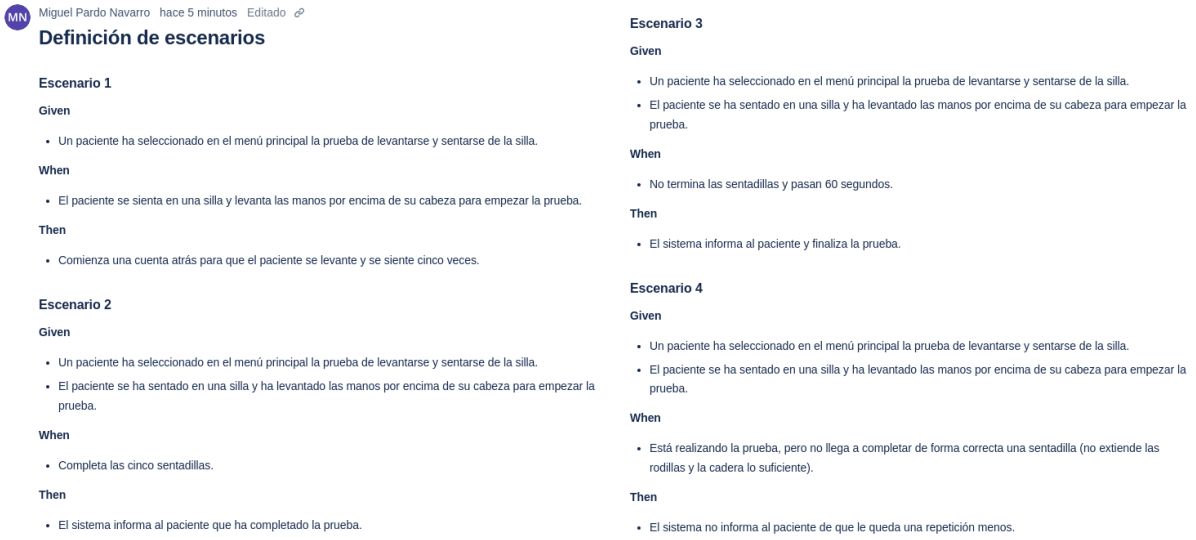


Figura 2.5: Descripción y *Definition of Done* de HU02

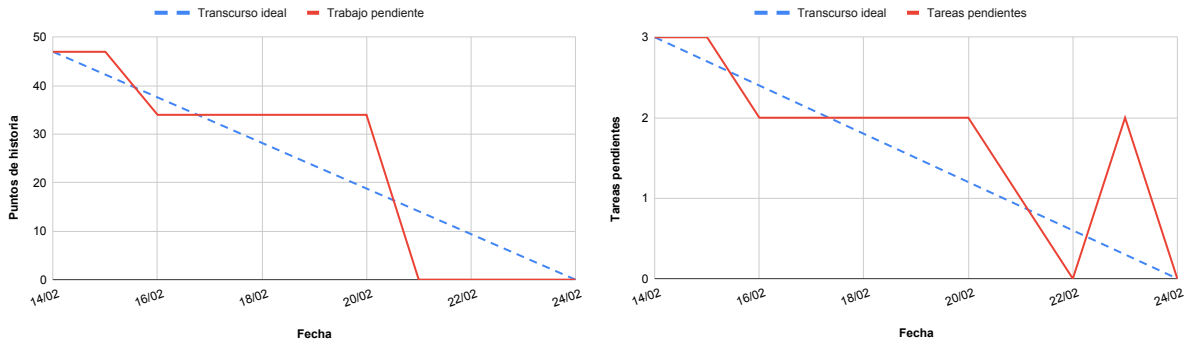


(a) Escenarios uno y dos

(b) Escenarios tres y cuatro

Figura 2.6: Definición de los escenarios de HU02

En el inicio de este *sprint* todavía no se consiguió generar el proyecto base y, por ello, en la reunión de planificación se propuso como objetivo seguir trabajando sobre la solución facilitada por el SDK de la Kinect desarrollando unas primeras aproximaciones de la lógica de las pruebas físicas. Finalmente, ambos objetivos fueron alcanzados, por un lado se desarrolló una funcionalidad para las dos pruebas que cumplía los *Definition of Done* (DoD) y los escenarios propuestos y, por otro lado, se consiguió resolver el problema que había a la hora de generar el proyecto propio desde cero. Una vez generado el proyecto base, se trasladó la implementación de la lógica de las pruebas ya implementadas y se aprovechó para realizar un refactorización de código y aplicar en mayor medida principios de *clean code* para hacerlo más mantenible y facilitar las siguientes modificaciones.



(a) Según los puntos de historia

(b) Según las tareas completadas

Figura 2.7: Gráficos *Burndown* del segundo *sprint*

La Figura 2.7a y 2.7b muestra los gráficos *Burndown* de esta segunda iteración donde se puede apreciar como, una vez terminadas las tareas inicialmente propuestas para el *sprint*

backlog, se añadieron y se completaron dos tareas más. Ambas tareas estaban relacionadas con la refactorización de código y fueron completadas antes de finalizar el *sprint*.

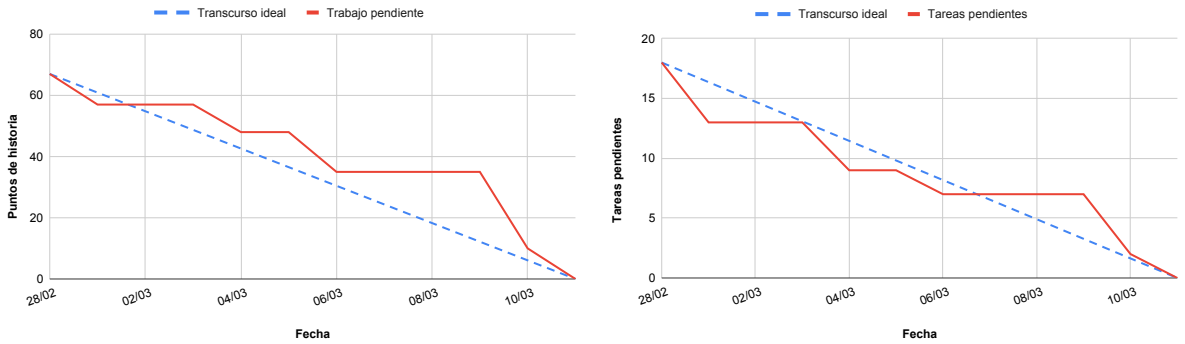
2.5.3. Tercer *Sprint*

La tercera iteración del proyecto tuvo como principal objetivo diseñar e implementar la interfaz gráfica de usuario de la aplicación. Además, en la reunión de planificación del *sprint* anterior se detectaron ciertos errores relacionados con la prueba de levantarse y sentarse de la silla. Por ello, tal y como se puede apreciar en la Figura 2.8, la pila del tercer *sprint* tiene asignada la resolución de dos *bugs* y el diseño e implementación de las pantallas de la aplicación. Cabe destacar que inicialmente se habían agrupado todas las tareas relacionadas con el diseño de la interfaz, pero debido a que se trataba de una tarea demasiado grande y poco especificada, se vio conveniente dividirla en subtareas más pequeñas para facilitar el seguimiento del proyecto. El diseño del sistema se realizó en Figma y, una vez terminado, se mostró y se validó con el propietario del producto antes de comenzar con la implementación de las pantallas.

Figura 2.8: *Backlog* del tercer *sprint* del proyecto

En el tercer *sprint* fue posible alcanzar los objetivos definidos para este y además, también se pudieron resolver los errores detectados sobre la prueba de las sentadillas. Es cierto que las tareas de diseño e implementación de las pantallas implicó una cantidad de trabajo menor

de lo esperado, pero todo lo relacionado con la lógica de la navegación supuso un problema importante que no se había tenido en cuenta en la planificación del *sprint* y que se terminó de resolver en la siguiente iteración.



(a) Según los puntos de historia

(b) Según las tareas completadas

Figura 2.9: Gráficos *Burndown* del tercer *sprint*

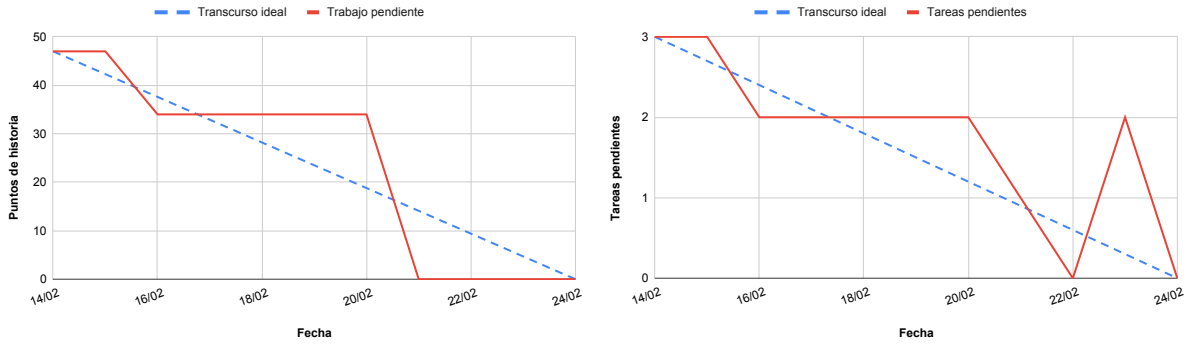
2.5.4. Cuarto *Sprint*

El objetivo de este cuarto *sprint* era completar las tareas e historias de usuario relacionadas con la identificación del paciente en el sistema. Como se ha comentado a lo largo de la memoria, la identificación de usuario se puede realizar mediante el escaneo de un código QR o a partir del DNI del paciente, siendo este segundo método más prioritario para el propietario del producto. La Figura 2.10 presenta las tareas e historias de usuario seleccionadas para esta nueva iteración en Jira en la que también se incluye el problema de la navegación detectado en el tercer *sprint*.

Figura 2.10: *Backlog* del cuarto *sprint* del proyecto

Los principales objetivos de este cuarto *sprint* fueron satisfechos, pero no se pudieron completar todas las tareas inicialmente propuestas. El motivo se debía a que la navegación supuso

replantear mucha lógica relacionada con la gestión de la vista. De hecho, se reemplazó el uso de *Windows* por *Pages* para solventar un problema de gestión de recursos que colapsaba la aplicación tras realizar varias veces alguna prueba. Por otra parte, la historia de usuario relacionada con la automatización de las tres pruebas también supuso una carga de trabajo mucho mayor de lo que inicialmente se esperaba. Tal y como se aprecia en la Figura 2.11a y 2.11b no se completó una tarea: la de mejorar la prueba de equilibrio.



(a) Según los puntos de historia

(b) Según las tareas completadas

Figura 2.11: Gráficos *Burndown* del cuarto *sprint*

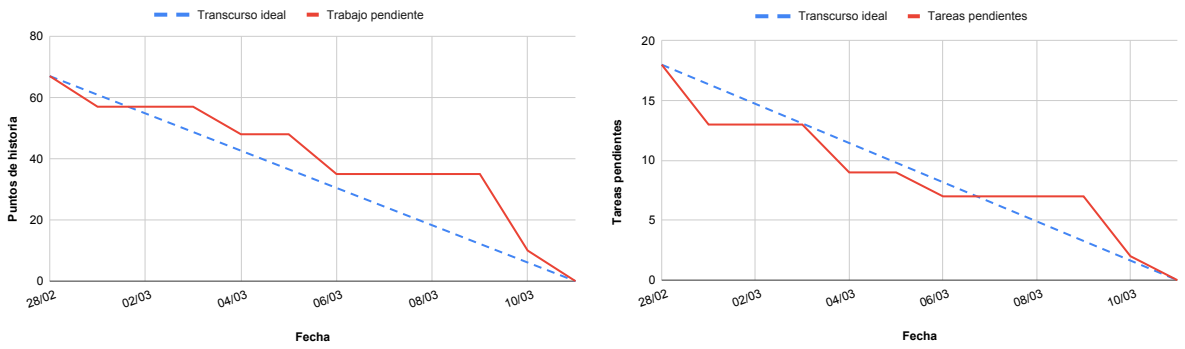
2.5.5. Quinto *Sprint*

Finalmente, el último *sprint* planteó en esencia dos objetivos. En primer lugar, se buscó desarrollar un algoritmo alternativo al inicialmente planteado para la prueba de equilibrio ya que era muy impreciso. En segundo lugar y con menor relevancia, también se pretendía mejorar ciertos aspectos de la interfaz gráfica, de la gestión y tratamiento de errores y del cálculo de ciertos parámetros como el número de pasos durante el transcurso del test. Es decir, esta última iteración tenía como principal objetivo trabajar en la lógica del test de equilibrio y, posteriormente, mejorar los acabados de la aplicación. La Figura 2.12 muestra la pila del último *sprint* del proyecto.

Figura 2.12: *Backlog* del quinto y último *sprint* del proyecto

En este quinto *sprint* se alcanzó el principal objetivo propuesto ya que se desarrolló un

método alternativo para la prueba de equilibrio de mayor precisión. Aún así, durante el desarrollo se detectó que esta prueba presentaba muchas dificultades debido a la incertidumbre del sensor Kinect a la hora de detectar la posición de los pies. En lo que respecta al resto de tareas, se pudieron completar las de mayor relevancia, incluyendo también la refactorización de código. La única tarea no completada fue la relacionada con la creación de animaciones para dar un resultado más profesional. Cabe destacar que esta tarea no era un requisito esencial del sistema. De la misma forma que para los anteriores *sprints*, la Figura 2.13 muestra los gráficos *Burndown* de esta última iteración.



(a) Según los puntos de historia

(b) Según las tareas completadas

Figura 2.13: Gráficos *Burndown* del quinto y último *sprint*

Para terminar, en la Figura 2.14 se presenta el *Velocity Chart* de los cinco *sprints* en los que se ha desarrollado el proyecto, donde se denotan dos aspectos principales. Por un lado, se puede apreciar como en casi todas las iteraciones ha ido creciendo la cantidad de puntos de historia desarrollados y, por otro lado, también se puede observar como se ha ido ajustando al trabajo que realmente se podía alcanzar en el periodo que suponía un *sprint*.

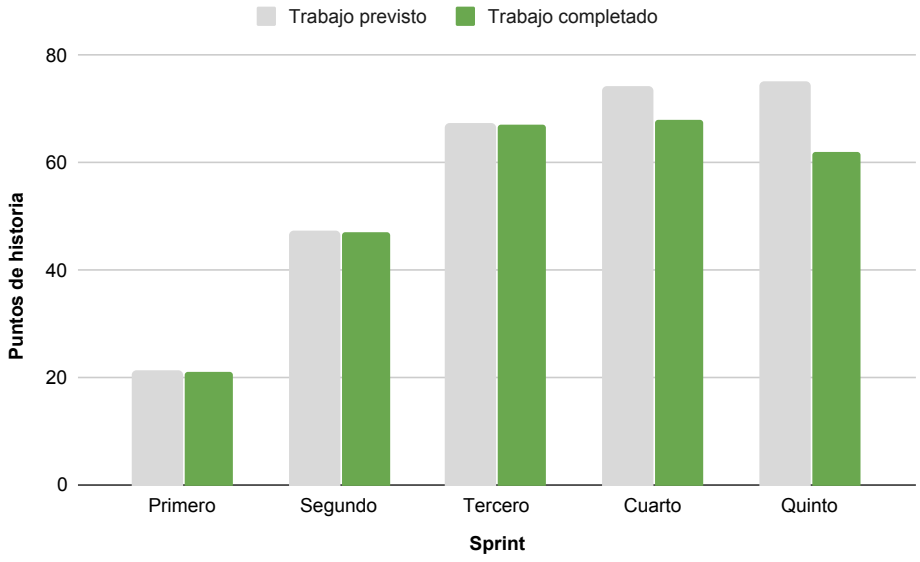


Figura 2.14: *Velocity report* del proyecto

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En esta sección se presenta la definición y especificación de requisitos y el modelado del sistema desarrollado. En primer lugar, la Figura 3.1 muestra el diagrama de casos de uso (DCU) del proyecto. Cabe destacar que, a pesar de hacer uso de una metodología ágil, se ha visto conveniente incluir un DCU para visualizar mejor la funcionalidad que plantea el sistema. Aún así, no se especificaron los casos de uso, sino las historias de usuario propuestas en el Cuadro 2.1.

