Trabajo Fin de Grado
Ingeniería de Telecomunicación

Aplicación móvil y servicio REST de apoyo a la toma de decisiones clínicas en el tratamiento del dolor crónico

Autor: Celia Llanes Rodríguez
Tutor: Jorge Calvillo Arbizu

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Aplicación móvil y servicio REST de apoyo a la toma de decisiones clínicas en el tratamiento del dolor crónico

Autor:
Celia Llanes Rodríguez

Tutor:
Jorge Calvillo Arbizu
Profesor Ayudante Doctor

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Carrera: Aplicación móvil y servicio REST de apoyo a la toma de decisiones clínicas en el tratamiento del dolor crónico

Autor: Celia Llanes Rodríguez

Tutor: Jorge Calvillo Arbizu

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos

A los que ya no están

Agradecimientos

En primer lugar, quiero agradecer a todos mis profesores por todo lo que me han enseñado y he podido aprender de ellos. En especial agradecer a Jorge Calvillo, mi tutor de este TFG, que en el primer año del grado fue uno de los profesores que más me motivó por cómo transmitía sus conocimientos en sus clases de Fundamentos de Programación II. Hiciste que me encantara el mundo de la programación. Años después, has confiado en mi para realizar este proyecto y no puedo estar más agradecida.

Con el desarrollo de este trabajo de fin de grado pongo fin a una de las etapas más importantes de mi vida. Aunque ha sido difícil y a veces pensaba que no iba a terminar nunca, ha sido una etapa en la que he sido muy feliz y que siempre recordaré con una sonrisa. Y sí, esto último es sin duda gracias a todas esas personas que han estado siempre ahí.

A Andrea D., Verónica, Víctor, Elena, Andrea J., David, Víctor, Luis, Paco, y Paula; mis amigos de siempre. Gracias por compartir conmigo tantos momentos y toda vuestra magia. Sois personas únicas y enormes, no hacéis más que generar ilusión y motivación en mí y en todos. Espero que sigamos creando momentos maravillosos siempre.

A Jacinto, Pablo, Juanjo, Fernando, Marta, Manolo, Rubio, Vicente, Julio y todos los que os habéis cruzado en estos años y habéis compartido toda esta etapa conmigo. Vosotros sois uno de los regalos que me ha dado estos estudios de ingeniería. Gracias por hacerlo todo más fácil, por aguantar mis lloros diarios y apoyarme siempre. Os habéis convertido en una parte muy importante para mí, espero que nuestros caminos sigan unidos por esta carrera y por una fuerte amistad.

A mi familia, tengo la suerte de tener a unas de las mejores que existe. A todos gracias por apoyarme en cada momento y cuidarme siempre. En especial a mis padres, a mi abuela y a mi hermana Paola… gracias por confiar en mi incondicionalmente, por vuestro cariño y por aguantarme diariamente. Sois las personas que más influís en mí y no puedo estar más agradecida por todo. Conseguís lo imposible y sois maravillosos, espero que nunca se pierda la complicidad que hay entre nosotros.

Y por supuesto a Ángel, agradecer todo su apoyo en estos últimos años, su confianza en mí y su capacidad de hacerme sonreír en cualquier momento, contigo todo es más fácil.

Sin duda, jamás habría podido llegar hasta aquí sin cada uno de vosotros. Por eso este logro es también vuestro. Os quiero con todo mi corazón, mil millones de gracias de nuevo.

Celia Llanes Rodríguez

Sevilla, 2022

Resumen

En los últimos años, el uso de las tecnologías se está multiplicando y está cambiando nuestra forma de trabajar. Cada vez son más los sectores que optan por una transformación digital movidos por las numerosas ventajas que esta puede llegar a ofrecer, entre las que destacan la optimización del tiempo, la automatización de procesos y la disminución de errores humanos.

En el sector sanitario las nuevas tecnologías están suponiendo una revolución y son cada vez más necesarias para garantizar una sanidad de calidad ya que ofrecen nuevos productos y servicios, así como mejoras de los ya existentes. Por ello, en el presente trabajo se contribuirá a la digitalización de la Unidad del Dolor del Hospital Universitario Virgen del Rocío de Sevilla. Dicha unidad está formada por el equipo de profesionales de la salud especializados en la ayuda y el cuidado de los pacientes con dolor. Se encargan de realizar diagnósticos y de proponer un plan de tratamiento con el fin de recuperar el bienestar físico y emocional del paciente.

Gran parte de los tratamientos del dolor crónico dependen de bombas de infusión de medicamentos para ser suministrados. Estas bombas son cargadas con unas concentraciones específicas para cada paciente. Estas concentraciones se obtienen a través de fórmulas matemáticas que los profesionales del dolor deben manejar diariamente. Será muy importante que no se cometa ningún error en este proceso. Por ello, el objetivo de este trabajo es diseñar y desarrollar una herramienta software para la prevención de los errores de medicación en la fase de prescripción de medicamentos para el dolor crónico y ayudar a la toma de decisiones clínicas.

Para cumplir con el objetivo del trabajo, se han llevado a cabo entrevistas con los profesionales del dolor de las que se han detectado una serie de necesidades y problemas a resolver. Tras formalizar y analizar los requisitos, se procedió a diseñar un sistema con una arquitectura en tres capas que suponía crear una aplicación móvil con sistema operativo Android cuyos datos fueran persistidos en un servidor de base de datos PostgreSQL externo. Para la comunicación entre la aplicación móvil y la base de datos, existe un servidor intermedio que proporciona una API REST. Esta comunicación se realizará de manera controlada gracias a los tokens JWT. Como dato adicional, el servidor PostgreSQL se ha gestionado mediante un servidor PgAdmin, ambos desplegados mediante contenedores Docker. El resultado final ha sido una aplicación plenamente funcional que resuelve los problemas planteados por los profesionales entrevistados.

Abstract

In recent years, the use of technologies is increasing and changing the way we work. More and more sectors are opting for digital transformation, driven by the many advantages it can offer, including time optimization, process automation and reduced human errors.

In the healthcare sector, new technologies are bringing about a revolution and are increasingly necessary to guarantee quality healthcare, since they offer new products and services, as well as improvements to existing ones. For this reason, the present work will contribute to the digitalisation of the Pain Unit of the Virgen del Rocío University Hospital in Seville. This unit is made up of a team of health professionals specialized in the support and care of patients with pain. They oversee making diagnoses and proposing a treatment plan with the aim of recovering the patient's physical and emotional well-being.

Much of the chronic pain's treatment relies on drug infusion pumps to deliver medications. These pumps are loaded with specific concentrations for each patient. These concentrations are obtained through mathematical formulas that pain professionals must manage daily. It will be very important that no mistakes are made in this process. Therefore, the aim of this work is to design and develop a software tool for the prevention of medication errors in the prescription phase of chronic pain medication and to help clinical decision making.

To meet the objective of the work, interviews have been conducted with pain professionals from which a series of needs and problems to be solved have been detected. After formalizing and analysing the requirements, we proceeded to design a system with a three-layer architecture that involved creating a mobile application with Android operating system whose data were persisted in an external PostgreSQL database server. For the communication between the mobile application and the database, there is an intermediate server that provides a REST API. This communication will be done in a controlled way thanks to JWT tokens. Additionally, the PostgreSQL server has been managed by a PgAdmin server, both deployed using Docker containers. The result has been a fully functional application that solves the problems raised by the professionals interviewed.

Índice

Agradecimientos viii

Resumen xi

Abstract xiii

Índice xvi

Índice de Tablas xviii

Índice de Figuras xx

Notación xxii

1 Introducción 11
1.1 Motivación y objetivos 11
1.2 Metodología 12
1.3 Plan de trabajo 12

2 Estado del arte 15
2.1 Marco teórico 15
2.1.1 Bombas de infusion de medicamentos 15
2.2 Herramientas software 16
2.2.1 Tecnologías 16
2.2.2 Entorno de trabajo 21

3 Resultados 26
3.1 Análisis y definición de requisitos 26
3.1.1 Introducción 26
3.1.2 Proceso de identificación de necesidades de negocio 26
3.1.3 Dominio del problema 26
3.1.4 Debilidades de la situación actual 27
3.1.5 Necesidades de negocio 28
3.1.6 Descripción de subsistemas 28
3.1.7 Catálogo de requisitos 30
3.1.8 Casos de uso 39
3.2 Diseño del sistema 48
3.2.1 Diseño del modelo de datos 48
3.2.2 Arquitectura del sistema 50
3.2.3 Diseño de la aplicación REST 51
3.2.4 Diseño de la aplicación móvil 53
3.2.5 Diseño de la seguridad 54
3.2.6 Diseño del funcionamiento del proyecto 54
3.3 Implementación 57
3.3.1 Base de datos PostgreSQL 57
3.3.2 Servicio API REST 59
3.3.3 Aplicación móvil 68

4 Conclusiones y líneas futuras 77
4.1 Conclusiones 77

4.2 Líneas futuras *78*

Referencias **79**

Anexo I: Procesos de cálculo **81**

5.1 Problema de cálculo de mezclas *81*

5.2 Problema del cálculo de dosis *83*

ÍNDICE DE TABLAS

Tabla 1. Plan de trabajo 14
Tabla 2. Módulos de Spring Framework (Recurso en línea: [4]) 18
Tabla 3. DEB-01: Catálogo de medicamentos 27
Tabla 4. DEB-02: Ausencia de control externo al seleccionar dosis 27
Tabla 5. DEB-03: Cálculos manuales 27
Tabla 6. DEB-04: Ausencia de control externo al obtener resultados 28
Tabla 7. SD-01: Cálculo de mezclas 29
Tabla 8. SD-02: Dosímetro 29
Tabla 9. SD-02: Catálogo de fármacos 29
Tabla 10. SD-04: Gestión de usuarios 29
Tabla 11. RINF-001:Información de un usuario 30
Tabla 12. RINF-002: Información de un hospital 30
Tabla 13. RINF-003: Información de un fármaco 31
Tabla 14. RINF-004: Información de roles 31
Tabla 15. RINF-005: Información cálculo de mezclas 32
Tabla 16. RINF-006: Información cálculo de dosis 32
Tabla 17. RINF-007: Información registro de fármacos 32
Tabla 18. RINF-008: Información edición de fármacos 32
Tabla 19. RINF-009: Información registro de un usuario 33
Tabla 20. RINF-010: Información inicio de sesión 33
Tabla 21. RINF-011: Información resumida fármaco 33
Tabla 22. RINF-012: Información receta 34
Tabla 23. RN-001: Restricción usuarios y hospitales 34
Tabla 24. RN-002: Restricción usuarios y roles 34
Tabla 25. RN-003: Restricción fármacos y hospitales 34
Tabla 26. RN-004: Identificación única de los usuarios en el sistema 35
Tabla 27. RN-005: Asociación de roles de usuarios 35
Tabla 28. RF-001: Obtención del catálogo de fármacos hospital 35
Tabla 29. RF-003: Inserción de fármacos en el catálogo de un hospital 36
Tabla 30. RF-003: Búsqueda de un fármaco dentro del catálogo 36
Tabla 31. RF-004: Consulta de los detalles de un fármaco 36
Tabla 32. RF-005: Modificación de la información de un fármaco 36
Tabla 33. RF-006: Eliminación de un fármaco del catálogo 37
Tabla 34. RF-007: Registro de un usuario 37