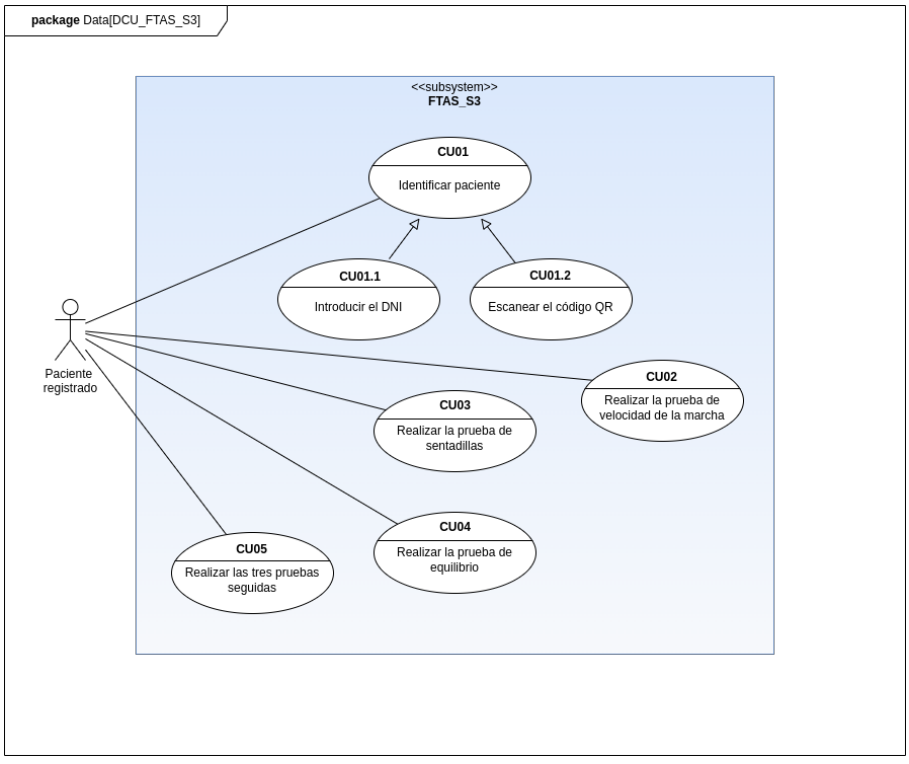


Figura 3.1: Diagrama de casos de uso del proyecto

En segundo lugar, el Cuadro 3.1 presenta todas las historias de usuario del proyecto. Estas siete historias de usuario ya han sido mostradas anteriormente en el Cuadro 2.1 al definir la pila del producto.

Id.	**Descripción**
HU01	Como paciente quiero poder realizar la prueba de velocidad de la marcha sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad.
HU02	Como paciente quiero poder realizar la prueba de levantarse y sentarse de la silla sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad.
HU03	Como paciente quiero poder realizar la prueba de equilibrio sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad.
HU04	Como paciente quiero poder identificarme en la aplicación mediante el DNI para realizar las pruebas para la detección de fragilidad física.
HU05	Como paciente quiero poder identificarme en la aplicación mediante el escaneo de un código QR presentado en la pantalla para facilitar la interacción con el sistema.
HU06	Como paciente quiero que el sistema envíe los resultados de las pruebas a un servidor para que pueden ser guardados y consultados posteriormente.
HU07	Como paciente quiero que el sistema permita realizar la prueba de velocidad de la marcha, la de levantarse y sentarse de la silla y la de equilibrio de forma continuada y autónoma sin emplear el teclado o ratón para facilitar la interacción con el sistema.

Cuadro 3.1: Historias de usuario del proyecto

En tercer lugar, en el Cuadro 3.2 se especifica la primera historia de usuario (HU01). La especificación incluye el DoD (*Definition of Done*) que describe y concreta cuando la historia de usuario ha sido completada. Además, junto con el DoD, también se incluyen los posibles escenarios que se pueden encontrar. La especificación del resto de historias de usuario han sido documentadas en el Jira del proyecto de la misma forma que se muestra en la Figura 2.5 y 2.6 para la segunda historia de usuario (HU02).

Especificación de HU01	
Descripción	Como paciente quiero poder realizar la prueba de velocidad de la marcha sin necesidad de ser asistido por un profesional de la salud para evaluar y analizar mi estado de fragilidad
***Definition of Done* (DoD)**	
1	El paciente es capaz de completar la prueba siendo guiado por el sistema.
2	El sistema es capaz de reconocer cuando el paciente empieza y termina la prueba.
3	El sistema es capaz de registrar el tiempo que toma al paciente recorrer los tres metros.
4	El sistema es capaz de calcular la puntuación de la prueba realizada en base al tiempo que le ha tomado al paciente recorrer los tres metros.
5	El sistema es capaz de enviar los resultados de la prueba al servidor para guardarlos.
Definición de escenarios	
1	**Dado** un paciente identificado en el sistema y habiendo seleccionado en el menú principal el botón para realizar la prueba de velocidad de la marcha. **Cuando** se coloca a una distancia de 4 metros respecto a la cámara y camina hacia ella. **Entonces** se empieza a registrar el tiempo y la posición del paciente.
2	**Dado** un paciente identificado en el sistema y habiendo seleccionado en el menú principal el botón para realizar la prueba de velocidad de la marcha. **Cuando** se coloca a una distancia de 4 metros respecto a la cámara, camina hacia ella y recorre los tres metros. **Entonces** el sistema procesa el tiempo y la cinemática del paciente registrada durante la prueba para obtener los resultados y enviarlos al servidor.
3	**Dado** un paciente identificado en el sistema y habiendo seleccionado en el menú principal el botón para realizar la prueba de velocidad de la marcha. **Cuando** se coloca a una distancia de 4 metros respecto a la cámara y camina hacia ella pero no recorre los tres metros sino que vuelve al inicio. **Entonces** el sistema deja de registrar el tiempo y reinicia el cronómetro para que el paciente pueda volver a realizar la prueba.
4	**Dado** un paciente identificado en el sistema y habiendo seleccionado en el menú principal el botón para realizar la prueba de velocidad de la marcha. **Cuando** selecciona el botón de abortar la prueba. **Entonces** el sistema deja de registrar el tiempo si lo estaba haciendo y vuelve al menú principal.

Cuadro 3.2: Especificación de la historia de usuario HU01

En cuarto lugar, en lo que respecta a las estructuras de datos para asegurar la persistencia de la información, en el Cuadro 3.3, 3.4, 3.5 y 3.6 se detallan los requisitos de los datos definidos y acordados con Arturo Gascó (el desarrollador del servidor). Estos requisitos se pueden contemplar como un acuerdo de lo que este subsistema debe enviar al servidor y lo que el subsistema espera recibir del servidor.

Requisito de datos RD01	
Nombre	GaitSpeedTestResult
Versión	1.0 (20/04/2022)
Autor	Miguel Pardo Navarro
Fuentes	*Frailty Test Assistance System*
Requisitos asociados	-
Datos específicos	Nombre del dispositivo, identificador del usuario, fecha, puntuación, tiempo realizado, velocidad media, distancia recorrida, pasos realizados y amplitud de los pasos
Ocurrencia media	Diez veces al día
Ocurrencia máxima	Cien veces al día
Importancia	Alta

Cuadro 3.3: Requisito de datos RD01

Requisito de datos RD02	
Nombre	GetUpTestResult
Versión	1.0 (20/04/2022)
Autor	Miguel Pardo Navarro
Fuentes	*Frailty Test Assistance System*
Requisitos asociados	-
Datos específicos	Nombre del dispositivo, identificador del usuario, fecha, puntuación, tiempo realizado, número de sentadillas, velocidad media de las sentadillas y altura media de las sentadillas
Ocurrencia media	Diez veces al día
Ocurrencia máxima	Cien veces al día
Importancia	Alta

Cuadro 3.4: Requisito de datos RD02

Requisito de datos RD03	
Nombre	BalanceTestResult
Versión	1.0 (20/04/2022)
Autor	Miguel Pardo Navarro
Fuentes	*Frailty Test Assistance System*
Requisitos asociados	-
Datos específicos	Nombre del dispositivo, identificador del usuario, fecha, puntuación y tiempo en equilibrio
Ocurrencia media	Diez veces al día
Ocurrencia máxima	Cien veces al día
Importancia	Alta

Cuadro 3.5: Requisito de datos RD03

Requisito de datos RD04	
Nombre	Patient
Versión	1.0 (20/04/2022)
Autor	Miguel Pardo Navarro
Fuentes	*Frailty Test Assistance System*
Requisitos asociados	-
Datos específicos	Identificador del usuario (DNI), nombre, apellidos y fecha de nacimiento
Ocurrencia media	Veinte veces al día
Ocurrencia máxima	Cien veces al día
Importancia	Alta

Cuadro 3.6: Requisito de datos RD04

A partir de estos requisitos de datos se puede definir y establecer el modelo conceptual del subsistema desarrollado tal y como se presenta en la Figura 3.2. Como se define en el diagrama UML, un paciente puede realizar multitud de pruebas físicas y obtener multitud de resultados, pero un resultado pertenece a un único paciente. Los datos del paciente son facilitados por el servidor, y los resultados de las pruebas es la información que este sistema recoge y envía al servidor para que sean guardados.

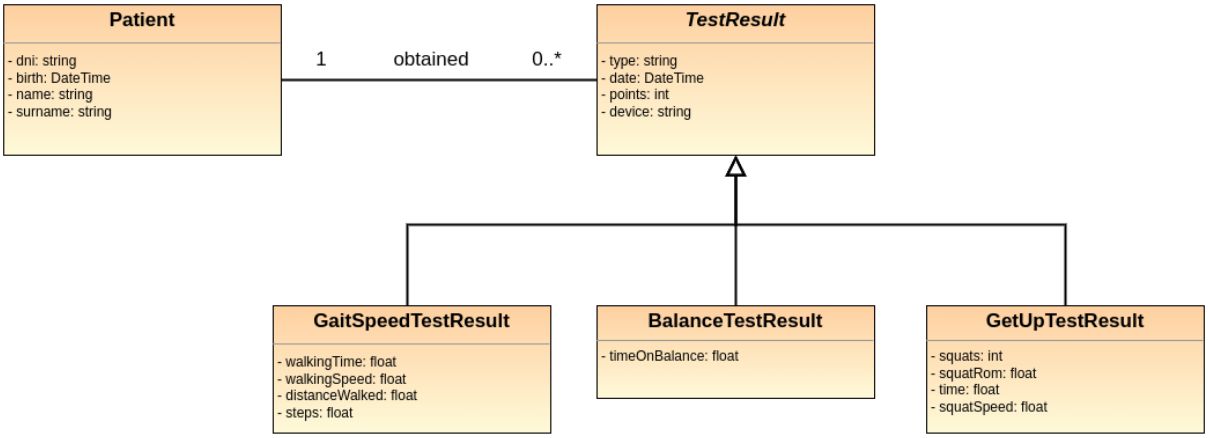


Figura 3.2: Modelo conceptual del sistema

En quinto lugar, en el Cuadro 3.7 y 3.8 se muestran dos requisitos de calidad para el sistema desarrollado. Cabe destacar que ambos requisitos han sido cubiertos durante el último *sprint* del proyecto.

Requisito de calidad RC01	
Nombre	Control de errores al identificarse con el DNI
Versión	1.0 (12/05/2022)
Autor	Miguel Pardo
Fuentes	*Frailty Test Assistance System*
Requisitos asociados	-
Descripción	El sistema debe ser capaz de detectar errores en el proceso de identificación mediante el DNI y proporcionar *feedback* al usuario
Importancia	Alta

Cuadro 3.7: Requisito de calidad RC01

Requisito de calidad RC02	
Nombre	Control de errores de conexión
Versión	1.0 (12/05/2022)
Autor	Miguel Pardo
Fuentes	*Frailty Test Assistance System*
Requisitos asociados	-
Descripción	El sistema debe ser capaz de detectar cuando no hay conexión o cuando no se puede conectar con el servidor y proporcionar *feedback* al usuario
Importancia	Media

Cuadro 3.8: Requisito de calidad RC02

Finalmente, la Figura 3.3 presenta el diagrama de actividades de la aplicación desde una perspectiva global. Para poder realizar las pruebas, el usuario se debe identificar mediante el escaneo del código QR, o bien a partir de su DNI. Una vez completado este proceso, como se aprecia en el diagrama, en caso de haberse identificado a través del código QR, el sistema lanzará las tres pruebas de forma seguida y automática para que el paciente no tenga que interactuar a través del ratón o teclado con el sistema. Por su parte, en caso de que el paciente se identifique mediante el DNI, la aplicación muestra una pantalla para que el paciente decida qué prueba realizar, e incluso si desea realizar las tres pruebas de forma seguida tal y como ocurre al escanear el QR. En cualquier caso, una vez finalizadas las pruebas, la aplicación redirige al paciente a la pantalla principal para repetir algún *test* o cerrar la sesión.

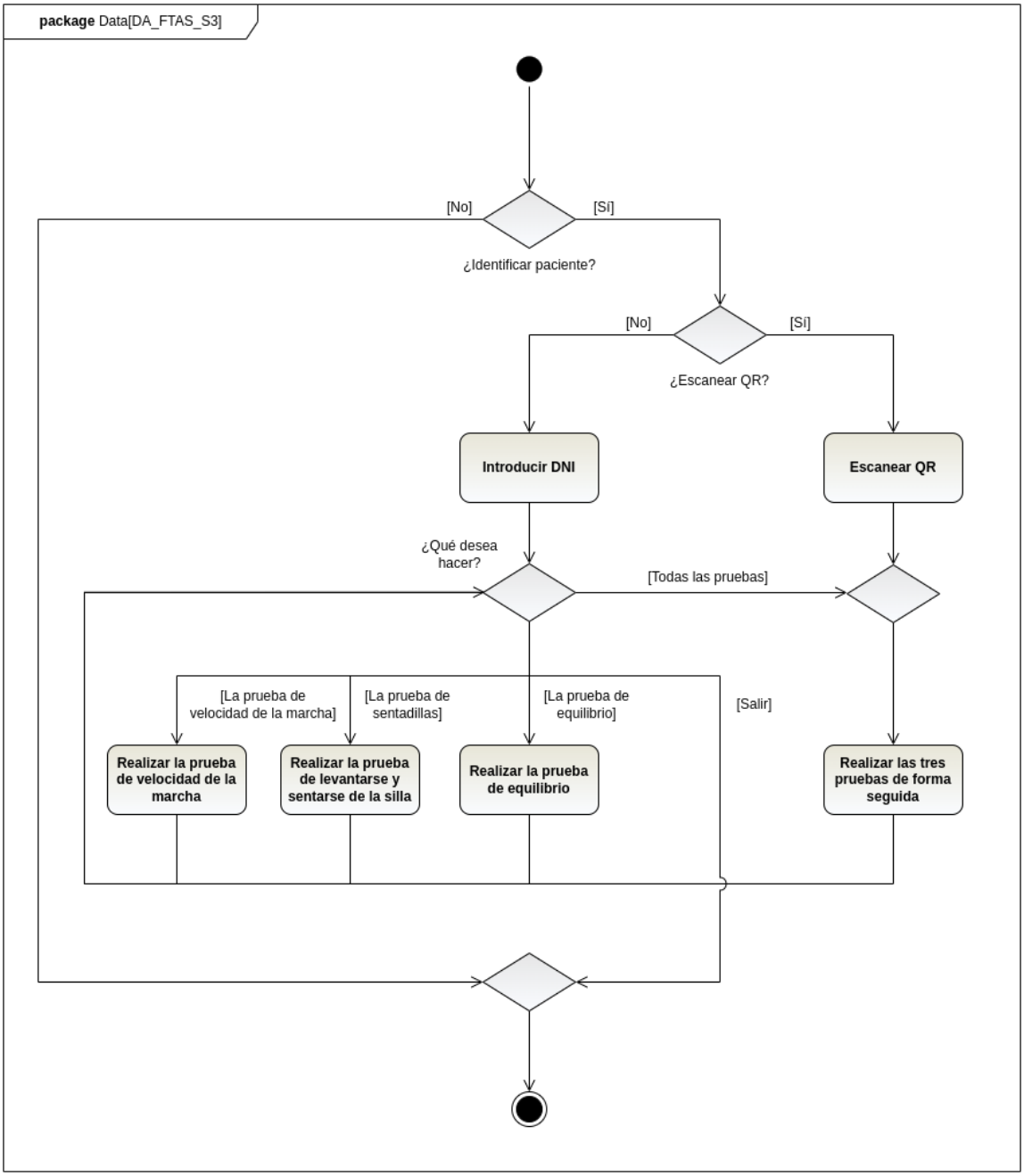


Figura 3.3: Diagrama de actividades del proyecto

3.2. Diseño de la arquitectura del sistema

En esta sección se van a describir los principales componentes del sistema y cómo estos se relacionan. Primero, es conveniente introducir el patrón arquitectónico Modelo-Vista-Controlador (MVC), o *Model-View-Controller* en inglés, ya que ha sido la arquitectura que se ha seguido a la hora de desarrollar el sistema descrito en este documento. Este conocido patrón permite desacoplar la interfaz gráfica de usuario con la lógica de negocio separando el sistema en distintos módulos tal y como se puede apreciar en la Figura 3.4.

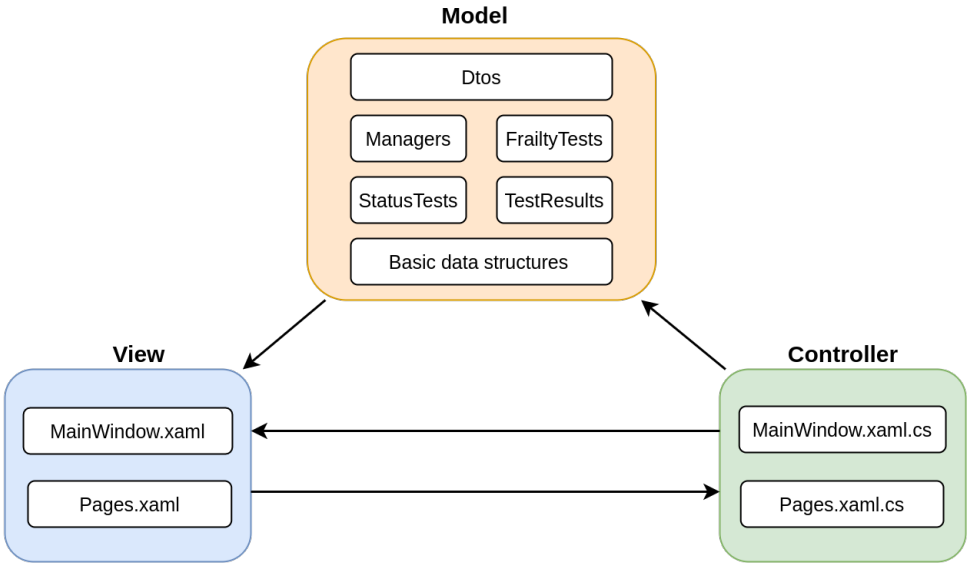


Figura 3.4: Modelo-Vista-Controlador

El *framework* WPF utiliza XAML para representar las vistas y, cada uno de estos ficheros está directamente asociado con un código escrito en el lenguaje C# que es normalmente denominado como el «*code behind*». Desde este código se pueden controlar los estados de la vista, acceder a los componentes de la misma e incluso escuchar o atender acciones que puede realizar el usuario como pulsar un botón. Estas características hacen que aplicar MVC sea casi obligatorio en el desarrollo de aplicaciones basadas en WPF, donde el *code behind* representa los controladores, los ficheros XAML representan la vista y el resto de clases y estructuras de datos representan el modelo. La Figura 3.5 muestra la analogía gráfica del *code behind* en WPF.

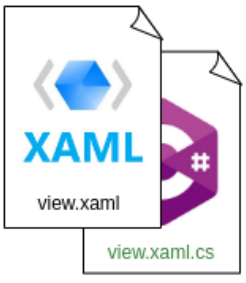


Figura 3.5: *Code behind* en WPF

Una vez descrita la arquitectura del sistema, en la Figura 3.6 se muestra el diagrama de clases resultante del proyecto tras la estancia en prácticas. En este diagrama UML, se incluyen las clases que representan el modelo conceptual y las que implementan la lógica y los algoritmos para realizar las pruebas físicas, gestionar la cámara Kinect, las peticiones HTTP, la comunicación con MQTT, etc. Cabe destacar que también se incluyen los controladores de las vistas representados de color azul para mostrar como estos interactúan con el modelo. A continuación, se detalla en profundidad las partes del diagrama de clases resultante.

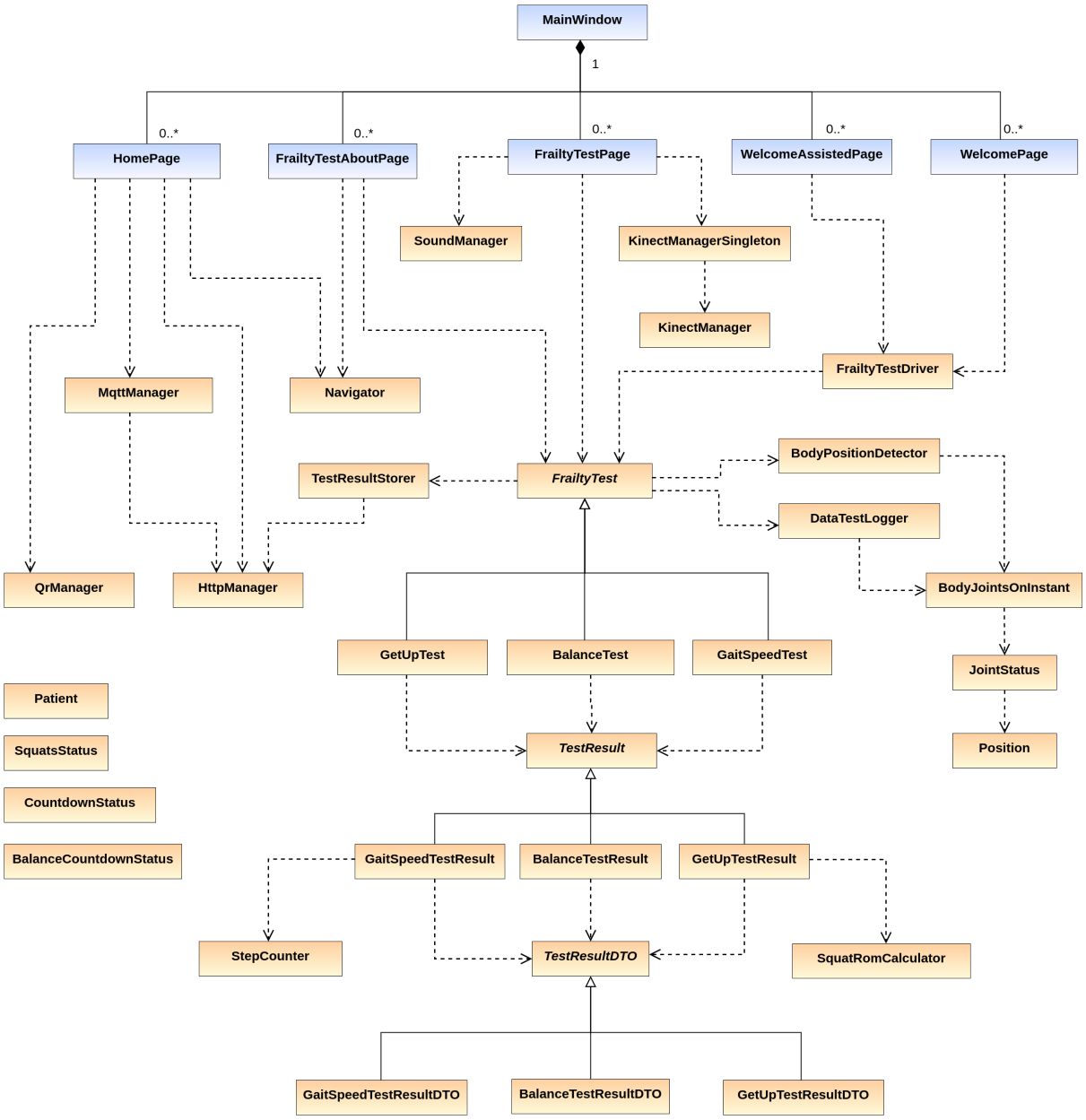


Figura 3.6: Diagrama de clases final del proyecto

En primer lugar, en la parte inferior izquierda del diagrama se pueden observar cuatro clases que no aparecen relacionadas: *Patient*, *SquatStatus*, *BalanceCountdownStatus* y *CountdownStatus*. Estas clases se detallan en la Figura 3.7. Por un lado, *Patient* define la estructura utilizada

por todas las clases que contienen la lógica de las pruebas y por los controladores de las vistas para almacenar la información del paciente que ha iniciado sesión en la aplicación. Esta clase ya ha sido definida y representada en el modelo conceptual en la Figura 3.2. No se ha decidido representar las relaciones de esta clase ya que complicaría en gran medida el diagrama presentado anteriormente en la Figura 3.6, pero cabe destacar que es una estructura de datos utilizada por muchas otras clases para asociar e identificar las pruebas físicas realizadas. Por otra parte, *SquatStatus*, *BalanceCountdownStatus* y *CountdownStatus* contienen gran parte de las cadenas de texto que se muestran en la pantalla para guiar al paciente durante el transcurso de las pruebas.

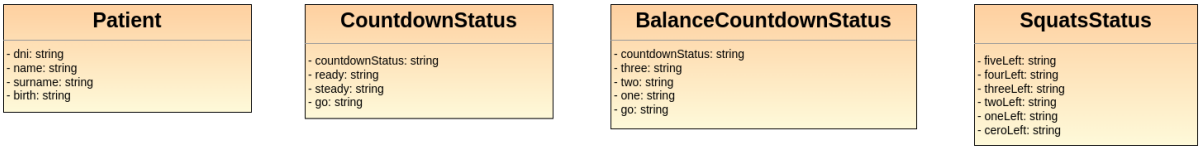


Figura 3.7: DC bloque 1. Estructuras básicas y contenedores de texto

En segundo lugar, la parte superior del diagrama de clases aparece detallada en la Figura 3.8. Esta sección representa los principales gestores del sistema y los controladores. Como ya se ha mencionado, los controladores se han representado de color azul y son las clases asociadas directamente con las vistas. Normalmente, los controladores no suelen aparecer representadas en el DC, pero debido a la peculiar estructura de WPF se ha visto conveniente incluirlos. Por otro lado, los gestores de la aplicación también se representan en la Figura 3.8 y, cada uno de ellos tiene una funcionalidad muy concreta:

- La clase *HttpManager* tiene la responsabilidad de gestionar y realizar las peticiones HTTP al servidor. Su funcionalidad incluye la de enviar los resultados de las pruebas y la de recoger la información del paciente a partir del DNI.
- La clase *QrManager* es la encargada de generar el código QR a partir del identificador de la cámara.
- La clase *MqttManager* gestiona la comunicación mediante MQTT con el servidor. Esta clase es quien recibe un mensaje cuando el código QR generado es escaneado.
- La clase *Navigator* incluye la funcionalidad de integrar una página en la ventana principal. Se trata de una clase estática que puede ser llamada desde cualquier parte del código para cambiar de pantalla, volver a la anterior, al menú principal o al menú de inicio.
- La clase *SoundManager* implementa una sencilla funcionalidad para producir sonido cuando alguna de las pruebas físicas no se está realizando correctamente o cuando es conveniente llamar la atención del usuario (por ejemplo al dar comienzo una prueba).
- La clase *KinectManager* gestiona y genera el identificador de la cámara y, además, contiene una variable del tipo *KinectSensor*. La clase *KinectSensor* es proporcionada por el SDK de la Kinect y es la que facilita la posición de las articulaciones en tiempo real de los cuerpos detectados por el dispositivo. Este mismo SDK proporciona otras clases de gran utilidad como *Body* o *Joint*.

- La clase *KinectManagerSingleton* asegura que no existen diferentes instancias e identificadores de un mismo sensor Kinect en la aplicación.
- Finalmente, la clase *FrailtyTestDriver* contiene la lógica para realizar las pruebas físicas de forma continuada sin necesidad de que el paciente tenga que interactuar mediante el teclado o ratón con la aplicación. Es decir, esta clase contiene la funcionalidad para «conducir» las pruebas cuando estas se desean realizar de forma seguida y no de forma individual.