Tabla 35. RF-008: Autenticación de usuarios 37
Tabla 36. RF-009: Autorización de usuarios 37
Tabla 37. RF:012: Asociación de un rol a un usuario 38
Tabla 38. RF:011:Cálculo de mezclas 38
Tabla 39. RF:012: Cálculo de nueva dosis 38
Tabla 40. RNF-001: Interfaz de usuario (usabilidad) 38
Tabla 41. RNF-002: Compatibilidad (portabilidad) 38
Tabla 42. RNF-003: Fiabilidad de los cálculos 39
Tabla 43. Contraseñas cifradas 39
Tabla 44. ACT-01: Personal clínico 41
Tabla 45. ACT-02: personal clínico gestor 41
Tabla 46. CU-001: Cálculo de mezclas 42
Tabla 47. CU-002: Cálculo de dosis 42
Tabla 48. CU-003:Visualización del catálogo de fármacos 43
Tabla 49. CU-004: Visualización de un fármaco de forma detallada 44
Tabla 50. CU-005: Eliminar fármaco del catálogo de fármacos 44
Tabla 51. CU-006:Modificar fármaco del catálogo de fármacos 45
Tabla 52. CU-007: Registro de nuevo fármaco 46
Tabla 53. CU-008: Inicio de sesión 46
Tabla 54. CU-009: Registrar nuevo usuario 47
Tabla 55. Modelo relacional del sistema 49
Tabla 56. Definición de los endpoints del servicio REST 53
Tabla 57. Resumen cálculo de mezclas 83

ÍNDICE DE FIGURAS

Figura 2-1. Logo Android 16
Figura 2-2. Logo Java 16
Figura 2-3. Logo Docker 20
Figura 2-4. Logo PostgreSQL 21
Figura 2-5. IntelliJ IDEA 21
Figura 2-6. Android Studio 22
Figura 2-7. DataGrip 22
Figura 2-8. Servidor PgAdmin 23
Figura 2-9. Postman 24
Figura 2-10. Control de versiones (GitHub) 25
Figura 2-11. Logo MagicDraw 25
Figura 3-1. Diagrama de casos de uso del sistema 40
Figura 3-2. Diagrama modelo E-R del sistema 49
Figura 3-3. Diagrama del modelo relacional del sistema 50
Figura 3-4. Diagrama arquitectura cliente/servidor en 3 capas 51
Figura 3-5. Diagrama de paquetes Servidor REST 52
Figura 3-6: Diagrama de paquetes proyecto Android 53
Figura 3-7. Diagrama de secuencia: Interacción entre componentes Spring (obtención listado fármacos) 54
Figura 3-8. Diagrama de actividad: Inicio de sesión en el sistema 55
Figura 3-9. Diagrama de actividad: Mostrar catálogo de fármacos 56
Figura 3-10. Diagrama de secuencia: Interacción entre componentes Aplicación Android (cálculo dosis) 57
Figura 3-11. Fichero docker-compose.yml 58
Figura 3-12. Variables de entorno docker 58
Figura 3-13. Contenedores Docker: postgreSQL y pgAdmin 58
Figura 3-14. Servidores PgAdmin y PostgreSQL 59
Figura 3-15. Spring Initializr 60
Figura 3-16. Configuración específica aplicación Spring 60
Figura 3-17. Test de conexión con la base de datos 61
Figura 3-18. Mapeo Entitidad - Tabla en base de datos 61
Figura 3-19. Entidades de la Aplicación Spring 61
Figura 3-20. Repositorios de la Aplicación Spring 62
Figura 3-21. Servicios de la Aplicación Spring 63
Figura 3-22. Controladores de la Aplicación Spring 64
Figura 3-23. Objetos de entrada al servicio REST (DTO) 64

Figura 3-24. Objetos de salida del servicio REST (Vistas) 65
Figura 3-25. Clases que permiten la seguridad del servicio 65
Figura 3-26. Clases implicadas en inicio de sesión y registro de nuevo usuario 66
Figura 3-27. Manejador de excepciones global 67
Figura 3-28. Ejemplo petición REST desde Postman 67
Figura 3-29. Colección Postman (Cliente REST) 68
Figura 3-30. Flujo de pantallas de Inicio de sesión y registro de usuarios 70
Figura 3-31. Flujo de pantallas catálogo fármacos 72
Figura 3-32. Flujo pantallas cálculo de mezclas 74
Figura 3-33. Pantalla de dosímetro 75

Notación

RINF	Requisito de información
RF	Requisito funcional
CU	Caso de uso
RNF	Requisito no funcional
JSON	JavaScript Object Notation
DTO	Data Transfer Object
POJO	Plain Old Java Object
API	Interfaz de Programación de Aplicaciones
REST	Representational State Transfer
CRUD	Create, Request, Update, Delete
SDK	Kit de Desarrollo Software
JPA	Java Persistence API
SGDB	Sistema Gestor de Base de Datos
DTO	Data Transfer Object
UML	Unified Modeling Languaje
URI	Uniform Resource Indentifier

1 INTRODUCCIÓN

El único modo de hacer un gran trabajo es amar lo que haces.

- Steve Jobs -

En el primer capítulo se expondrán la motivación y los principales objetivos del presente trabajo de fin de grado cuyo resultado final será la elaboración de la aplicación *CPMedical (Chronic Pain Medical*) destinada a ser usada por los profesionales sanitarios de las unidades de dolor crónico. Además, se definirá la metodología y el plan de trabajo seguido.

1.1 Motivación y objetivos

La innovación tecnológica de las últimas décadas está suponiendo una revolución en la mayoría de los sectores por las innumerables ventajas que puede llegar a ofrecer, entre las que destacan la optimización del tiempo, la automatización de procesos y la disminución de errores humanos.

Este proyecto estará altamente relacionado con la aplicación de la tecnología en el mundo de la salud. Existen infinidad de estudios en busca de nuevas soluciones, o mejoras de las ya existentes, con el objetivo de poder llegar a garantizar una sanidad de calidad y libre de errores.

Algunos de los múltiples beneficios de la existencia de una sanidad digitalizada y automatizada se enumeran a continuación:

- El diagnóstico precoz de enfermedades mediante el uso de la inteligencia artificial y el análisis de datos.
- La optimización de la toma de decisiones de los profesionales de la salud mediante diferentes tipos de aplicaciones y gracias al Big Data.
- La posibilidad de realizar diagnósticos e incluso cirugías a distancia mediante la telemedicina.
- La monitorización de la salud mediante herramientas digitales como relojes inteligentes.
- La mejor formación de los sanitarios gracias al entrenamiento médico con realidad virtual.

La principal motivación del proyecto gira en torno a la búsqueda de mecanismos para el uso seguro y eficiente de los fármacos. Los errores de medicación (cualquier error en el uso de medicamentos), son todavía muy frecuentes y la mayoría de ellos se deben a errores humanos durante alguno de las fases por las que este pasa: prescripción, transcripción, dispensación y administración. Sin embargo, con aplicaciones tecnológicas estos errores pueden disminuirse en gran medida. En concreto, existen varias formas de disminuir los errores según la fase, se podrían destacar las siguientes:

- Errores de prescripción y transcripción: aplicaciones de prescripción electrónica y/o asistida.
- Errores de dispensación: sistemas de almacenamiento y dispensación automatizados.
- Errores de administración: sistemas de administración con código de barras o inteligentes

Se considera de gran importancia la búsqueda de soluciones para prevenir los errores de medicación ya que estos tienen un gran impacto que afecta directamente a la salud del paciente que recibe un tratamiento incorrecto. Por ejemplo, en el sector del dolor crónico se usan medicamentos muy fuertes cuyas dosis tienen que controlarse rigurosamente dado que si se supera un valor concreto pueden llegar a ser tóxicas para el receptor.

El principal objetivo que se pretende conseguir con la elaboración de este proyecto de fin de grado es diseñar y

desarrollar una herramienta software para la prevención de los errores de medicación en la fase de prescripción de medicamentos para el dolor crónico.

Este objetivo principal depende de otros objetivos secundarios como son:

- Conocer las herramientas tecnológicas existentes en el sector sanitario.
- Analizar, diseñar e implementar una aplicación a partir de los requisitos de profesionales de la salud reales.

Otros objetivos complementarios que se espera alcanzar son:

- Afianzar y aumentar los conocimientos relacionados el desarrollo de aplicaciones software.
- Afianzar y aumentar los conocimientos relacionados con los servicios REST.

1.2 Metodología

Las metodologías de desarrollo de software son un conjunto de técnicas que pretenden organizar las diferentes etapas involucradas en el ciclo de vida de un producto final de la mejor forma posible. Una buena organización disminuirá la probabilidad de errores, retrasos. Además, reducirá el nivel de dificultad, optimizará el tiempo y mejorará el resultado final.

Para el desarrollo del presente trabajo de fin de grado se ha empleado principalmente una metodología en cascada, es decir, las etapas se han ido ejecutando de forma secuencial de modo que hasta que no se termine una no se pasa a la siguiente. Cabe destacar que no se ha seguido esta metodología de manera estricta ya que en algunas de las etapas han surgido problemas que han implicado retroceder a las anteriores etapas y replantear varios aspectos. Por ejemplo, en la etapa de implementación de uno de los endpoints del servicio REST, se observó que no se obtenía el resultado esperado ya que había un problema en las relaciones del diseño del modelo de datos.