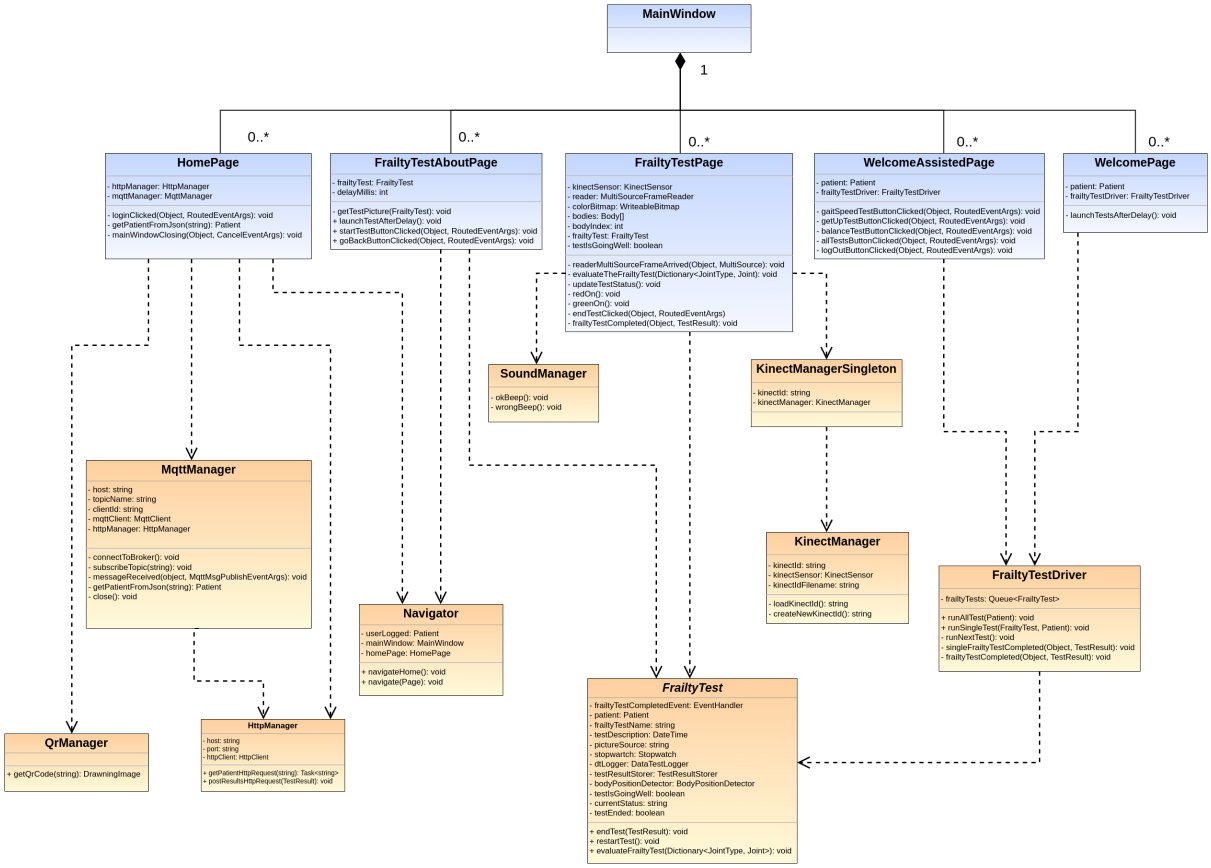


Figura 3.8: DC bloque 2. Controladores de la vista y gestores principales

El bloque central del diagrama se representa en la Figura 3.9. Esta parte se podría considerar el núcleo de la aplicación ya que contiene la lógica más importante y característica de este subsistema. Las clases *GetUpTest*, *BalanceTest* y *GaitSpeedTest* contienen la lógica de las pruebas físicas para la detección de fragilidad. Por su parte, *FrailtyTest* contiene la funcionalidad común de estas pruebas y también sirve como un punto de acceso para los gestores y controladores. El resto de clases de este tercer bloque se describen a continuación:

- *BodyPositionDetector* encapsula la lógica para reconocer la posición de un cuerpo dadas las posiciones de las articulaciones en un instante de tiempo determinado.
- *DataTestLogger* es la clase que se encarga de almacenar las posiciones de las articulaciones en cada instante de tiempo durante el transcurso de la prueba para, posteriormente, procesar la información y generar los resultados.

- *BodyJointsOnInstant* es una estructura de datos que contiene el estado de todas las articulaciones en un instante de tiempo determinado.
- *JointStatus* representa el estado de una articulación. El estado se refiere a la posición en tres dimensiones y su visibilidad respecto a la cámara: *tracked*, *inferred* o *nottracked*.
- La clase *Position* representa un punto del espacio en tres dimensiones.
- *TestResultStorer* es la clase cuya responsabilidad es la de asegurar la persistencia de los resultados una vez finalizada la prueba. Por ello, esta clase hace uso del gestor HTTP para enviar los resultados al servidor.

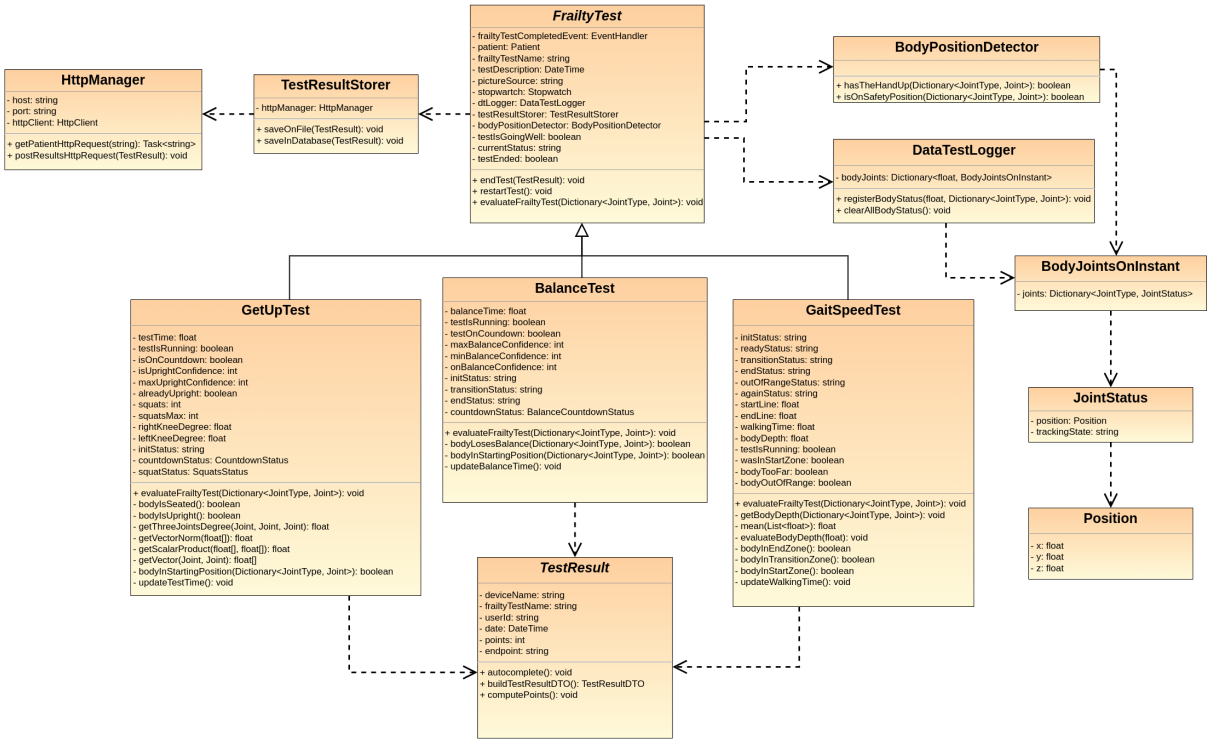


Figura 3.9: DC bloque 3. Lógica de las pruebas físicas

La parte inferior del diagrama de clases se detalla en la Figura 3.8. Esta zona representa las estructuras más cercanas al modelo relacional de la base de datos en la que se persiste la información. De hecho, las clases *GaitSpeedTestResultDTO*, *BalanceTestResultDTO* y *GetUpTestResultDTO* representan las clases definidas en el modelo conceptual de la Figura 3.2.

Siguiendo la misma jerarquía, las clases *GaitSpeedTestResult*, *BalanceTestResult*, *GetUpTestResult* y *TestResult* contienen la lógica necesaria para generar los resultados de las pruebas una vez realizadas. Es decir, cuando una prueba se lleva a cabo, esta genera un *TestResult* que, en base a ciertos parámetros como el tiempo y la posición de las articulaciones durante el transcurso de la prueba, calcula la puntuación, la velocidad y otros parámetros relevantes. Por último, cuando se completa la fase de cálculo de todos los parámetros, el resultado genera un tipo de *TestResultDTO*. De esta forma, solo será necesario serializar este DTO (*Data Transfer Object*) en formato JSON para enviarlo al servidor.

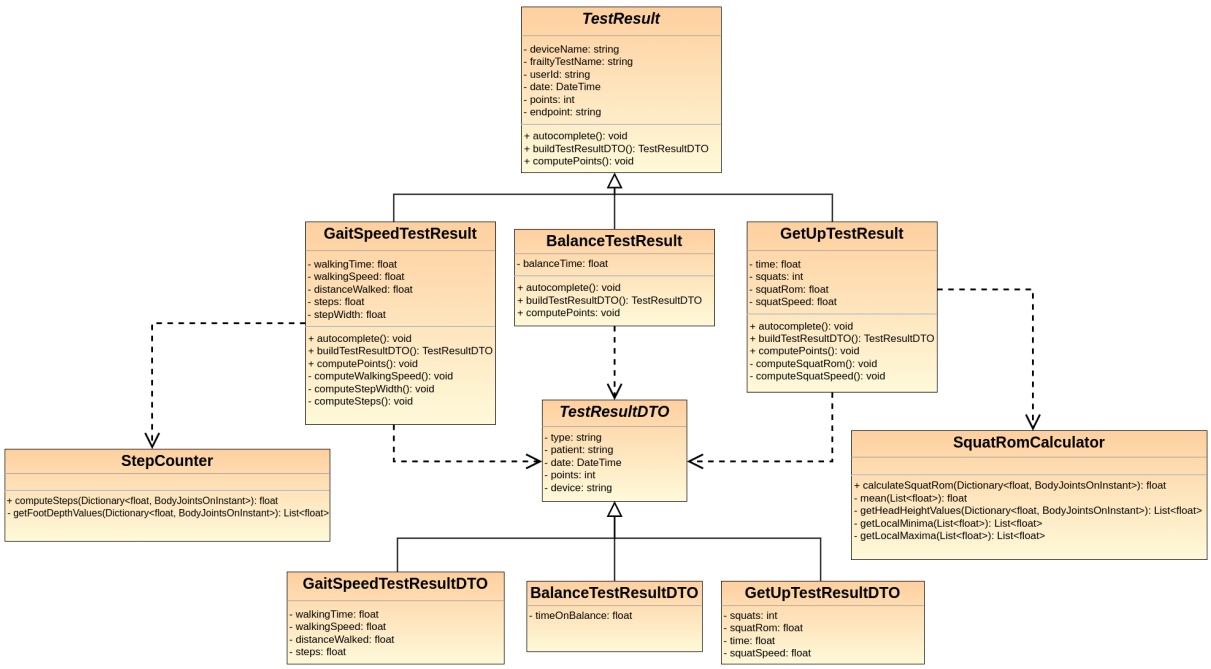


Figura 3.10: DC bloque 4. Modelo y lógica de los resultados

3.3. Diseño de la interfaz

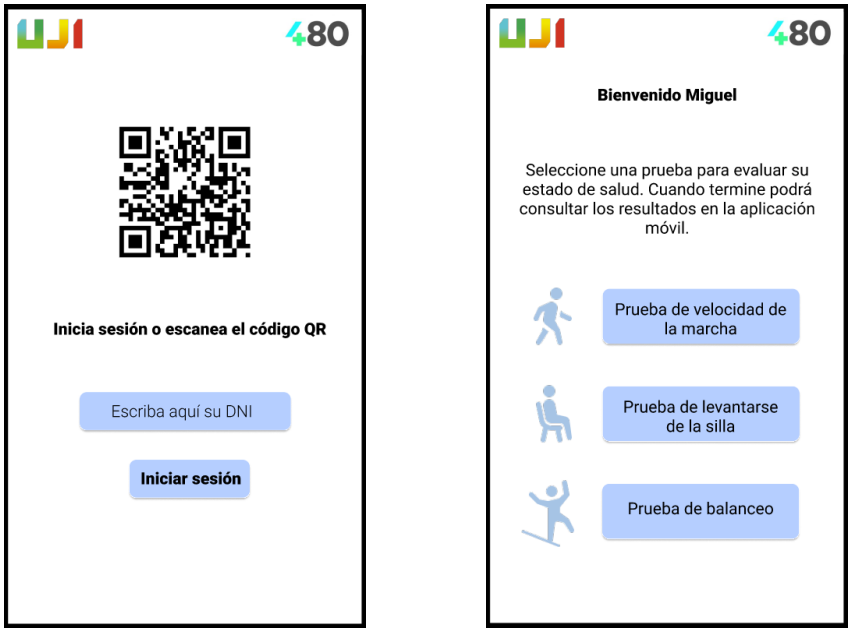
La interfaz gráfica de usuario ha sido diseñada haciendo hincapié en que el usuario final de este subsistema son personas de edad avanzada. Por esta razón se realizó un diseño sencillo y minimalista para facilitar el uso de la aplicación a este segmento de la población. Algunas de las características que incluye la interfaz para cumplir con este objetivo son:

- Sigue un estilo y diseño consistente.
- Utiliza una fuente sencilla y con tamaños de letra grandes.
- Ofrece una navegación simple y muy clara.
- Proporciona información adecuada sobre los posibles errores al introducir el DNI, e incluso durante el transcurso de las pruebas físicas.
- Utiliza botones grandes, con colores apropiados y sin abusar de símbolos o iconos reconocidos por usuarios habituados a las tecnologías.
- Se evitan colores estridentes o aquellos que puedan causar confusión en personas con problemas de visión.

De forma previa a la implementación de la interfaz se realizó el prototipado de la aplicación en Figma que fue aprobado y validado por el propietario del producto. Como se puede observar en las siguientes secciones, el resultado de la interfaz tras la estancia en prácticas es bastante fiel al diseño propuesto inicialmente en la fase de prototipado.

3.3.1. Prototipado

La Figura 3.11a muestra el diseño para la pantalla de inicio de la aplicación. Cuando el paciente se identifica se muestra la pantalla del menú principal cuyo prototipo se presenta en la Figura 3.11b. Por otro lado, para describir la pruebas antes de llevarlas a cabo se propusieron los diseños que se muestran en la Figura 3.12.



(a) Prototipo del menú de inicio (b) Prototipo de menú principal

Figura 3.11: Prototipos de las pantallas de inicio y del menú principal

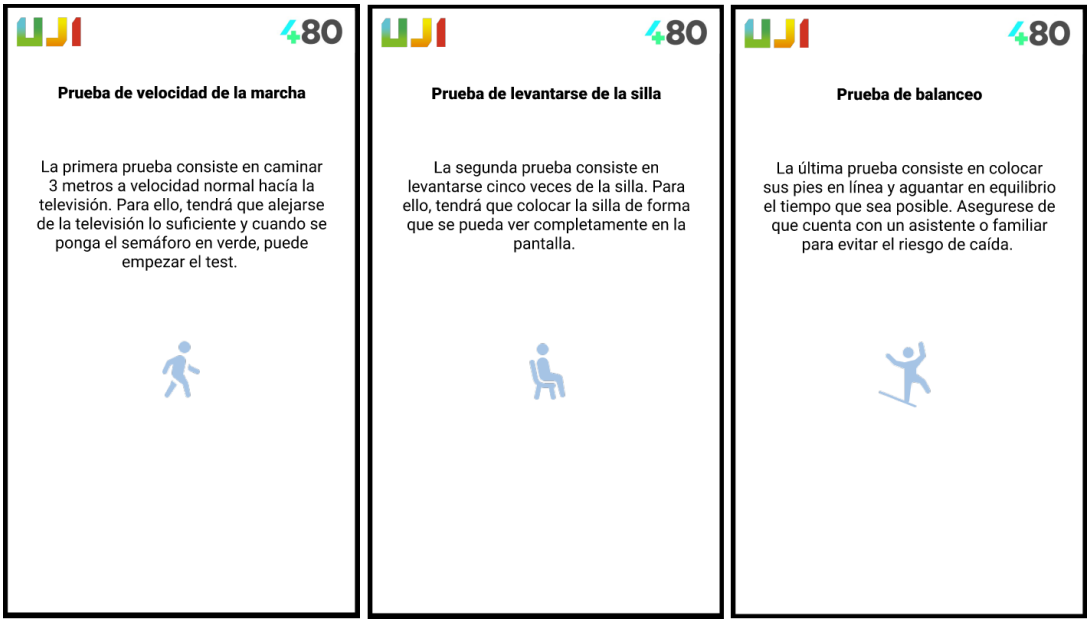


Figura 3.12: Prototipos de las pantallas de información previa a cada prueba

En el momento en el que se diseñaron los prototipos todavía no existía una idea clara de lo que se debía representar durante el transcurso de las pruebas. Lo que se propuso fue mostrar la imagen de la cámara Kinect en tiempo real para que el paciente pudiera verse en la pantalla como si fuera un espejo. Además, también se propuso mostrar en todo momento un texto con la instrucción que el paciente debe realizar para que sepa que acciones debe tomar. Por último, para proporcionar *feedback* al usuario del estado de la prueba se propuso hacer uso de un método parecido al de un semáforo. Esto permite a los pacientes reconocer cuando están realizando las pruebas de forma adecuada o, por el contrario, cuando ha surgido algún problema. Esta decisión se basó en la idea de lo efectivo y conocido que es el mensaje que proporcionan los semáforos en la circulación incluso para las personas de edad avanzada. Lo que se pretende expresar de forma implícita cuando el semáforo está en rojo durante el transcurso de la prueba es algo como «Párese un momento y atienda a las instrucciones que se le proporcionan». Por su parte, cuando el semáforo está en verde se pretende proporcionar un mensaje del tipo «Continúe, la prueba la está realizando correctamente».