Por otra parte, en la etapa de implementación se ha seguido una metodología incremental, es decir, se fueron implementando las distintas funcionalidades de manera progresiva y se iban marcando objetivos funcionales con el fin de ver resultados de forma más temprana. Por ejemplo, primero se implementó todo lo relacionado con el módulo de inicio de sesión.

Las distintas etapas que se han ejecutado se describirán en el siguiente apartado.

1.3 Plan de trabajo

Antes de comenzar un proyecto es importante que se defina el plan de trabajo que se va a seguir con el fin de organizar y optimizar el tiempo, así como para aumentar la eficiencia. Para la ejecución del presente proyecto, se han definido una serie de etapas que son por las que la mayoría de los equipos de desarrollo software pasan para llegar a un producto final. Estas etapas definen el ciclo de vida del software y son las siguientes:

- **Documentación y planificación.** Se trata de la fase en la que se estudia el contexto y los fundamentos teóricos relacionados con el proyecto. Se han investigado los distintos conceptos y técnicas sanitarias relacionadas con el área del dolor crónico y se ha realizado un estudio sobre las posibles tecnologías a utilizar y el mercado actual. Con estos conocimientos previos sería más efectiva la entrevista para la toma de requisitos con el usuario final.
- **Toma de requisitos.** Se trata de la fase en la que se conocen cuáles son los objetivos y necesidades que el usuario final quiere conseguir con el producto software. En este caso, los requisitos se han definido a través de entrevistas con el coordinador de la Unidad del Dolor en el Hospital Universitario Virgen del Rocío.
- **Análisis.** Se trata de la fase en la que se han formalizado los requisitos extraídos de la entrevista con el usuario final y se han definido el alcance, los objetivos y los diferentes casos de uso que se esperan del sistema.

- **Diseño.** Se trata de la fase en la que se han estudiado las posibles opciones de implementación y estructura para el correcto funcionamiento del producto software. También se ha definido cuál será la arquitectura final del sistema y se han diseñado un prototipo de la aplicación y una serie de diagramas con el objetivo de facilitar la siguiente etapa.
- **Implementación.** Se trata de la fase en la que se han elegido las herramientas, el entorno de desarrollo y el lenguaje de programación adecuado para el tipo de software a construir. A continuación, se han desarrollado todas las funcionalidades contempladas en el alcance del proyecto siguiendo el catálogo de buenas prácticas.
- **Pruebas.** Durante esta fase se ha buscado detectar y corregir los posibles fallos cometidos en las etapas anteriores, así como testear que el funcionamiento del sistema es el esperado.

A continuación, se inserta una tabla con el desglose de tareas implicadas en cada una de las etapas anteriormente descritas, así como la planificación temporal de las mismas y la dedicación final:

Tareas	Horas estimadas	Horas reales
Documentación y planificación	10h	15h
Preparación de entrevista para toma de requisitos	3h	3h
Entrevista para toma de requisitos	3h	3h
Comprensión y definición de algoritmos de cálculo	8h	15h
Definición de objetivos y alcance	2h	3h
Formalización y especificación de requisitos del sistema	10h	15h
Definición de casos de uso	15h	20h
Diseño de prototipo	10h	16h
Análisis y documentación de tecnologías y herramientas	20h	20h
Diseño del sistema	10h	16h
Diseño del modelo de datos	6h	8h
Definición de la arquitectura del sistema	5h	5h
Creación de la base de datos	5h	6h
Implementación del servicio API REST	25h	28h
Implementación de la aplicación Android	100h	110h

Pruebas	20h	20h
Refactorizaciones y mejoras	10h	10h
Correcciones	8h	7h
Desarrollo de la memoria	70h	80h
Total de horas	**350h**	**390h**

Tabla 1. Plan de trabajo

2 ESTADO DEL ARTE

El placer más noble es el júbilo de comprender.

- Leonardo Da Vinci -

En el segundo capítulo del documento se va a proceder a contextualizar el ámbito de la aplicación desde un punto de vista tanto sanitario como tecnológico. Es por ello por lo que se comenzará detallando ciertos aspectos relacionados con el dolor crónico y, a continuación, se describirán las tecnologías y herramientas usadas para el análisis, el diseño, la implementación de las aplicaciones (Android y web) y las pruebas.

2.1 Marco teórico

Aunque se hablará más adelante de los requisitos concretos de la aplicación que se va a diseñar e implementar, será importante entender bien los cálculos que los profesionales del dolor deben realizar en su día a día. De este modo, se comprenderá la situación de partida y el nacimiento del problema. En concreto, destacan dos tipos de cálculo: de mezclas y de dosis. Además, se comentará qué es una bomba de infusión de medicamentos ya que los cálculos anteriormente mencionados están altamente relacionados con estas.

2.1.1 Bombas de infusion de medicamentos

Las bombas de infusión de fármacos (sistemas de administración intratecal de fármacos) suministran medicamentos para el dolor en la zona llena de líquido que rodea a la médula espinal (llamada espacio intratecal) [1]. Dado que los medicamentos para el dolor van directamente al área que rodea la columna vertebral, una bomba de infusión de fármacos puede ofrecer un grado considerable de control del dolor con una dosis menor que la requerida con medicamentos orales. Este suministro dirigido está diseñado para reducir el dolor y minimizar los efectos secundarios, mejorando la calidad de vida de las personas.

Durante la cirugía, el médico llena la bomba con medicamentos para el dolor con una jeringa. La bomba envía el medicamento a través del catéter al área de la columna vertebral donde se encuentran los receptores del dolor.

El sistema consiste en una bomba y un catéter que se implantan mediante cirugía debajo de la piel. La bomba es un dispositivo redondeado que almacena y suministra medicamentos para el dolor. Normalmente, ésta se implanta en el abdomen. El catéter (un tubo delgado y flexible) se introduce en la columna vertebral y se conecta a la bomba.

El funcionamiento de este tipo de bombas puede explicarse con la analogía siguiente: la médula espinal es como una carretera por la cual viajan señales del dolor que se dirigen al cerebro. Cuando la bomba envía el medicamento para el dolor directamente a la columna vertebral, interrumpe las señales del dolor antes de que lleguen al cerebro.

Todos los tratamientos y resultados son específicos de cada paciente y forman parte de la consulta con un profesional médico. El sanitario determinará qué fármacos se van a suministrar y cuál será la dosis A partir de ahí, deberá realizar una serie de cálculos para determinar qué volumen de cada medicamento recetado deberá insertarse en la bomba. El resultado final es lo que se conoce como mezcla o carga de la bomba.

En la actualidad, los fármacos relacionados con el dolor pueden encontrarse en farmacia a través de distintas presentaciones comerciales que varían en función de la concentración que contienen del fármaco en cuestión. Es por ello por lo que el sanitario también deberá especificar qué presentación se va a usar ya que esta se debe tener en cuenta en los cálculos de la mezcla para garantizar la dosis deseada.

Por otra parte, dado que los fármacos relacionados con el tratamiento del dolor son muy fuertes y puede darse el caso de que no sean aceptados por el organismo del paciente, el procedimiento habitual es recetar los fármacos nuevos en una menor dosis para ir viendo la evolución del paciente. Si el fármaco es aceptado, se tendrá que modificar la mezcla que se le añadió en la bomba. En este tipo de situaciones, también se realizan cálculos para evitar tener que vaciar las bombas de infusión.

En el *Anexo I: Procesos de cálculo* se puede encontrar más información acerca del proceso de cálculo de mezclas y de nuevas dosis.

2.2 Herramientas software

2.2.1 Tecnologías

2.2.1.1 Android

Android es un sistema operativo móvil basado en el kernel Linux especialmente diseñado para dispositivos móviles [2]. Se caracteriza por ser de software libre, lo que permite que los fabricantes puedan usarlo sin necesidad de pagar royalties. Por otra parte, al correr sobre Linux, es fácilmente portable y adaptable a casi cualquier hardware.

Este sistema operativo será el que soporte la aplicación móvil desarrollada. Se ha elegido por ser uno de los sistemas operativos más usados en la actualidad además de Open Source.

Figura 2-1. Logo Android

2.2.1.2 Java

Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado, y cada día se crean más [3]. Java es rápido, seguro y fiable. Desde ordenadores portátiles hasta centros de datos, desde consolas para juegos hasta computadoras avanzadas, desde teléfonos móviles hasta Internet, Java está en todas partes. Si es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados.

Este es el lenguaje de programación usado para desarrollar la aplicación móvil y el servicio REST.

Figura 2-2. Logo Java

2.2.1.3 Servicios web

Un servicio web facilita un servicio a través de Internet: se trata de una interfaz mediante la que dos máquinas

(o aplicaciones) se comunican entre sí. Esta tecnología se caracteriza por estos dos rasgos:

- **Multiplataforma**: cliente y servidor no tienen por qué contar con la misma configuración para comunicarse. El servicio web se encarga de hacerlo posible.
- **Distribuida**: por lo general, un servicio web no está disponible para un único cliente, sino que son diferentes los que acceden a él a través de Internet.

Cuando se utiliza un web service, un cliente manda una solicitud a un servidor, desencadenando una acción por parte de este. A continuación, el servidor devuelve una respuesta al cliente [4].

Para ofrecer la interfaz, los servicios web cuentan con un Uniform Resource Identifier (URI) unívoco, esto es, la dirección del servicio web o endpoint. Es similar al Uniform Resource Locator (URL) que permite acceder a páginas web.

En cuanto a la comunicación, funciona exclusivamente mediante diferentes protocolos y arquitecturas. Entre ellos, son muy populares el protocolo de red SOAP en combinación con el estándar de Internet HTTP o los servicios web basados en una arquitectura REST.

En SOAP, que emplea el intercambio de datos XML, las operaciones son definidas como puertos WSDL y la dirección es única, a lo que hay que añadir que numerosas instancias del proceso comparten la misma operación [5]. En REST, sin embargo, las operaciones se definen en los propios mensajes y hay una dirección única para cada instancia del proceso. Los componentes no se acoplan del mismo modo en ambos casos; mientras que en SOAP están fuertemente acoplados, en una API REST esa unión es débil.

En este proyecto se ha implementado un servicio de tipo API RESTful ya que, frente a SOAP, es más simple, escalable y flexible; consume menos recursos y el cliente no requiere información de enrutamiento, solo necesita la URI inicial. Esta API REST servirá para que la aplicación móvil pueda interaccionar con el sistema de datos o pueda obtener recursos a través de ella.

2.2.1.4 Spring

2.2.1.4.1 Spring Framework

Spring Framework es un framework Open Source que facilita la creación de todo tipo de aplicaciones en Java, Kotlin y Groovy [6]. Está dividido en diversos módulos que ofrecen diferentes funcionalidades:

- **Core container:** proporciona inyección de dependencias e inversión de control.
- **Web:** permite crear controladores Web tanto de vistas MVC como de aplicaciones REST.
- **Acceso a datos:** proporciona abstracciones sobre JDBC, ORMs, sistemas OCM (Object XML Mappers), JSM y transacciones.
- **Programación orientada a objetos (AOP):** ofrece el soporte de aspectos.
- **Instrumentación:** proporciona soporte para la documentación de clases.
- **Test:** proporciona soporte para Junit TestNG y todo lo necesario para probar los mecanismos de Spring.

Estos módulos son opcionales, por lo que pueden ser utilizados sin tener que llenar el classpath con clases que no se van a emplear.

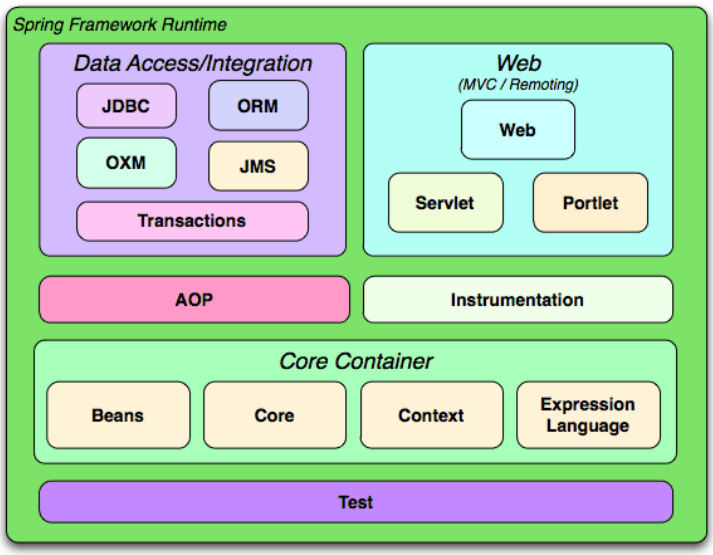


Tabla 2. Módulos de Spring Framework (Recurso en línea: [6])

Una de las ventajas de Spring es que aumenta la productividad y reduce la complejidad ofreciendo abstracciones sobre implementaciones de tecnologías concretas. Además, es Open Source y tiene una gran comunidad.

Spring Framework es un es un conjunto de proyectos Open Source que hacen que programar en lenguajes como Java sea más rápido, fácil y seguro ya que proporciona soluciones a diversos problemas técnicos.

2.2.1.4.2 Spring Boot

En Spring Framework, la configuración inicial de un proyecto y el despliegue de este resultaban complejos [7]. Así, nació Spring Boot con el fin de permitir crear aplicaciones autocontenidas delegando en Spring Boot las labores de dependencias y el despliegue del servicio de aplicaciones.

Para ello, Spring Boot utiliza internamente un servidor de aplicaciones embebido. Lo más normal es utilizar Tomcat. Aunque también se puede hacer con Jetty o Undertow. Además, dispone de un completo gestor de dependencias. Y facilita la creación de aplicaciones independientes [8].

Los principales mecanismos que ofrece son:

- **Contenedor de aplicaciones integrado:** compilar las aplicaciones web como un archivo .jar único. Que después se puede ejecutar como una aplicación Java normalmente. Así, se pueden distribuir aplicaciones de una forma mucho más sencilla porque se configura el servidor junto con la aplicación, siendo una opción muy útil en arquitecturas de microservicios.
- **Starters:** son una serie de dependencias preestablecidas que se añaden al proyecto dependiendo de las necesidades del desarrollo. Cada starter proporciona las dependencias y configuraciones de una herramienta. Tanto las que son de Spring. Como las que son de terceros. Por ejemplo, si desea utilizar Spring Data JPA para el acceso a la base de datos, basta con incluir spring-boot-starter-data-jpa. La ventaja de usarlos permite aumentar la productividad, disminuir el tiempo de configuración, reducir la cantidad de dependencias y el uso de configuraciones de dependencia probadas y admitidas
    - Aumente la productividad al disminuir el tiempo de configuración para los desarrolladores.
    - Administrar el POM es más fácil ya que se reduce la cantidad de dependencias que se agregarán.
    - Configuraciones de dependencia probadas, listas para producción y admitidas.

En este proyecto se ha usado Spring para el desarrollo de la aplicación que ofrece el servicio REST. Para ello, se han usado los starters que permiten el uso de Spring Data JPA, Spring MVC, Spring Validation y Spring Security.

2.2.1.4.3 Spring Security

Spring Security trata de agrupar todas las funcionalidades de seguridad sobre proyectos Spring [9].

El control de acceso permite limitar las opciones que pueden ejecutar un determinado conjunto de usuarios o roles sobre la aplicación. En esta dirección, Spring Security controla las invocaciones a la lógica de negocios o limita el acceso de peticiones HTTP a determinadas URLs.

Para ello, se debe configurar cómo debe comportarse la capa de seguridad pudiendo realizar parametrizaciones y ajustes. Además, proporciona una serie de interfaces Java que, si se implementan, permiten cambiar el comportamiento de una determinada funcionalidad, como el ir a buscar los usuarios a una fuente de datos determinada.

Es recomendable usar Spring Security, ya que es código muy maduro ampliamente testado y además se mantiene actualizado por la comunidad cuando se publica un aviso de seguridad o ante cualquier publicación un nuevo estándar.

En este proyecto se ha usado Spring Security para autenticar y autorizar a los usuarios que desean obtener algún recurso del servicio REST.

2.2.1.5 Patrones

2.2.1.5.1 DAO

El patrón Data Access Object (DAO) pretende principalmente independizar la aplicación de la forma de acceder a la base de datos, o cualquier otro tipo de repositorio de datos [10]. Para ello se centraliza el código relativo al acceso al repositorio de datos en las clases llamadas DAO. Fuera de las clases DAO no debe haber ningún tipo de código que acceda al repositorio de datos.

En este proyecto, los repositorios de Spring siguen este patrón.

2.2.1.5.2 MVC

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos [11].

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

- El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La **Vista**, o interfaz de usuario, que compone la información que se envía al usuario final y los mecanismos interacción con éste.
- El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

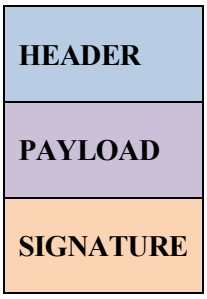
En este proyecto, la arquitectura de Spring escogida sigue este patrón.

2.2.1.6 Token JWT

Un token JWT (JSON Web Token) es un estándar definido en la RFC 7519 [12]. Se trata de un mecanismo que permite intercambiar entre dos partes y de forma segura la identidad y los privilegios de un usuario.

Este tipo de token tiene un formato un JSON codificado en Base64 y puede estructurarse en las siguientes partes:

- **Header:** parte en la que se indica el algoritmo de cifrado con el que se firmará el token y el tipo de token en cuestión.
- **Payload:** parte en la que se indican los datos del usuario, los privilegios de este y cualquier otra información necesaria.
- **Signature:** parte que incluye la firma que permitirá verificar si el token es válido, es decir, que el remitente es quien dice ser y que el mensaje no se ha modificado.



En cuanto al funcionamiento de JWT, actúa de modo que cuando el usuario quiera iniciar sesión en el sistema, deberá realizar una petición HTTP POST en la que incluya sus credenciales. Si estas credenciales son correctas, el sistema de autenticación creará un token JWT firmado con la clave del usuario y se lo enviará a este. De este modo, cuando el cliente quiera acceder a algún recurso protegido deberá agregar su token en la petición y, si dispone de los privilegios adecuados y la firma son los adecuados, el servidor autorizará la petición.

2.2.1.7 JPA

A la hora de desarrollar, lenguajes como Java manejan su información utilizando objetos (Programación Orientada a Objetos), pero las bases de datos utilizan otros paradigmas: las relaciones y las claves que las relacionan [13]. Esta diferencia entre la forma de programar y la forma de almacenar da lugar a lo que se llama "desfase de impedancia" y complica la persistencia de los objetos.

JPA es una especificación que indica cómo se debe realizar la persistencia. Dado que es una especificación, JPA no proporciona clase alguna para poder trabajar con la información. En la práctica significa que lo que se va a utilizar es una biblioteca de persistencia que implemente JPA, no JPA directamente.

Existen diversas implementaciones disponibles, como DataNucleus, ObjectDB, o Apache OpenJPA, pero las dos más utilizadas son EclipseLink y sobre todo Hibernate.

Por otra parte, una de las grandes ventajas de JPA es que permite que el acceso a datos sea independiente al sistema gestor de base de datos usado (siempre que este sea compatible con JPA).

En este proyecto se ha usado JPA y el motor de Hibernate para el acceso a los datos desde el servicio REST. Spring JPA ha facilitado las labores de configuración.

2.2.1.8 Docker

Docker es una herramienta que permite empaquetar una aplicación y sus dependencias en un contenedor muy ligero [14]. Es como si tomaras una aplicación completa con todo lo que necesita para funcionar para poder transportarla sin problema a cualquier otro servidor con Docker instalado, ya sea para seguir desarrollándola o para hacer desplegarla. De este modo, se caracterizan por la portabilidad que ofrecen, su ligereza y su autosuficiencia.

En este proyecto se ha usado Docker para el despliegue del sistema gestor de base de datos de PostgreSQL y el administrador de PgAdmin con el objetivo que estos sistemas sean fácilmente portables.

Figura 2-3. Logo Docker

2.2.1.9 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacionales (RDBMS) libre y de Open Source que hace énfasis en la extensibilidad y el cumplimiento de SQL [15]. Tiene las siguientes características

- Posibilidad de realizar consultas complejas.
- Clave foránea para conectar datos de dos tablas.
- Disparadores (*trigger*) que se inician de forma automática a partir de una entrada y la comprueban, la confirman o la eliminan o emplean datos de referencia.
- Vistas actualizables.

- Concepto muy amplio de transacción.
- Control de concurrencia mediante versiones múltiples (*Multiversion Concurrency Control*, MVCC) para que el acceso simultáneo a la BD se ejecute de forma eficiente.

En este proyecto se ha usado sistema gestor de base de datos y se ha desplegado a través de Docker.