Las Figuras 3.13, 3.14 y 3.15 muestran el diseño con algunos de los estados posibles durante el transcurso de la prueba de velocidad de la marcha, la prueba de levantarse y sentarse de la silla y la prueba de equilibrio respectivamente (siguiendo la idea descrita en el párrafo anterior). En los diseños, la zona gris representa la imagen «espejo» proporcionada por la cámara.

Figura 3.13: Prototipos de las pantallas durante la prueba de velocidad de la marcha

Figura 3.14: Prototipos de las pantallas durante el transcurso de la prueba de sentadillas

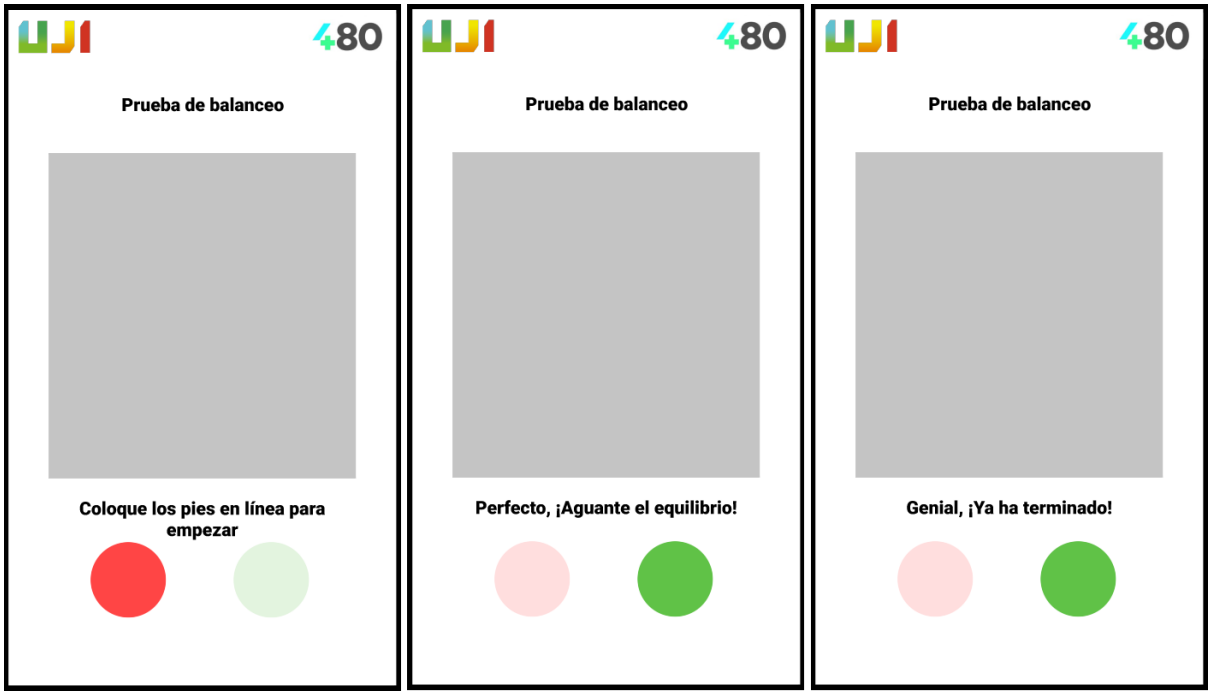


Figura 3.15: Prototipos de las pantallas durante el transcurso de la prueba de equilibrio

3.3.2. Diseño final

Finalmente, en esta sección se presentan los resultados tras la estancia en prácticas de la interfaz gráfica. Siguiendo el mismo orden que en los prototipos, la Figura 3.16a muestra la pantalla de inicio y la Figura 3.11b muestra la pantalla con el menú principal. Las pantallas que muestran la información de las pruebas se pueden ver en la Figura 3.17.

(a) Pantalla del menú de inicio (b) Pantalla del menú principal

Figura 3.16: Pantallas de inicio y del menú principal

Figura 3.17: Pantallas de información previa a cada prueba

Las pantallas que se muestran durante el transcurso de las pruebas se pueden observar en la Figura 3.18 para la prueba de velocidad de la marcha, en la Figura 3.19 para la prueba de levantarse y sentarse de la silla y en la Figura 3.20 para la prueba de equilibrio.

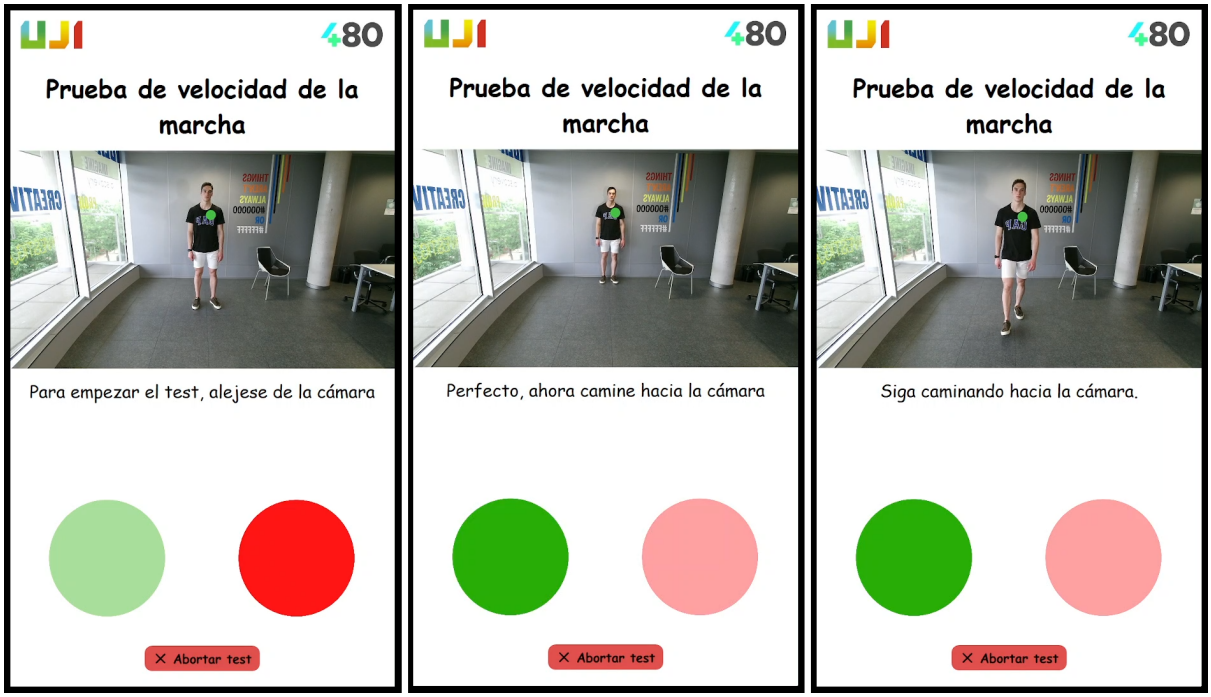


Figura 3.18: Pantallas durante el transcurso de la prueba de velocidad de la marcha

Figura 3.19: Pantallas durante el transcurso de la prueba de sentadillas

Figura 3.20: Pantallas durante el transcurso de la prueba de equilibrio

La principal diferencia respecto a los diseños planteados se encuentra en la pantalla principal, donde se han añadido dos nuevos botones, uno que permite realizar las pruebas de forma seguida simulando el mismo proceso que al escanear el código QR y otro para salir de la sesión. Además, en la vista que se muestra el transcurso de las pruebas se añadió el botón de abortar test en caso de que el paciente no desee completar la prueba en cuestión o haya ocurrido algún problema.

Capítulo 4

Implementación y pruebas

4.1. Detalles de implementación

En esta sección se describe el trabajo de programación realizado. Primero se describe la estructura y organización del proyecto, posteriormente se presentan y se detallan los patrones de diseño aplicados y, seguidamente se detallan los aspectos más relevantes en lo que se refiere a la implementación de los algoritmos y a la comunicación con los otros subsistemas que componen *Frailty Tests Assistance System* (FTAS).

4.1.1. Estructura y organización del proyecto

Para la consecución del proyecto se generaron 3.115 líneas de código fuente, 33 clases, 5 vistas con sus respectivos controladores y 2 archivos de configuración. En la Figura 4.1 se muestra la estructura y organización del proyecto. Como se puede apreciar, este se divide principalmente en cuatro carpetas o paquetes:

- *Images* contiene las figuras e imágenes estáticas para la interfaz gráfica.
- *Model* contiene las clases correspondientes al modelo, es decir, las clases que integran toda la lógica de negocio y las estructuras que definen el modelo conceptual. Todas estas clases han sido detalladas en la Sección 3.2 de este documento. A su vez, este paquete queda dividido en seis carpetas según el rol o responsabilidad de cada fichero o, en este caso, cada clase.
  - *BasicData* contiene las clases básicas y simples como *Patient* o *Position* que no contienen lógica, sino más bien sirven como estructura de datos a otras clases.
  - *Dtos* contiene las clases cuya estructura es la que espera recibir el servidor. Estas clases representan el modelo conceptual mostrado en la Figura 3.2.
  - *FrailtyTests* contiene las clases más representativas de este proyecto. En esta carpeta se encuentra la lógica de las tres pruebas físicas para la detección de fragilidad.

- *Managers* contiene las clases que gestionan el sistema. Cada uno de estos gestores tiene una funcionalidad muy concreta y, en muchos casos, tienen la utilidad de servir funcionalidad a los controladores de las vistas y a la lógica de las pruebas físicas.
- *StatusTest* contiene tres archivos. El objetivo de estas clases es contener y facilitar muchos de los textos que se muestran en pantalla durante el transcurso de las pruebas físicas.
- *TestResults* contiene las estructuras de datos para gestionar los resultados de las pruebas y cierta lógica para calcular la puntuación de las pruebas realizadas.

- *Sounds* contiene los archivos de sonido añadidos en la aplicación. Estos archivos son solo dos, uno para informar al paciente cuando la prueba está siendo realizada correctamente y otro para informar al paciente cuando la prueba no se está realizando de la forma en la que se espera.
- *View* contiene los ficheros XAML y sus respectivos controladores (*code behind*). Por tanto, este paquete no solo contiene lo que el patrón MVC agrupa como vista, sino también los controladores.

Por otra parte, *App.xaml* y su respectivo controlador (*App.xaml.cs*) es el código que realiza el arranque de la aplicación. Se podría decir que es como el *Main* de los proyectos con el lenguaje de programación Java. Este fichero llama a *MainWindow* que, como se puede deducir, es el fichero que integra las vistas de la aplicación en la ventana visible.

Finalmente, el resto de ficheros son principalmente archivos de configuración. Por ejemplo, *Properties* se trata de un directorio especial que contiene los ficheros que hacen posible el ensamblado del sistema y ejecución del mismo. Por su parte, *Referencias* contiene las dependencias externas y otras librerías.

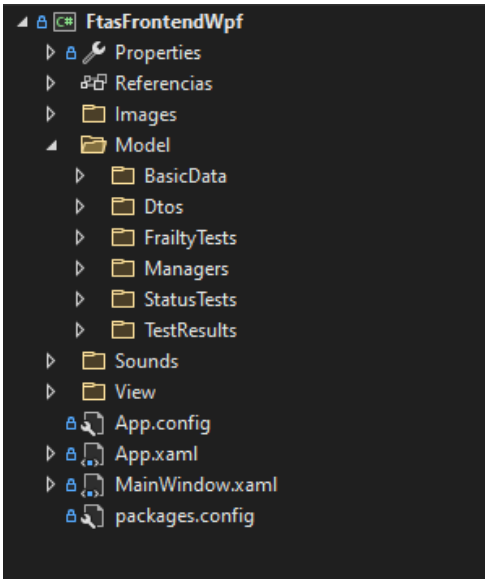


Figura 4.1: Estructura y organización del proyecto

4.1.2. Patrones de diseño

En esta sección se van a detallar y justificar los patrones de diseño aplicados en el sistema desarrollado.

Model-View-Controler (MVC)

El primer patrón que implementa el sistema desarrollado es el Modelo-Vista-Controlador. Este patrón ya ha sido introducido en secciones previas porque define la arquitectura del sistema desarrollado. Este patrón tiene el objetivo principal de desacoplar la interfaz de usuario con la lógica del sistema y, por ello, propone separar el código en tres módulos: el modelo, la vista y los controladores.

En el modelo se incluyen las estructuras de datos, los gestores y las clases que contienen la lógica de las pruebas físicas y de los resultados de las mismas.

Los controladores incluyen la lógica de las decisiones que se deben tomar frente a una nueva acción del usuario. Como ya se ha detallado previamente, en WPF este módulo es abarcado por el denominado *code behind*.

Finalmente, la vista contiene los ficheros *XAML* y su responsabilidad es la de presentar la interfaz de usuario. Las vistas deben ser simples y tener el mínimo conocimiento del modelo para que los posibles cambios en la lógica de negocio no impliquen realizar grandes modificaciones en el código de la vista y viceversa.

Actualmente existen derivaciones o adaptaciones de este conocido patrón arquitectónico, como el *Model-View-Presenter* (MVP) o *Model-View-ViewModel* (MVVM), que están tomando mucha fuerza. Este último, el MVVM, es ampliamente utilizado en WPF y presenta ciertas ventajas a la hora de realizar pruebas unitarias y desacoplar más todavía la vista del modelo. Durante el desarrollo del proyecto se planteó refactorizar el código para adaptar el sistema según el patrón MVVM, pero debido a las dificultades que planteaba en relación al enlace de la cámara Kinect y el coste que suponía realizar todos los cambios se optó por no llevarlo a cabo. Además, una de las principales ventajas que aporta MVVM tiene que ver con la creación de *tests* unitarios en el controlador y esto quedaba fuera del alcance del proyecto.

Singleton

El segundo patrón de diseño implementado es el *Singleton*. Se trata de un patrón de diseño creacional y, sin duda, uno de los patrones más conocidos en el ámbito de la Ingeniería del Software. El patrón *Singleton* asegura que una clase tiene una única instancia y, además, le proporciona acceso global a dicha instancia [13]. En este proyecto, ha sido necesario asegurar que los controladores accedan siempre a la misma instancia del sensor Kinect para evitar posibles fugas de memoria o problemas relacionados con la gestión de recursos del sistema. Además, este patrón ha facilitado también la identificación de la cámara que aloja el dispositivo ya que, cuando se desea obtener el identificador de la cámara Kinect (por ejemplo para generar el código QR

o para asociar los resultados de las pruebas con la cámara), se genera y se guarda una cadena aleatoria y única que permite identificar el sensor. De esta forma, la segunda y siguientes veces que el sistema necesite conocer el identificador de la cámara lo recogerá del fichero guardado en memoria asegurando que el sensor siempre tiene el mismo identificador.