Figura 2-4. Logo PostgreSQL

2.2.2 Entorno de trabajo

2.2.2.1 IntelliJ IDEA

IntelliJ IDEA es un IDE inteligente y sensible al contexto para trabajar con Java y otros lenguajes JVM como Kotlin, Scala y Groovy en todo tipo de aplicaciones. Además, IntelliJ IDEA ofrece potentes herramientas integradas y compatibilidad avanzada con marcos de trabajo populares como Spring, Spring Boot [16].

Este entorno se ha usado para el desarrollo del servicio REST mediante Spring y se ha empleado su versión de estudiante.

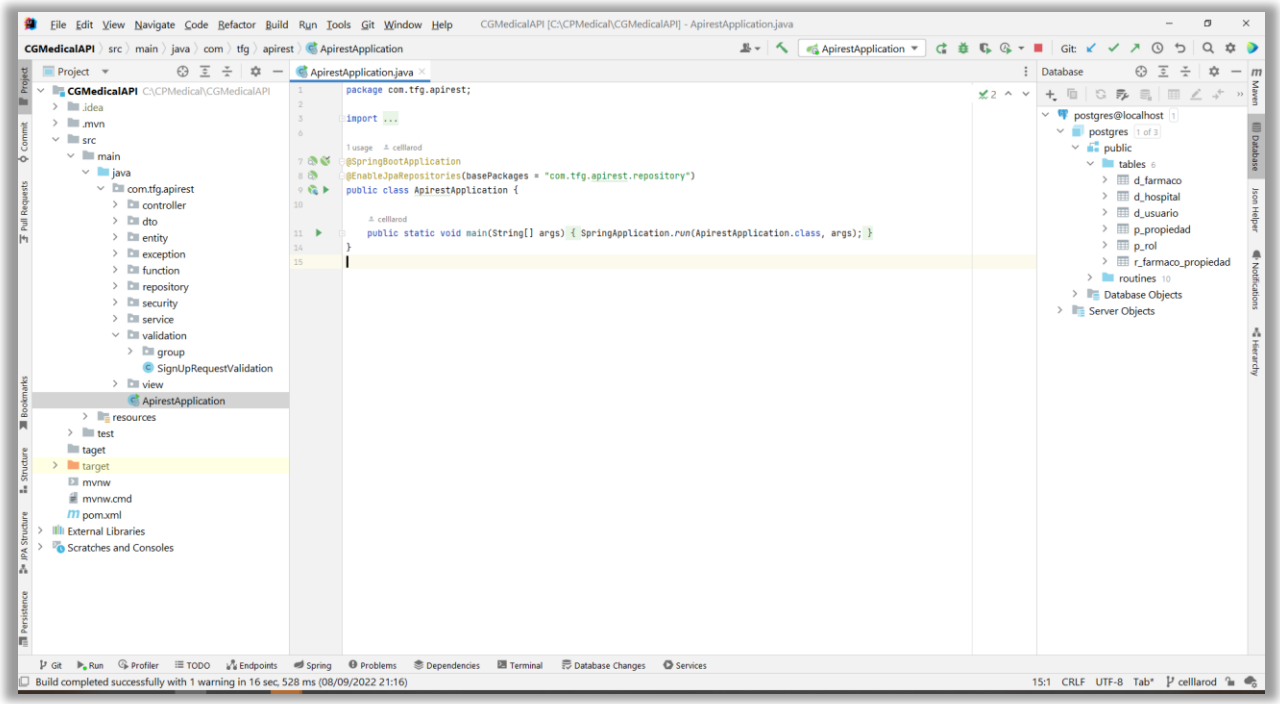


Figura 2-5. IntelliJ IDEA

2.2.2.2 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y está basado en IntelliJ IDEA [17]. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan la productividad cuando se desarrollan aplicaciones para Android.

Este entorno se ha usado para el desarrollo de la aplicación móvil Android.

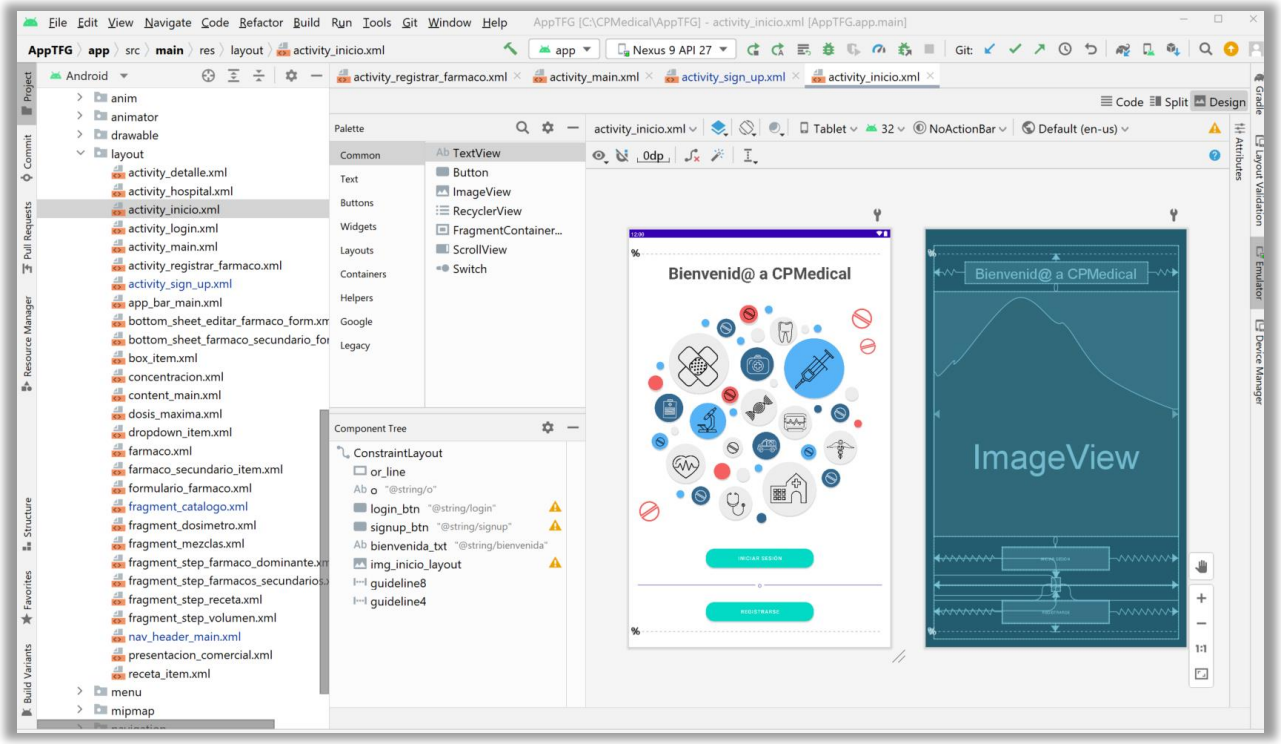


Figura 2-6. Android Studio

2.2.2.3 DataGrip

DataGrip es un entorno de gestión de bases de datos para desarrolladores. Está diseñado para consultar, crear y administrar bases de datos ofreciendo numerosas herramientas que agilizan estas tareas [18]. Las bases de datos pueden funcionar localmente, en un servidor o en la nube. Admite MySQL, PostgreSQL, Microsoft SQL Server, Oracle y más.

Este entorno se ha empleado para gestionar la base de datos PostgreSQL a través de su versión de estudiante.

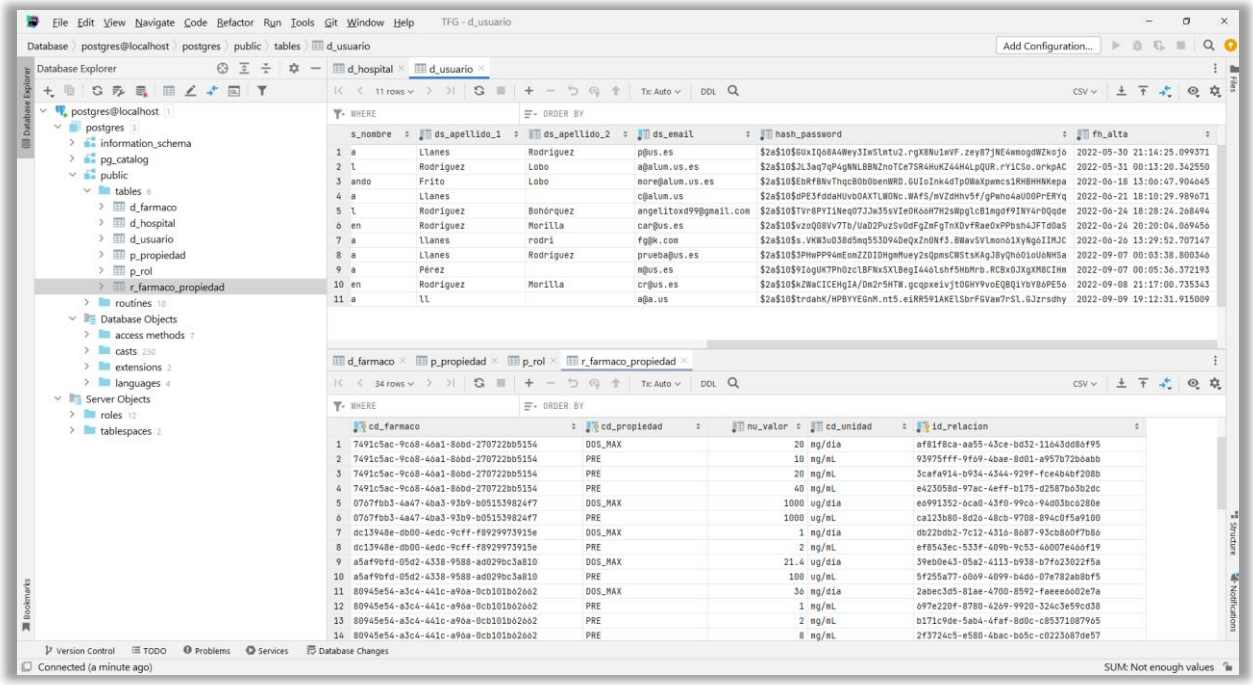


Figura 2-7. DataGrip

2.2.2.4 PgAdmin

PgAdmin es un gestor especializado de PostgreSQL . Es fácil de utilizar, intuitivo y Open Source. Una de las características más destacables que ofrece frente a otros gestores es la opción de monitorización del servidor.