Data Transfer Object (DTO)

El tercer patrón de diseño utilizado es conocido como *Data Transfer Object* (DTO). El objetivo es sencillamente encapsular la información que se desea enviar o compartir entre distintos sistemas o aplicaciones [6]. En el proyecto, aplicar este patrón tenía mucho sentido a la hora de crear una clase o estructura de datos para los resultados de las pruebas ya que estos debían ser transformados en formato JSON para enviarse al servidor siguiendo una estructura determinada. Por ello, utilizar DTOs permitía tener clases desacopladas de la lógica del sistema sin que se vea gravemente afectada a la hora de realizar cambios en las estructuras que el servidor esperaba recibir. Además, facilitaba la lógica a la hora de generar el JSON, ya que C# permite crear cadenas de texto según dicho formato a partir del nombre de los atributos de la clase y los valores que contiene la instancia. El proceso en el que se ven involucrados los DTOs aparece representado en la Figura 4.2.

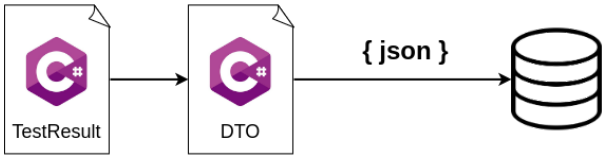


Figura 4.2: Envío de los resultados de las pruebas con el patrón DTO

4.1.3. Algoritmos para la detección de fragilidad

El núcleo de este proyecto se basa en la detección y seguimiento de los pacientes para realizar tres pruebas físicas con una cámara 3D. Generar esta lógica ha sido, sin duda, la parte que más trabajo ha requerido en el desarrollo de la aplicación.

Como ya se ha mencionado en la introducción, el sensor Kinect v2 es capaz de determinar la posición de 25 articulaciones de hasta 6 cuerpos distintos. Esta información se recoge de forma iterativa con una frecuencia aproximada de treinta veces por segundo (30 *frames per second*). En cada una de estas iteraciones, la SDK de la Kinect proporciona distintas fuentes de información para recoger la imagen en color o la lista de cuerpos detectados con sus respectivas articulaciones y posiciones. La imagen en color y las posiciones de los cuerpos es la información que la aplicación desarrollada recogía, aunque cabe mencionar que también proporciona imagen de profundidad y captura de audio.

Como cabe esperar, toda la gran cantidad de información facilitada en tan poco tiempo obligaba a gestionar de forma óptima los recursos hardware para no colapsar la aplicación o el funcionamiento de la cámara. Obviamente, para guiar al paciente durante el transcurso de las pruebas era necesario evaluar las posiciones de las articulaciones en tiempo real, pero

siempre y cuando fuera posible, debían hacerse los cálculos una vez liberados los recursos de la Kinect. Dicho esto, a continuación se describe de forma simplificada los pasos de los algoritmos implementados para guiar y asistir a los pacientes durante el transcurso de las pruebas.

El algoritmo comienza cuando el paciente ya se ha identificado y desea realizar alguna de las pruebas físicas. En este momento, el primer paso es inicializar y «abrir» el sensor Kinect además de suscribirse a las fuentes de datos que van a ser necesarias para poder recoger la información de cada nuevo *frame*.

Una vez inicializado el dispositivo comienza la segunda fase, donde se recoge de forma iterativa (con cada nuevo *frame*) todos los cuerpos detectados. En este proyecto, únicamente era necesario realizar el seguimiento de un esqueleto o cuerpo: el del paciente. Por ello, aunque la Kinect proporcione las articulaciones y posiciones de hasta seis cuerpos, siempre se descartaban todos menos uno. Esto se hacía de forma sencilla ya que la cámara proporcionaba un *array* con los cuerpos manteniendo la misma posición para cada uno de ellos en el vector. Esto, cuando la cámara detectaba varios cuerpos en distintos *frames*, permitía fijar uno de ellos ya que su posición en el vector no cambiaba mientras no desapareciese del espectro del sensor o fuese confundido por otra persona u objeto. En este punto de la iteración, el sistema ya tenía fijado un *Body* que, normalmente, se correspondía con el *Body* fijado en el *frame* previo.

El siguiente paso del algoritmo consiste en evaluar el estado de la prueba física dadas las articulaciones que proporciona la clase *Body* del SDK. Toda esta lógica varía según el tipo de prueba y es lo que se encapsula en la clase *FrailtyTest* y sus hijas: *GetUpTest*, *BalanceTest* y *GaitSpeedTest*. Toda la lógica que se presenta en estas clases es para evaluar el estado de la prueba en una iteración determinada: si se está levantando de la silla, si está caminando, si acaba de recorrer los tres metros caminando, si está manteniendo el equilibrio, si acaba de completar cinco sentadillas, etc. En este momento también se guarda el estado y posición de las articulaciones del esqueleto en la clase *DataTestLogger* para poder procesarlos al finalizar las pruebas. Para no guardar información redundante, el registro de las articulaciones solo se realiza cuando se detecta que está realizando la prueba, es decir, mientras el paciente se está preparando para empezar la prueba no se guardan las posiciones de sus articulaciones. Una vez que se ha evaluado el estado de la prueba y se han registrado las posiciones del cuerpo se actualiza la vista, si es necesario, para informar al paciente del estado de la prueba.

El proceso anterior se repite hasta que la prueba finaliza. Es decir, el sistema continua evaluando datos de posición de las articulaciones mientras que el paciente todavía no haya completado el *test*.

Cuando en una iteración se detecta que la prueba ha terminado, se cierra la cámara Kinect y se liberan los recursos asociados. En este momento, ya se pueden procesar los datos recogidos para completar los resultados de las pruebas como la puntuación o el número de pasos. Seguidamente, se envían los resultados procesados al servidor.

El Algoritmo 1 muestra el pseudocódigo descrito desde una perspectiva muy general. La funcionalidad que se incluye en la línea nueve del pseudocódigo es distinta para la prueba de velocidad de la marcha, la de sentarse y levantarse de la silla y la de equilibrio. En las siguientes secciones se va a describir cómo se ha implementado este paso del algoritmo.

Algorithm 1 Esquema algorítmico del seguimiento y evaluación de las pruebas físicas

```
1: Inicialización del sensor Kinect
2: while prueba no terminada do
3:     if primer frame then
4:         Obtención del primer Body registrado por la cámara
5:         Guardado de la posición en el vector de cuerpos del Body escogido
6:     else
7:         Obtención del Body cuyo índice en el vector ha sido fijado previamente
8:     end if
9:     Evaluación de la prueba a partir de las articulaciones del Body fijado en dicho instante
10:    Actualización de la vista según el nuevo estado de la prueba
11: end while
12: Cierre del sensor Kinect y liberación de los recursos asociados
13: Procesamiento de los datos recogidos y completado de los resultados de las pruebas
14: Envío de los resultados al servidor
```

Detalle del algoritmo de la prueba de velocidad de la marcha

Como se ha comentado anteriormente, en cada nuevo *frame* o iteración se debe evaluar la posición del paciente para comprobar que el *test* se está realizando correctamente. En este caso, la clase encargada de definir el estado de la prueba en cada nuevo *frame*, durante la prueba de velocidad de la marcha, es *GaitSpeedTest*. Esta clase hereda de la clase abstracta *FrailtyTest*, que le obliga a definir el método *EvaluateFrailtyTest(IReadOnlyDictionary<JointType, Joint> joints)*. Dicho método es el que se ejecuta cada vez que se recibe un nuevo *frame* durante la prueba de velocidad de la marcha y, como se puede observar, recibe como parámetro un diccionario con el nombre y las articulaciones del esqueleto registrado.

Realmente, solo con la posición de las articulaciones en un instante determinado no es posible definir algunos estados del *test*. Por ejemplo, es posible conocer cuando el paciente está en la línea de salida sabiendo únicamente su posición, pero no se podría saber si esta yendo al punto de partida para empezar la prueba o si, en realidad, ya ha empezado la prueba y acaba de pasar por la distancia de salida establecida. Por ello, tanto en esta prueba como en las dos siguientes es necesario guardar ciertos aspectos del estado de la prueba como por ejemplo si ya ha empezado el *test*, si se encontraba en la zona de salida, etc.

Entonces, para implementar el algoritmo de la prueba de velocidad de la marcha se definieron dos distancias que determinan el punto de inicio y final de la prueba. Los espacios adyacentes a estas líneas se denominaron *StartZone*, *TransitionZone* y *EndZone* tal y como se presenta en la Figura 4.3.

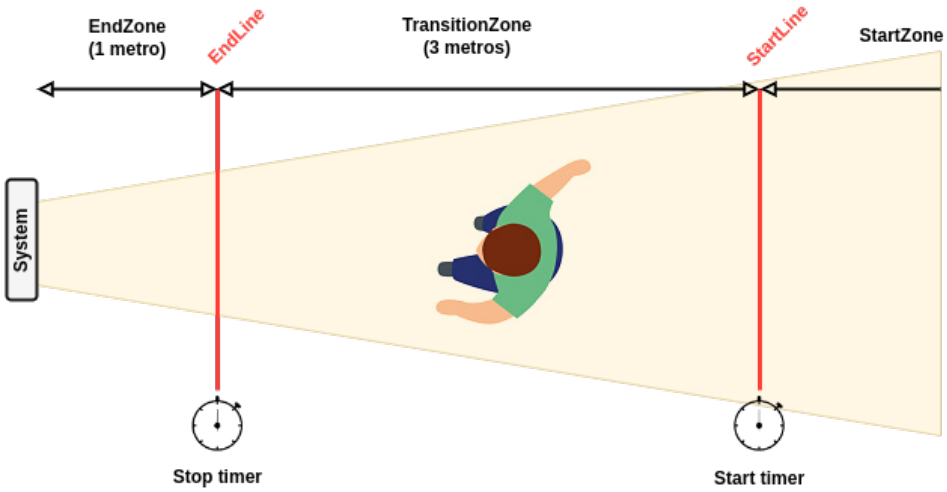


Figura 4.3: Digitalización de la prueba de velocidad de la marcha

Lo primero que se realiza en cada nuevo *frame* o iteración es calcular la distancia a la que se encuentra el paciente. Para ello, se recogen los valores de profundidad de cuatro *joints*: *SpineBase*, *SpineMid*, *SpineShoulder* y *Head*. De estos cuatro, se descartan aquellos valores que están siendo estimados por la cámara o que directamente no son visibles. De los restantes, se hace una media y este se considera el punto en el que se encuentra el paciente respecto al eje Z (profundidad respecto a la cámara). La Figura 4.4 representa las articulaciones y los respectivos nombres proporcionados por la cámara Kinect.

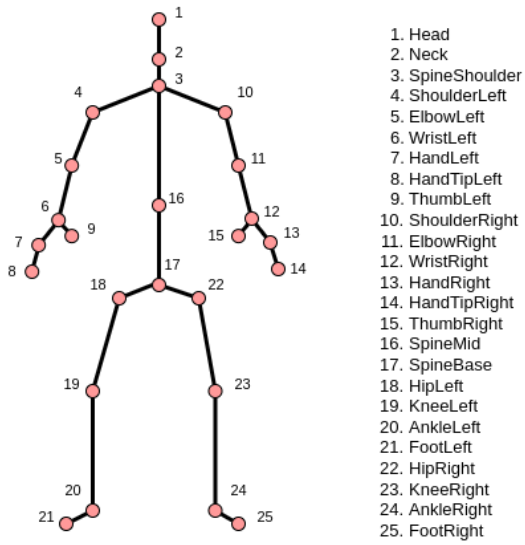


Figura 4.4: Articulaciones que proporciona el sensor Kinect

Una vez conocida la posición del paciente, solo es necesario identificar en que zona y estado se encuentra. Si el paciente se encuentra en la zona final o en la zona de transición y no había estado en la zona de inicio, se informa al paciente de que tiene que alejarse de la cámara para empezar la prueba de velocidad de la marcha. Durante estos momentos, como no está recorriendo los

tres metros válidos para la prueba, no se registran las posiciones de las articulaciones para luego procesarlas. Cuando el paciente llega a la zona de inicio, se le pide que, cuando esté listo, camine hacia la cámara. En el momento en el que se detecta que cruza la línea, da comienzo la prueba y se empieza a registrar el tiempo y la posición de las articulaciones de cada *frame* o instante. Cuando cruza la linea final se para el cronómetro, se dejan de registrar las articulaciones y se notifica al paciente de que ya ha terminado la prueba.

En el párrafo anterior, se ha descrito lo que sería el flujo normal a la hora de realizar esta prueba, pero se deben contemplar otros escenarios como por ejemplo cuando el paciente había empezado la prueba y ya había estado en la zona de transición, pero en estos momentos su posición es detectada en la zona de inicio. Esto significa, que el paciente no ha completado la prueba y ha vuelto al punto de partida. En este caso, se reinicia el cronómetro, se borran los datos de las articulaciones registrados y si notifica al paciente que vuelva a repetir la prueba.

El algoritmo implementado para evaluar el estado de la prueba de velocidad de la marcha en cada *frame* o iteración se presenta en Algorithm 2.

Algorithm 2 Algoritmo para la evaluación de la prueba de velocidad de la marcha

```
1: Obtención de la posición del paciente
2: if bodyInEndZone() and testIsRunning then                    ▷ Si completa el test
3:     Se para el cronómetro
4:     Se registra el estado de las articulaciones en este instante
5:     Se genera el resultado de la prueba con la información básica
6:     Se actualiza la información para la vista
7:     Se finaliza el test
8: else if bodyInTransitionZone() and testIsRunning then        ▷ Si está en medio del test
9:     Se registra el estado de las articulaciones en este instante
10: else if bodyInTransitionZone() and wasInStartZone then      ▷ Si acaba de empezar el test
11:     Se inicia el cronómetro
12:     testIsRunning = true
13:     wasInStartZone = false
14:     Se registra el estado de las articulaciones en este instante
15:     Se actualiza la información para la vista
16: else if bodyInStartZone() and testIsRunning then            ▷ Si vuelve a de empezar el test
17:     Se restablece el cronómetro
18:     testIsRunning = false
19:     wasInStartZone = true
20:     Se borran los datos de las articulaciones guardadas
21:     Se actualiza la información para la vista
22: else if bodyInStartZone() and not wasInStartZone then       ▷ Si está para empezar el test
23:     Se restablece el cronómetro
24:     wasInStartZone = true
25:     Se actualiza la información para la vista
26: end if
```

Detalle del algoritmo de la prueba de sentadillas

La forma de evaluar el estado del *test* durante la prueba de sentadillas sigue un esquema parecido al de la prueba de velocidad de la marcha. En primer lugar, se obtiene del diccionario *joints* las posiciones de las articulaciones que son necesarias. En este caso, se optó por deducir si el paciente está sentado o no en función del grado de flexión de sus dos rodillas. Para calcular el grado de flexión de cada rodilla era necesario recoger la posición de la cadera, rodilla y tobillo de ambas piernas. De esta forma, a partir de operaciones con vectores es posible estimar el grado de flexión de las rodillas del paciente según la siguiente expresión matemática:

$$\cos\alpha = \frac{\vec{v}\cdot\vec{u}}{|\vec{v}|\cdot|\vec{u}|} \tag{4.1}$$

Por tanto, primero es necesario definir los dos vectores $\vec{u}$ y $\vec{v}$. El primero se forma con la posición de la cadera y la posición de la rodilla. El segundo se forma con la posición de la rodilla y la posición del tobillo. Seguidamente se realiza el producto escalar de $\vec{u}$ y $\vec{v}$ y se calcula para cada vector su norma, cuya expresión es la siguiente:

$$\vec{AB} = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2} \tag{4.2}$$

De la misma forma que ocurría en la prueba anterior, el grado de articulación de las rodillas del paciente no permite conocer que acción concreta está realizando el usuario. Por ejemplo, en este caso, se puede detectar que el paciente tiene ambas rodillas con una flexión aproximada de 120 grados en un determinado *frame*, pero únicamente con esta información no se puede determinar si el paciente se está levantando de la silla o, por el contrario, se está sentando. Por tanto, será necesario hacer uso de variables de clase para guardar el número de sentadillas que ya ha realizado, si estaba en pie en el *frame* previo, etc.

El pseudocódigo que se muestra en Algorithm 3 describe el algoritmo implementado en la función *EvaluateFrailtyTest* en la clase *GetUpTest*.

Detalle del algoritmo de la prueba de equilibrio

Finalmente, en lo que respecta a cómo se evalúa en cada nuevo *frame* la prueba de equilibrio, cabe destacar ciertos aspectos.

En primer lugar, se tomó la decisión de valorar si el paciente se encuentra en posición de equilibrio utilizando las articulaciones del tobillo y pie de cada pierna ya que parecía lo más evidente. Esto planteaba un gran inconveniente y es que la posición del pie que proporciona el sensor es frecuentemente mal estimada o confundida con el suelo, generando una gran incertidumbre y variabilidad en el resultado. La solución que se optó fue pedir al paciente que se colocara con los pies en línea mirando de frente a la cámara. De esta forma, si la cámara no es capaz de detectar una de las dos piernas, se considera que el paciente se encuentra en posición de equilibrio ya que se deduce que uno de los pies se encuentra en línea detrás del pie visible.

Algorithm 3 Algoritmo para la evaluación de la prueba de levantarse y sentarse de la silla

```
1: Obtención de los grados de flexión de la rodilla derecha
2: Obtención de los grados de flexión de la rodilla izquierda
3: if not testIsRunning and not isOnCountdown and bodyInStartingPosition() then ▷ Si está preparado para empezar el test
4:     isOnCountdown = true
5:     testIsRunning = false
6:     Se inicia la cuenta atrás
7:     Se actualiza la información para la vista
8: else if isOnCountdown and testTime >= 3.0 then ▷ Si termina la cuenta atrás
9:     squats = 0
10:    isOnCountdown = false
11:    testIsRunning = true
12:    Se inicia el cronómetro
13:    Se registra el estado de las articulaciones en este instante
14:    Se actualiza la información para la vista
15: else if isOnCountdown then ▷ Si está en la cuenta atrás para empezar el test
16:    Se actualiza la información para la vista
17: else if testIsRunning and testTime ¿= 60.0 then ▷ Si no es capaz de terminar el test
18:    testIsRunning = false
19:    isOnCountdown = false
20:    Se para el cronómetro
21:    Se genera el resultado de la prueba con la información básica
22:    Se finaliza el test
23: else if testIsRunning then ▷ Si está en el transcurso del test
24:    Se registra el estado de las articulaciones en este instante
25:    Se actualiza la información para la vista
26:    if bodyIsUpright() and not alreadyUpright then ▷ Se ha levantado
27:        alreadyUpright = true
28:        squats++
29:        if squats == 5 then
30:            testIsRunning = false
31:            isOnCountdown = false
32:            Se para el cronómetro
33:            Se genera el resultado de la prueba con la información básica
34:            Se finaliza el test
35:        end if
36:    else if bodyIsSeated() and alreadyUpright then ▷ Se ha sentado
37:        alreadyUpright = false
38:    end if
39: end if
```

En segundo lugar, para reducir el error de la prueba también se evaluaba la posición de equilibrio en función de un valor de confianza. Este valor de confianza no era más que una variable del tipo *int* que, por cada *frame* en el que se detectaba que el paciente estaba en posición de equilibrio, se sumaba uno. Por el contrario, si detectaba que no estaba en equilibrio se restaba uno. El número máximo establecido que podía alcanzar el valor era cinco, y el mínimo cero. De esta forma, en una iteración concreta se podía determinar con mayor seguridad si el paciente se encontraba en posición de equilibrio: cuando el valor de confianza era cinco.

De la misma forma que para las dos anteriores, el Algorithm 4 presenta el pseudocódigo para la evaluación de la prueba de equilibrio.

Algorithm 4 Algoritmo para la evaluación de la prueba de equilibrio

```
1: if not testIsRunning and not isOnCountdown and bodyInStartingPosition() then ▷ Si está preparado para empezar el test
2:     isOnCountdown = true
3:     testIsRunning = false
4:     Se inicia la cuenta atrás
5:     Se actualiza la información para la vista
6: else if isOnCountdown and testTime >= 6.0 then ▷ Si termina la cuenta atrás
7:     isOnCountdown = false
8:     testIsRunning = true
9:     Se inicia el cronómetro
10:    Se registra el estado de las articulaciones en este instante
11:    Se actualiza la información para la vista
12: else if isOnCountdown then ▷ Si está en la cuenta atrás para empezar el test
13:    Se actualiza la información para la vista
14: else if testIsRunning and bodyLosesBalance() then ▷ Si pierde el equilibrio
15:    testIsRunning = false
16:    Se para el cronómetro
17:    Se genera el resultado de la prueba con la información básica
18:    Se actualiza la información para la vista
19:    Se finaliza el test
20: else if testIsRunning and balanceTime >= 10.0 then ▷ Supera los diez segundos
21:    testIsRunning = false
22:    Se para el cronómetro
23:    Se genera el resultado de la prueba con la información básica
24:    Se actualiza la información para la vista
25:    Se finaliza el test
26: else if testIsRunning and balanceTime < 10.0 then ▷ Se mantiene en equilibrio
27:    Se registra el estado de las articulaciones en este instante
28: end if
```

4.1.4. Cálculo de los pasos durante la prueba de velocidad de la marcha

Una de las funcionalidades adicionales en el proyecto ha sido el cálculo de los pasos durante la prueba de velocidad de la marcha. Esta funcionalidad no era un objetivo principal del proyecto y, por esta razón, fue implementada en el último *sprint* cuando se habían completado todas las historias de usuario.

La idea de esta pequeña funcionalidad extra era proporcionar algo más de valor e información a los resultados de las pruebas, en este caso a la prueba de velocidad de la marcha.

Como se ha comentado varias veces a lo largo de este capítulo, durante el transcurso de las pruebas físicas se registraban los datos de las posiciones del paciente para, una vez liberados los recursos de la Kinect, procesar los datos e intentar extraer información relevante sin colapsar los recursos del sistema. Una de sus utilidades es precisamente para calcular el número de pasos que el paciente ha necesitado tomar para recorrer los tres metros.

Para abordar cómo realizar el cálculo de los pasos, fue necesario realizar un pequeño análisis sobre la evolución de las posiciones de distintas articulaciones durante el paseo. La Figura 4.5 muestra un gráfico muy representativo de cómo evoluciona la profundidad de los tobillos respecto a la cámara durante la prueba de velocidad de la marcha. A partir de esta información fue relativamente sencillo proponer un algoritmo con coste lineal que aproximara el número de pasos que el paciente había realizado. El pseudocódigo del algoritmo implementado se muestra en el Algorithm 5.

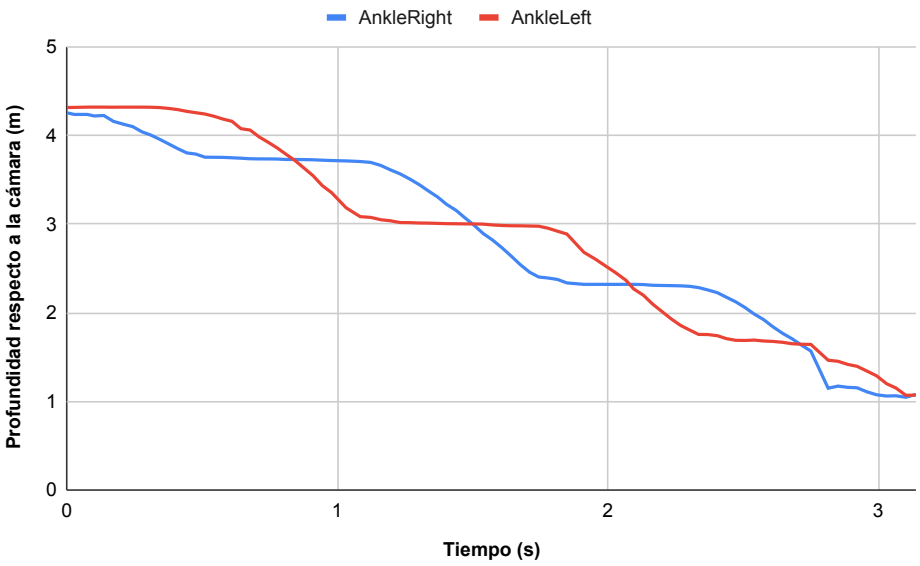


Figura 4.5: Evaluación de la profundidad de los tobillos durante el paseo

Algorithm 5 Algoritmo para el conteo de pasos tras la prueba de velocidad de la marcha

```
Se almacena en un vector las posiciones del pie izquierdo a lo largo de la prueba
Se almacena en un vector las posiciones del pie derecho a lo largo de la prueba
steps = 0
for i = 1; i < tamañoVectores; i++ do
    if rightFootDepthValues[i] >= leftFootDepthValues[i] and
rightFootDepthValues[i - 1] < leftFootDepthValues[i - 1] then
        steps++
    else if rightFootDepthValues[i] <= leftFootDepthValues[i] and
rightFootDepthValues[i - 1] > leftFootDepthValues[i - 1] then
        steps++
    end if
end for
return steps
```

4.1.5. Cálculo de la altura durante la prueba de sentadillas

Otra funcionalidad adicional que puede aportar información relevante a los resultados de las pruebas es la altura media de las sentadillas que ha realizado el paciente durante la prueba de levantarse y sentarse de la silla. Al igual que para el cálculo de los pasos, está información se obtiene una vez concluido el test.

La forma de abordar el algoritmo se puede deducir fácilmente a partir de la Figura 4.6, donde se puede apreciar como varía la altura del cuello y la cabeza con respecto a la Kinect durante la prueba de levantarse y sentarse de la silla.

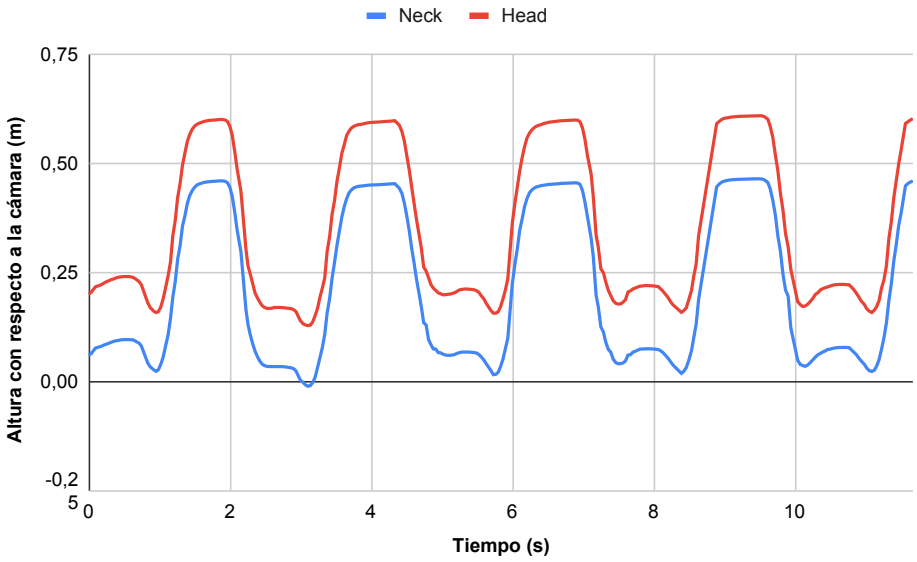


Figura 4.6: Evaluación de la altura durante la prueba de sentadillas

Lo solución adoptaba consistía sencillamente en hacer una media de todos los máximos

locales, otra media de todos los mínimos locales y finalmente restar ambos valores. Cabe destacar que fue suficiente tomar las posiciones que proporcionaba la cabeza.

4.1.6. Comunicación y acoplamiento de los subsistemas

Además de los algoritmos implementados para realizar el seguimiento del paciente durante el transcurso de las pruebas físicas, existe otro aspecto relevante y muy característico de este sistema desarrollado a lo largo de la estancia en prácticas. Se trata de cómo se comunica esta aplicación con el resto de los subsistemas que componen FTAS. Como ya se ha mencionado previamente, la aplicación se comunica con el servidor mediante el protocolo HTTP y también haciendo uso de la tecnología MQTT.

La comunicación mediante el protocolo HTTP ha sido utilizada para obtener el detalle del paciente a partir de un DNI y para registrar los resultados de las pruebas físicas. En el primer caso se realiza una petición del tipo *GET* enviando el identificador del paciente como parámetro. En caso de que el servidor responda y encuentre el recurso, se espera que devuelva la información del paciente en formato JSON. En el segundo caso, cuando un paciente completa las pruebas, se generan las clases DTO para ser directamente transformadas en cadenas siguiendo el formato JSON. De esta forma son adjuntadas como cuerpo de la petición *POST* para que los resultados sean guardados en el servidor.

Por su parte, la comunicación mediante el protocolo MQTT se realiza para permitir la identificación del paciente con el escaneo el código QR que se muestra en la pantalla de inicio de la aplicación. Este proceso se divide en las fases que se detallan a continuación.

La primera fase ocurre cuando la aplicación descrita en este documento arranca. Este proceso consiste en que el dispositivo que aloja la cámara Kinect se suscribe a un «*topic*» para que sea notificado cuando un paciente escanee el código QR. En MQTT, un *topic* no es más que un canal por el que el dispositivo que se suscribe puede enviar y recibir mensajes. La Figura 4.7 muestra la comunicación de esta primera fase del proceso de identificación mediante el escaneo del código QR.

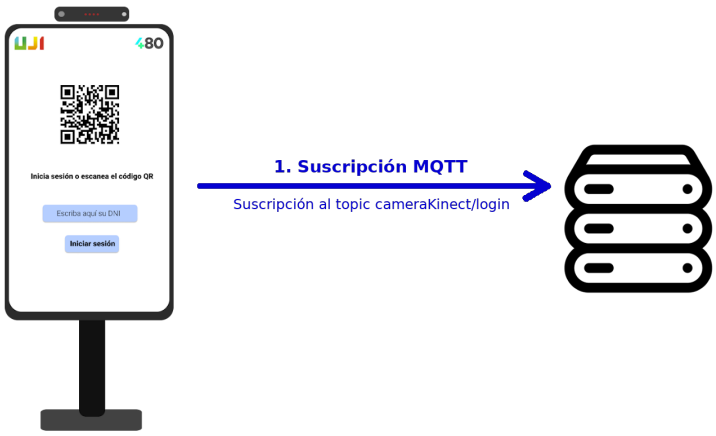


Figura 4.7: Primera fase del proceso de identificación con el código QR

La segunda fase se produce cuando el dispositivo móvil del paciente escanea el código QR de la aplicación. El código QR generado es la representación del identificador único de cada dispositivo que aloja una cámara Kinect y, por tanto, al escanear este código se obtiene una cadena de texto que identifica dicho sensor. La Figura 4.8 muestra la acción de escaneo para proceder con la identificación del paciente.