En este proyecto se ha usado para gestionar la base de datos PostgreSQL y se ha desplegado a través de Docker.

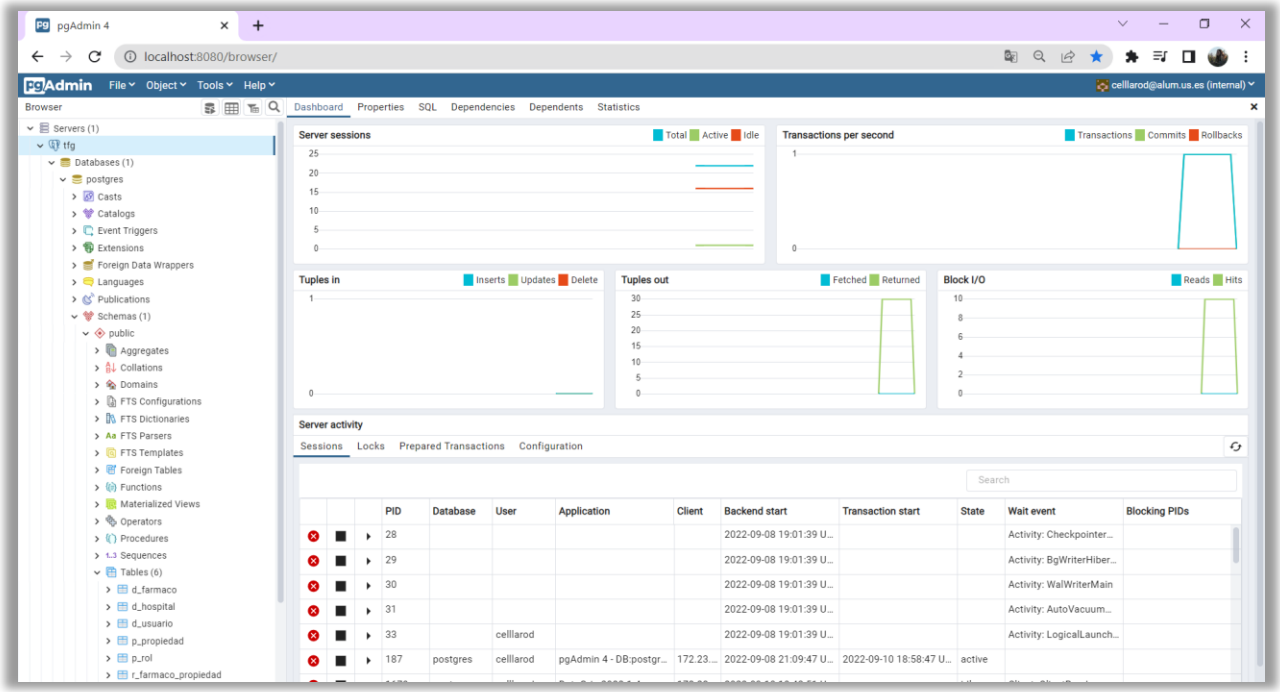


Figura 2-8. Servidor PgAdmin

2.2.2.5 Postman

Postman es una herramienta que actúa como cliente HTTP y está destinada a realizar pruebas de una API a través de una interfaz gráfica de usuario. Ofrece numerosas herramientas y posibilidades para realizar peticiones y analizar sus respuestas, así como para generar documentación de la API .

En este proyecto se ha usado Postman para probar cada uno de los endpoints que forman la API REST ofrecida por la aplicación de Spring.

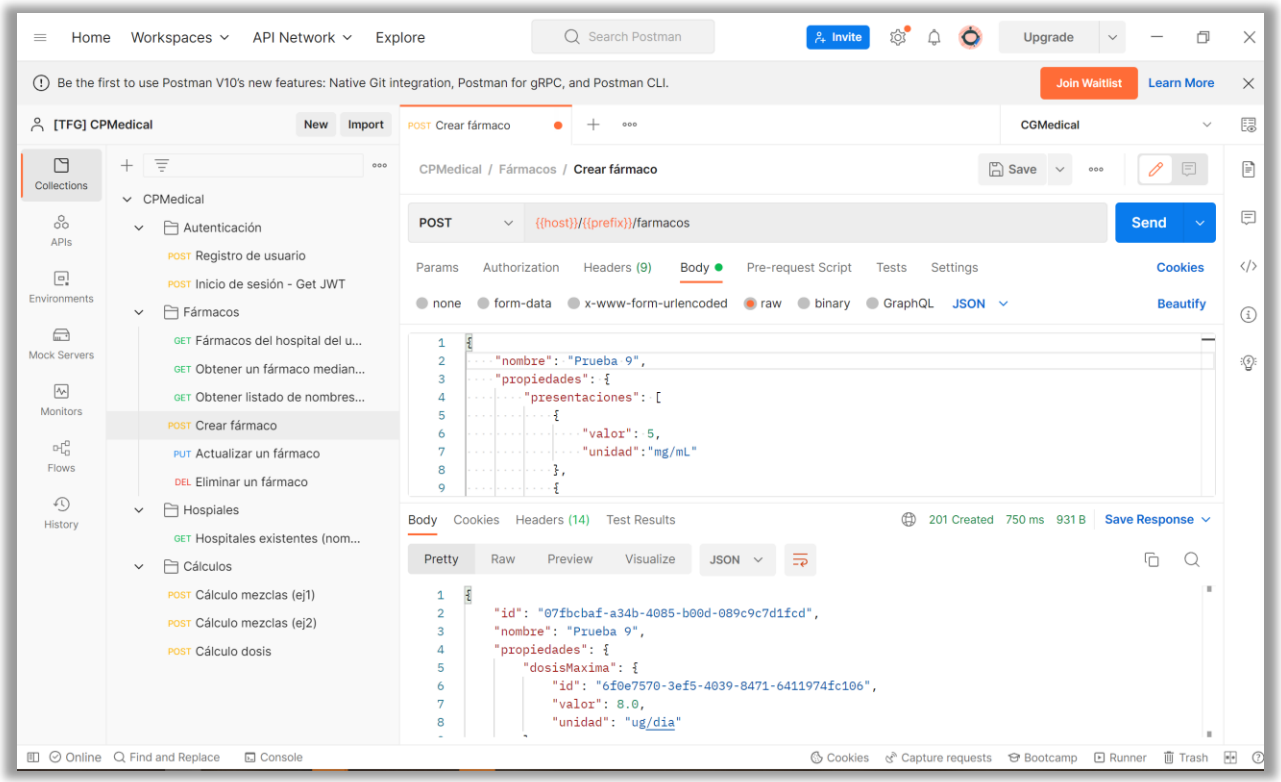


Figura 2-9. Postman

2.2.2.6 Control de versiones: Git y GitHub

El control de versiones, también conocido como "control de código fuente", es la práctica de rastrear y gestionar los cambios en el código de software [19]. Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo.

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error.

En este proyecto se ha usado para tener un control de versiones de todo el código que se ha ido desarrollando, tanto del servicio REST como de la aplicación Android. Para la administración en la nube del repositorio de Git creado, se ha usado GitHub. Además, también se ha empleado SourceTree, que es una herramienta visual para realizar operaciones sobre el repositorio.

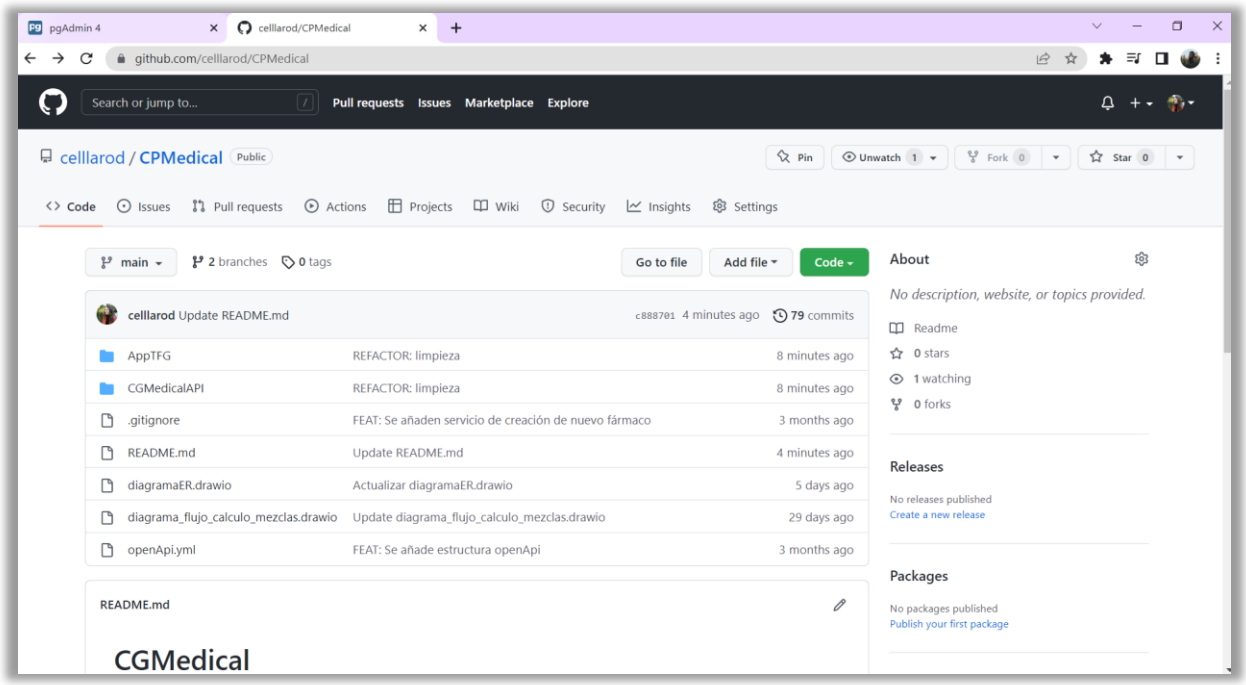


Figura 2-10. Control de versiones (GitHub)

2.2.2.7 MagicDraw

MagicDraw, herramienta CASE desarrollada por No Magic. Es compatible con el estándar UML 2.3, desarrollo de código para diversos lenguajes de programación (Java, C++ y C#, entre otros) así como para modelar datos [20].

En este proyecto se ha empleado para crear algunos de los diagramas que se pueden encontrar en el apartado de diseño.