Figura 4.8: Segunda fase del proceso de identificación con el código QR

Una vez la aplicación móvil del paciente ha obtenido el identificador del dispositivo que ha escaneado, es posible asociar un paciente (ya que este se habrá registrado en la aplicación) con un dispositivo que aloja el sensor Kinect para realizar las pruebas físicas. Este proceso abarca lo que sería la tercera fase, donde el dispositivo móvil envía una petición HTTP al servidor con el identificador del usuario y del dispositivo escaneado. Cabe recordar que tanto esta tercera fase como la anterior son parte de la aplicación móvil y, por tanto, no ha sido desarrollado por el autor de este documento, aunque es necesario detallar este proceso para aclarar el funcionamiento de esta forma de identificación. La Figura 4.9 muestra la tercera fase del proceso de identificación mediante el escaneo del código QR.

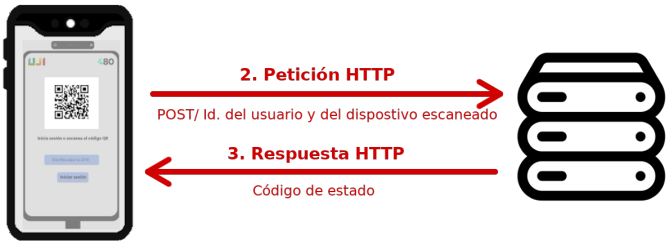


Figura 4.9: Tercera fase del proceso de identificación con el código QR

En este punto, el servidor ya conoce la asociación entre el sensor Kinect y el paciente. Ahora, queda notificar al dispositivo que aloja la cámara que su código QR ha sido escaneado. Para ello, el servidor publica por el canal de comunicación en el que el dispositivo se suscribió en la primera fase, el identificador del paciente. A partir de aquí, la aplicación que se ha desarrollado a lo largo de este proyecto ya conoce que un paciente ha escaneado su código QR y el identificador de dicho paciente. La Figura 4.10 muestra la comunicación entre el dispositivo y el servidor mediante MQTT descrito en esta cuarta fase.

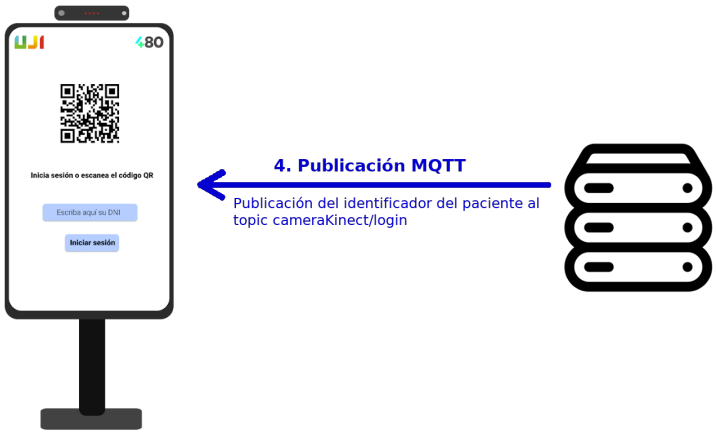


Figura 4.10: Cuarta fase del proceso de identificación con el código QR

La quinta y última fase de comunicación se realiza para conocer los datos del paciente y así, además de poder asociar los resultados de las pruebas con un paciente, poder dar la bienvenida a la aplicación mostrando el nombre de la persona que ha escaneado el código QR. Para ello es necesario recoger la información del paciente de la misma forma que se haría cuando un paciente se identifica introduciendo el DNI de forma manual: con una petición *GET* siguiendo el protocolo de comunicación HTTP. La Figura 4.11 muestra la última fase del proceso de identificación con el código QR.

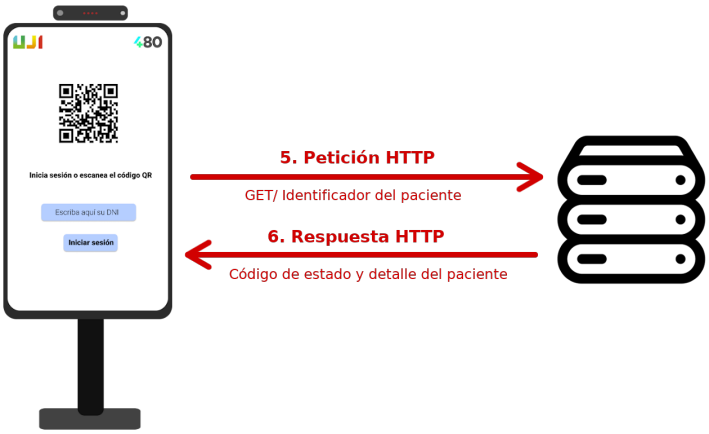


Figura 4.11: Quinta fase del proceso de identificación con el código QR

4.2. Verificación y validación

En esta sección se va a detallar como se ha verificado y validado el software desarrollado.

4.2.1. Verificación de las pruebas físicas

En primer lugar, se verificaron los resultados que recoge la Kinect a la hora de realizar las pruebas físicas. Como se comentaba al principio del documento, uno de los objetivos del proyecto era proporcionar un método alternativo al tradicional, y de mayor precisión, para la estimación del tiempo que requiere al paciente realizar cada prueba.

Esta verificación se realizó con un pequeño grupo de voluntarios y compañeros, donde la prueba de velocidad de la marcha destacó por presentar unos tiempos muy semejantes a los que tomaba el humano. La aplicación también conseguía unos resultados muy parecidos frente a los humanos en la prueba de levantarse y sentarse de la silla. En cambio, la prueba de equilibrio presentaba una tasa de fallo mucho mayor y, en muchos casos, bastante dispar a los tiempos que se tomaban de forma manual.

Dada la muestra y la cantidad de pruebas realizadas no se puede demostrar las afirmaciones anteriores, pero si parece que el objetivo de proporcionar un método alternativo al tradicional, y puede que de mayor precisión, se haya conseguido para dos de las tres pruebas. Realizar esta afirmación implicaría verificar el sistema de forma exhaustiva con un sistema alternativo y validado, que sea capaz de tomar el tiempo que toma a los pacientes realizar cada una de las pruebas.

4.2.2. Pruebas de sistema

Para acoplar la aplicación con los otros subsistemas que componen FTAS fue necesario realizar pruebas de sistema. En este punto, fue fundamental la comunicación con el desarrollador del servidor y del segundo subsistema: Arturo Gascó. Por un lado, se comprobó en repetidas ocasiones la identificación del paciente mediante el escaneo del código QR. También se comprobó que los resultados de las pruebas físicas eran recibidos en el servidor y guardados como se esperaba en la base de datos.

4.2.3. Análisis estático del código

Finalmente, el código desarrollado de la aplicación fue habitualmente analizado de forma estática, atendiendo a no generar clases con demasiada responsabilidad y complejidad para que fuera más mantenible en el futuro y no planteara grandes problemas a la hora de añadir nueva funcionalidad. La Figura 4.12 muestra el análisis estático del código del proyecto una vez finalizada la estancia en prácticas. La herramienta con la que se ha realizado el análisis estático es la misma que se ha utilizado como entorno de programación: *Visual Studio* [10].

Como se puede observar en la Figura 4.12, el índice de mantenimiento de todas las clases implementadas son considerados óptimos. Este valor se calcula en función del número de líneas, el nivel de complejidad ciclomática y el volumen de Halstead. Este último es calculado a partir de la variabilidad y cantidad de operadores y operandos en el código [7]. Los valores calculados oscilan entre 0 y 100, donde los valores próximos a 100 expresan código fácilmente mantenible.

Como también se puede apreciar, todas las clases y códigos presentan niveles de complejidad ciclomática relativamente bajos. La complejidad ciclomática expresa la cantidad de posibles caminos o flujos del programa en cuestión. Dicho de otra forma, este nivel indica la complejidad estructural del código y, por tanto, cuanto mayor es el nivel de complejidad mas dificultades presenta a la hora de mantenerlo [16].

Jerarquía	Índice de mantenimiento	Complejidad ciclomática	Líneas de código fuente
FtasFrontendWpf (Debug)	79	462	3.115
FtasFrontendWpf	75	9	133
App	75	4	55
MainWindow	75	5	63
FtasFrontendWpf.Model.BasicData	90	30	131
BodyJointsOnInstant	75	9	60
JointStatus	94	5	20
Patient	100	9	12
Position	92	7	26
FtasFrontendWpf.Model.Dtos	91	32	83
BalanceTestResultDTO	94	3	10
GaitSpeedTestResultDTO	91	9	19
GetUpTestResultDTO	91	9	19
TestResultDTO	91	11	23
FtasFrontendWpf.Model.FrailtyTests	70	124	689
BalanceTest	62	30	153
FrailtyTest	90	22	111
GaitSpeedTest	65	36	198
GetUpTest	64	36	212
FtasFrontendWpf.Model.Managers	77	95	656
BodyPositionDetector	70	9	30
DataTestLogger	92	5	30
FrailtyTestDriver	73	9	63
HttpManager	81	5	38
KinectManager	82	6	55
KinectManagerSingleton	84	4	32
MqttManager	76	9	67
Navigator	83	8	49
Painter	72	6	42
QrManager	82	2	12
SoundManager	78	3	31
SquatRomCalculator	71	14	60
StepCounter	66	9	42
TestResultStorer	70	6	57
FtasFrontendWpf.Model.StatusTests	88	6	106
BalanceCountdownStatus	88	2	34
CountdownStatus	89	2	29
SquatsStatus	89	2	34
FtasFrontendWpf.Model.TestResults	82	70	332
BalanceTestResult	81	8	41
GaitSpeedTestResult	76	26	113
GetUpTestResult	81	19	92
TestResult	92	17	74
FtasFrontendWpf.Properties	80	6	101
Resources	75	5	56
Settings	85	1	15
FtasFrontendWpf.View	72	90	884
FrailtyTestAboutPage	72	14	155
FrailtyTestPage	66	39	318
HomePage	69	16	168
WelcomeAssistedPage	80	15	132
WelcomePage	74	6	74

Figura 4.12: Métricas de calidad y complejidad del código implementado

Capítulo 5

Conclusiones

En lo que respecta al proyecto, el resultado ha sido muy gratificante ya que fue posible alcanzar los objetivos propuestos. Tras la estancia en prácticas se consiguió desarrollar un producto mínimo viable, que permite realizar tres pruebas físicas para digitalizar el proceso y recogida de datos en la detección de fragilidad en personas de edad avanzada.

Además, este trabajo abre un amplio abanico de posibilidades a la hora de futuros trabajos. Por ejemplo, sería ideal desarrollar un método o algoritmo alternativo para la prueba de equilibrio de mayor precisión. También sería muy interesante mejorar la interacción con el sistema aprovechando las posibilidades que ofrece una cámara 3D y, finalmente, también sería vital validar el sistema con psicólogos y verificar con un método alternativo la toma de resultados para así poder trasladarlo a centros de salud y entornos reales.

Bajo mi punto de vista, se trata de uno de los proyectos más interesantes e innovadores que he realizado. Por un lado, no estaba familiarizado con ninguna de las tecnologías utilizadas en el desarrollo del sistema. Este gran reto me ha permitido conocer el lenguaje de programación C#, que es ampliamente utilizado, y el protocolo de comunicación MQTT, que ha demostrado ser potente y versátil. Además, el desarrollo de software con la cámara Kinect v2 me ha permitido tener la nueva y entretenida experiencia de trabajar con cámaras 3D. Algo que sin duda está teniendo mucho auge en la actualidad.

Por otro lado, he podido trabajar y aprender sobre un concepto desconocido para mí y muy interesante: la fragilidad. Creo que es muy importante ser consciente de los problemas a los que se enfrentan nuestros mayores para, siempre que sea posible, aportar nuestro granito de arena en mejorar su calidad de vida.

En lo que se refiere al ámbito profesional, también puedo decir que he tenido una experiencia muy enriquecedora. Agradezco mucho la oportunidad que me ha proporcionado la empresa Cuatroochenta para realizar la estancia en prácticas ya que he sido capaz de conocer e integrarme en el entorno profesional. También cabe mencionar la ayuda y colaboración de mis compañeros, en especial la de Arturo Gascó, con quien sin duda también he forjado una gran amistad.

En el ámbito académico, me gustaría destacar el aprendizaje de algunas de las asignaturas

cursadas a lo largo de estos cuatro años que me han permitido aportar valor a este proyecto como Metodologías Ágiles, Fundamentos de la Ingeniería del Software, Análisis de Software, Diseño de Software, Paradigmas de Software, Algoritmia y otras muchas más que me han permitido complementar y adquirir una base para estos conocimientos.

En resumen, creo que ha sido una de las experiencias más enriquecedoras y gratificantes de mi experiencia académica, profesional y personal. Me alegra haber mejorado mis aptitudes en lo que se refiere al trabajo en equipo, comunicación y formación, pero sobre todo me alegra mucho haber podido ser parte del desarrollo de un sistema tan complejo y amplio que pretende mejorar la calidad de vida y salud de las personas.

Bibliografía

[1] Jira — Issue and Project Tracking Software. https://www.atlassian.com/software/jira, 2022. Accedido: 15-05-2022.

[2] Laisi Cai, Ye Ma, Shuping Xiong, and Yanxin Zhang. Validity and Reliability of Upper Limb Functional Assessment Using the Microsoft Kinect V2 Sensor. *Applied Bionics and Biomechanics*, 2019:1–14, 2019.

[3] Andrew Clegg, John Young, Steve Iliffe, Marcel Olde Rikkert, and Kenneth Rockwood. Frailty in elderly people. *The Lancet*, 381(9868):752–762, 2013.

[4] Cuatroochenta — desarrollo e implantación de soluciones cloud. https://cuatroochenta.com/. Accedido: 20-03-2022.

[5] Figma: the collaborative interface design tool. https://www.figma.com/, 2022. Accedido: 15-05-2022.

[6] Martin Fowler. bliki: LocalDTO. https://martinfowler.com/bliki/LocalDTO.html, 10 2004. Accedido: 09-06-2022.

[7] IBM. Halstead Metrics. https://www.ibm.com/docs/en/rtr/8.0.0?topic=SSSHUF_8.0.0/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm, 2022. Accedido: 12-06-2022.

[8] Instituto Nacional de Estadística (INE). Proyecciones de Población 2020-2070. Technical report, Instituto Nacional de Estadística (INE), 09 2020.

[9] Kinect — Windows app development. https://developer.microsoft.com/en-us/windows/kinect/, 2022. Accedido: 13-04-2022.

[10] Microsoft. Visual Studio 2022 — Descargar gratis. https://visualstudio.microsoft.com/es/vs/, 05 2022. Accedido: 15-06-2022.

[11] MQTT - The Standard for IoT Messaging. https://mqtt.org/. Accedido: 17-03-2022.

[12] Postman Api Platform. https://www.postman.com/, 2022. Accedido: 15-05-2022.

[13] Refactoring.Guru. Singleton. https://refactoring.guru/design-patterns/singleton, 01 2022. Accedido: 3-06-2022.

[14] Scrum - The Home of Scrum. https://www.scrum.org/. Accedido: 10-04-2022.

[15] Salario para Programador en España - Salario Medio. https://es.talent.com/salary?job=Programador, 2022. Accedido: 14-05-2022.

[16] Calculate code metrics - Visual Studio (Windows). https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022, 06 2022. Accedido: 15-06-2022.

[17] What is WPF? - Visual Studio (Windows). https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2022, 04 2022. Accedido: 14-05-2022.