Figura 2-11. Logo MagicDraw

3 RESULTADOS

El ignorante afirma, el sabio duda y reflexiona

- Aristóteles-

En este capítulo se describirá en profundidad todo el proceso que se ha llevado a cabo para la realización del presente trabajo y la obtención del producto final: la aplicación móvil "CPMedical". En primer lugar, se describirá la fase de análisis y definición de requisitos. A continuación, se detallarán las etapas de diseño, implementación y despliegue del sistema final.

3.1 Análisis y definición de requisitos

3.1.1 Introducción

El análisis y la especificación de requisitos forman parte de la fase inicial de todo proyecto software, además de ser una de las más relevantes. Implica realizar un estudio profundo del dominio del problema con el fin de obtener las necesidades del usuario final, marcar unos objetivos y poder proponer un sistema completo y usable sin entrar en aspectos de diseño.

3.1.2 Proceso de identificación de necesidades de negocio

Como se ha mencionado en capítulos anteriores, el fin del sistema será ofrecer una herramienta para digitalizar algunas de las tareas diarias que deben realizar los profesionales del sector de la Unidad del Dolor de un hospital. De este modo, para identificar las necesidades reales de estos profesionales y de entender a la perfección cada una de las tareas a digitalizar, los requisitos del sistema se han formalizado a través de entrevistas con el coordinador de la Unidad del Dolor del Hospital Universitario Virgen del Rocío de Sevilla.

3.1.3 Dominio del problema

El problema principal está altamente relacionado con las bombas de infusión de medicamentos. La mayoría de los tratamientos del dolor crónico dependen de bombas de infusión que son cargadas con unas concentraciones específicas para cada paciente. Estas concentraciones se obtienen a través de procesos matemáticos que los profesionales del dolor deben manejar diariamente.

Tras la entrevista, se extrajeron dos procesos de cálculo principales. Estos son los siguientes:

1. **Cálculo de mezclas.** Tras realizar el diagnóstico de un paciente, se determinan los fármacos que se le van a suministrar a través de una bomba de infusión de medicamentos. El personal clínico indica la dosis diaria de cada fármaco requerida por el paciente a través de una serie de criterios médicos. Tras esta decisión, se procede a realizar una serie de cálculos con el fin de especificar qué carga (cantidad o volumen) de cada uno de los fármacos se debe añadir a la bomba.
2. **Cálculo de dosis.** Los diagnósticos suelen implicar el suministro de uno o más fármacos al paciente. En la mayoría de los casos, de todos los fármacos recetados siempre existe uno dominante, es decir, uno cuya dosis diaria es superior y que, por tanto, tendrá mayor presencia en la bomba. El resto de los fármacos (secundarios) se suelen introducir en el paciente mediante una dosis baja para ver cómo son aceptados en su organismo.

   Tras la primera infusión, cuando el paciente vuelve a la consulta (al cabo de 15 días, por ejemplo), el

especialista analiza la reacción que se ha producido en el organismo del paciente ante cada uno de los fármacos suministrados. Si la reacción ha sido satisfactoria, se podrá subir su dosis diaria, en caso contrario, habrá que disminuirla. Al establecer una nueva dosis para un medicamento existen dos actuaciones posibles:

a. Por una parte, se puede vaciar completamente la bomba y realizar de nuevo todo el proceso de cálculo de mezclas teniendo en cuenta la nueva dosis. Esta opción no es rentable ya que implica desperdiciar fármacos de elevado coste.

b. Por otra parte, se puede realizar una simulación que calcule cuánto debe variar la dosis de un fármaco si se sube la dosis de otro. Esta opción es la que se suele usar por su rentabilidad.

Actualmente, todos estos procesos de cálculo se realizan de forma manual. Se suelen reunir varios componentes del equipo médico y cada uno de ellos valida los cálculos realizados para minimizar posibles errores ya que estos pueden afectar gravemente a la salud del paciente que recibiría un tratamiento incorrecto. Sin embargo, los errores humanos siempre pueden producirse.

3.1.4 Debilidades de la situación actual

El sistema actual presenta una serie debilidades que se describirán a continuación. Se procurará construir un sistema que elimine estas debilidades.

DEB-01	**Catálogo de medicamentos**
Versión	1. 0 (28/03/2022)
Descripción	El personal médico debe consultar la información de los fármacos a la hora de realizar los diferentes cálculos. Esto puede provocar que se escoja por error algún dato que no esté asociado al medicamento deseado.

Tabla 3. DEB-01: Catálogo de medicamentos

DEB-02	**Ausencia de control externo al seleccionar la dosis de un fármaco**
Versión	1. 0 (28/03/2022)
Descripción	El personal médico puede elegir por error valores de dosis con un rango superior al permitido o relaciones entes dosis de distintos fármacos no recomendadas.

Tabla 4. DEB-02: Ausencia de control externo al seleccionar dosis

DEB-03	**Cálculos manuales**
Versión	1. 0 (28/03/2022)
Descripción	Se realizan todos los cálculos de manera manual. Para evitar posibles errores, estas operaciones son repetidas por distintos profesionales, lo que supone un elevado coste temporal y, aunque disminuye la probabilidad de error, no la elimina.

Tabla 5. DEB-03: Cálculos manuales

DEB-04	**Ausencia de control externo al obtener resultados**
Versión	1. 0 (28/03/2022)
Descripción	El personal clínico puede obtener a partir de los cálculos resultados no admisibles que pueden ser pasados por alto erróneamente.

Tabla 6. DEB-04: Ausencia de control externo al obtener resultados

3.1.5 Necesidades de negocio

Una de las principales necesidades del sistema está ligada al manejo de errores. Resulta crucial anteponerse a ellos ya que pueden conllevar serios problemas. En concreto, la principal consecuencia de un fallo en estos aspectos afectará al paciente en cuestión que recibirá una medicación incorrecta, lo que podrá poner en peligro su salud.

Con relación a lo anterior, también resulta importante controlar los errores para evitar cargar las bombas con mezclas incorrectas ya que esto obligará a que se vacíe la misma, desechando así fármacos de elevado coste.

Por otra parte, la realización de cálculos de forma reiterada supone un gran impacto en el tiempo de los trabajadores. Así, surge la necesidad de hacer más eficiente el trabajo de los profesionales del dolor mediante el desarrollo de un software que permita digitalizar, automatizar y agilizar ciertas tareas que hasta ahora se realizaban manualmente.

3.1.5.1 Objetivos

A continuación, se enumerarán los objetivos de negocio que se esperan alcanzar cuando el sistema software esté en funcionamiento:

- **OBJ-01:** Eliminar y advertir de los posibles errores que se pueden producir a la hora de cargar bombas de infusión de medicamentos y de calcular las dosis para los distintos pacientes.
- **OBJ-02:** Gestión y digitalización del catálogo de fármacos existente en un hospital relacionados con el tratamiento del dolor crónico.
- **OBJ-03:** Optimizar el tiempo del personal clínico.
- **OBJ-04:** Digitalizar, automatizar y agilizar las tareas relacionadas con los cálculos de dosis y mezclas de una bomba de infusión.
- **OBJ-05:** Controlar el acceso al sistema y permitir que este pueda ser utilizado de forma independiente en diferentes hospitales.

3.1.5.2 Alcance

El sistema está enfocado principalmente a facilitar los cálculos que deben realizar el personal clínico a la hora de cargar las bombas de infusión de medicamentos y de decidir la combinación de fármacos que recibirán los pacientes con dolor crónico. De este modo, se pretenden prestar los siguientes servicios:

1. Cálculo de mezclas o carga de la bomba de infusión.
2. Cálculo de dosis.
3. Gestión de fármacos para el tratamiento del dolor.

3.1.6 Descripción de subsistemas

Los requisitos del sistema se han dividido en cuatro subsistemas principales que se describirán a continuación.

SD-01	**Cálculo de mezclas**
Versión	1. 0 (29/03/2022)

Descripción	Este subsistema agrupa los requisitos relacionados con los procesos de obtención de mezclas. A partir de una prescripción de fármacos realizada por un especialista a un determinado paciente, el subsistema será capaz de determinar qué volumen de cada medicamento deberá insertarse en la bomba de infusión para que dichos fármacos puedan ser suministrados al receptor correspondiente.
Importancia	Este subsistema forma parte del núcleo central del software.
Prioridad	Muy alta

Tabla 7. SD-01: Cálculo de mezclas

SD-02	**Dosímetro**
Versión	1. 0 (29/03/2022)
Descripción	Este subsistema agrupa los requisitos relacionados con los procesos de cálculo de la nueva dosis. En concreto, permitirá recalcular la nueva dosis de un fármaco cuando varía la dosis de otro de los fármacos implicados en un tratamiento específico.
Importancia	Este subsistema forma parte del núcleo central del software.
Prioridad	Muy alta

Tabla 8. SD-02: Dosímetro

SD-03	**Catálogo de fármacos**
Versión	1. 0 (30/03/2022)
Descripción	Este subsistema agrupa los requisitos relacionados con la gestión del catálogo de los fármacos para el tratamiento del dolor crónico relacionados a cada hospital.
Importancia	Este subsistema forma parte del núcleo central del software.
Prioridad	Alta

Tabla 9. SD-02: Catálogo de fármacos

SD-04	**Gestión de usuarios**
Versión	1. 0 (30/03/2022)
Descripción	Este subsistema agrupa los requisitos relacionados con la gestión de los usuarios del sistema.
Importancia	Este subsistema forma parte del núcleo central del software.
Prioridad	Alta

Tabla 10. SD-04: Gestión de usuarios

3.1.7 Catálogo de requisitos

3.1.7.1 Requisitos de información

Los requisitos de información son aquellos que revelan las necesidades y restricciones de almacenamiento del sistema.

RINF-001	***Información de un usuario***
Versión	1. 0 (04/04/2022)
Dependencias	RINF-004
Descripción	El sistema deberá almacenar la información correspondiente a sus usuarios. En concreto:
Datos específicos	• Nombre • Apellidos • Contraseña cifrada • Dirección de correo electrónico • Rol o permisos del usuario en el sistema (RINF-004) Nota: el segundo apellido será opcional.
Prioridad	Muy alta

Tabla 11. RINF-001: Información de un usuario

RINF-002	***Información de un hospital***
Versión	1. 0 (04/04/2022)
Dependencias	RINF-001, RINF-003
Descripción	El sistema deberá almacenar la información de los distintos hospitales que se creen en el sistema. En concreto:
Datos específicos	• Nombre • Identificador • Conjunto de fármacos para el tratamiento del dolor disponibles en un hospital concreto (catálogo de fármacos) (RINF-003) • Conjunto de usuarios que podrán acceder a su catálogo de fármacos. (RINF-004)
Prioridad	Muy alta

Tabla 12. RINF-002: Información de un hospital

RINF-003	***Información de un fármaco***
Versión	1. 0 (04/04/2022)
Descripción	El sistema deberá almacenar la información de los distintos fármacos que se hayan registrado y sus propiedades para casa uno de los hospitales en los que esté disponible.

	En concreto:
Datos específicos	• Nombre del fármaco • Identificador • Propiedades del fármaco • Presentaciones comerciales disponibles (medida en mg/mL o µg/mL) • Dosis diaria máxima admitida (medida en mg/día o µg/día)
Prioridad	Muy alta

Tabla 13. RINF-003: Información de un fármaco

RINF-004	***Información de roles***
Versión	1. 0 (04/04/2022)
Descripción	El sistema deberá almacenar la información de los distintos roles posibles de sus usuarios. En concreto:
Datos específicos	• Nombre del rol
Prioridad	Muy alta

Tabla 14. RINF-004: Información de roles

RINF-005	***Información cálculo de mezclas***
Versión	1. 0 (04/04/2022)
Descripción	El sistema deberá solicitar esta información cuando se quiera iniciar el proceso de cálculo de mezclas. En concreto:
Datos específicos	• Volumen de medicamentos que soportará la bomba de infusión de medicamentos (medido en mL). • Nombre del fármaco dominante (con mayor presencia en el tratamiento). • Dosis diaria que suministrar del fármaco dominante (medida en mg/día o µg/día). • Concentración para el fármaco dominante (medida en mg/mL o µg/mL). • Presentación comercial que usar para el fármaco dominante (medida en mg/mL o µg/mL). • Listado de fármacos secundarios (con menor presencia en el tratamiento). - Dosis diaria que suministrar del fármaco secundario (medida en mg/día o µg/día) - Presentación comercial para usar del fármaco secundario (medida en mg/mL o µg/mL). Nota: el listado de fármacos secundarios será opcional.
Prioridad	Muy alta

Tabla 15. RINF-005: Información cálculo de mezclas

RINF-006	***Información cálculo de dosis***
Versión	1. 0 (04/04/2022)
Descripción	El sistema deberá solicitar esta información cuando se quiera iniciar el proceso de cálculo de nuevas dosis. En concreto:
Datos específicos	• Nombre del fármaco dominante (con mayor presencia en el tratamiento) • Dosis diaria que se suministra actualmente del fármaco dominante (medida en mg/día o µg/día) • Nombre de un fármaco secundario • Dosis diaria que se suministra actualmente del fármaco secundario • Nueva dosis de uno de los dos fármacos (medida en mg/día o µg/día)
Prioridad	Muy alta

Tabla 16. RINF-006: Información cálculo de dosis

RINF-007	***Información registro de fármacos***
Versión	1. 0 (04/04/2022)
Dependencias	RINF-003
Descripción	El sistema deberá solicitar esta información cuando se quiera iniciar el proceso de asociar un nuevo fármaco al sistema. En concreto:
Datos específicos	Los datos serán los mismos a los indicados en el RINF-003.
Prioridad	Alta

Tabla 17. RINF-007: Información registro de fármacos

RINF-008	***Información edición de fármacos***
Versión	1. 0 (04/04/2022)
Dependencias	RINF-003
Descripción	El sistema deberá solicitar esta información cuando se quiera iniciar el proceso de editar un fármaco del sistema. En concreto:
Datos específicos	• Nuevas presentaciones comerciales (medida en mg/mL o µg/mL). • Nueva dosis máxima
Prioridad	Alta

Tabla 18. RINF-008: Información edición de fármacos

RINF-009	***Información registro de un usuario***

Versión	1. 0 (04/04/2022)
Dependencias	RINF-003
Descripción	El sistema deberá solicitar esta información cuando se quiera iniciar el proceso de registrar a un nuevo usuario. En concreto:
Datos específicos	• Datos del RINF-001 • Nombre del hospital asociado al nuevo usuario
Prioridad	Alta

Tabla 19. RINF-009: Información registro de un usuario

RINF-010	***Información inicio de sesión***
Versión	1. 0 (04/04/2022)
Descripción	El sistema deberá solicitar esta información cuando un usuario quiera acceder al mismo. En concreto:
Datos específicos	• Dirección de correo electrónico • Contraseña
Prioridad	Alta

Tabla 20. RINF-010: Información inicio de sesión

RINF-011	***Información resumida fármaco***
Versión	1. 0 (04/04/2022)
Descripción	El sistema ser capaz de proporcionar la información más relevante de un fármaco.
Datos específicos	• Nombre del fármaco • Dosis máxima diaria (medida en mg/día o µg/día)
Prioridad	Alta

Tabla 21. RINF-011: Información resumida fármaco

RINF-012	***Información receta***
Versión	1. 0 (04/04/2022)
Descripción	El sistema debe ser capaz de proporcionar la información relacionada con una receta. Una receta indica la cantidad de cada uno de los fármacos de un tratamiento específico que se deberá añadir a la bomba de infusión.
Datos específicos	Listado de: • Nombre del fármaco • Carga del fármaco en la bomba de infusión (medida en mL)

Prioridad	Alta

Tabla 22. RINF-012: Información receta

3.1.7.2 Requisitos de reglas de negocio

Las reglas de negocio corresponden con las restricciones o condiciones que debe satisfacer el sistema.

RN-001	***Restricción usuarios y hospitales***
Versión	1. 0 (04/04/2022)
Dependencias	RINF-001, RINF-002
Descripción	El sistema deberá respetar la siguiente regla de negocio: - *Un usuario solo podrá estar asociado a un hospital* - *Un hospital podrá tener asociados uno o más usuarios.*
Prioridad	Alta

Tabla 23. RN-001: Restricción usuarios y hospitales

RN-002	***Restricción usuarios y roles***
Versión	1. 0 (04/04/2022)
Dependencias	RINF-001, RINF-004
Descripción	El sistema deberá respetar la siguiente regla de negocio: *un usuario solo podrá tener un rol dentro del sistema.*
Prioridad	Alta

Tabla 24. RN-002: Restricción usuarios y roles

RN-003	***Restricción fármacos y hospitales***
Versión	2. 0 (05/04/2022)
Dependencias	RINF-002, RINF-003
Descripción	El sistema deberá respetar la siguiente regla de negocio: - *Un hospital podrá tener asociados más de un fármaco.* - *El catálogo de fármacos será exclusivo e independiente para cada uno de los hospitales existentes en el sistema. De este modo, cada hospital podrá indicar propiedades específicas para cada fármaco.*
Prioridad	Alta

Tabla 25. RN-003: Restricción fármacos y hospitales

RN-004	***Identificación única de los usuarios en el sistema***

Versión	2. 0 (05/04/2022)
Dependencias	RINF-001
Descripción	El sistema deberá respetar la siguiente regla de negocio: *los diferentes usuarios del sistema se identificarán unívocamente a partir de su dirección de correo electrónico. De este modo, no podrán registrarse dos usuarios con el mismo email en el sistema.*
Prioridad	Alta

Tabla 26. RN-004: Identificación única de los usuarios en el sistema

RN-005	***Asociación de roles de usuarios***
Versión	2. 0 (05/04/2022)
Dependencias	RINF-001
Descripción	El sistema deberá respetar la siguiente regla de negocio: *Un rol será asignado a un usuario cuando se registre en el sistema en función de si se asocia a un hospital ya existente en el sistema o de si indica un nuevo hospital.*
Prioridad	Alta

Tabla 27. RN-005: Asociación de roles de usuarios

3.1.7.3 Requisitos funcionales

Los requisitos funcionales son aquellos que establecen cómo se debe comportar el sistema y qué se espera de él cuando esté en funcionamiento.

RF-001	***Obtención del catálogo de fármacos hospital***
Versión	1. 0 (07/04/2022)
Dependencias	OBJ-02, SD-03
Descripción	El sistema deberá ser capaz de obtener todos los fármacos existentes en su sistema de persistencia y que, además, estén asociados al hospital al que pertenece el usuario que solicita esta visualización. Se deberá obtener un listado con la información resumida de cada fármaco (RINF-011).
Prioridad	Alta

Tabla 28. RF-001: Obtención del catálogo de fármacos hospital

RF-002	***Inserción de fármacos en el catálogo de un hospital***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-02, SD-03
Descripción	El sistema deberá permitir añadir nuevos fármacos a su sistema de persistencia y asociarlos a un determinado hospital.

Prioridad	Alta
Comentario	Esta funcionalidad es exclusiva para los usuarios gestores del sistema.

Tabla 29. RF-003: Inserción de fármacos en el catálogo de un hospital

RF-003	***Búsqueda de un fármaco dentro del catálogo***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-02, SD-03
Descripción	El sistema deberá permitir filtrar el listado de fármacos de un catálogo indicando el nombre del elemento que se pretende encontrar.
Prioridad	Alta

Tabla 30. RF-003: Búsqueda de un fármaco dentro del catálogo

RF-004	***Consulta de los detalles de un fármaco***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-02, SD-03
Descripción	El sistema deberá ser capaz de obtener un determinado fármaco (RINF-003) de su sistema de persistencia con el fin de poder conocer la información específica del medicamento.
Prioridad	Alta

Tabla 31. RF-004: Consulta de los detalles de un fármaco

RF-005	***Modificación de la información de un fármaco***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-02, SD-03
Descripción	El sistema deberá permitir actualizar la información relacionada con las propiedades del fármaco (las presentaciones comerciales y la dosis máxima).
Prioridad	Alta
Comentario	Esta funcionalidad es exclusiva para los usuarios gestores del sistema. El nombre del fármaco no se podrá modificar. Si fuese necesaria su modificación, habría que eliminar el fármaco en cuestión y recrearlo indicando el nuevo nombre.

Tabla 32. RF-005: Modificación de la información de un fármaco

RF-006	***Eliminación de un fármaco del catálogo***
Versión	1. 0 (08/04/2022)

Dependencias	OBJ-02, SD-03
Descripción	El sistema deberá permitir eliminar fármacos existentes en el catálogo.
Prioridad	Alta
Comentario	Esta funcionalidad es exclusiva para los usuarios gestores del sistema.

Tabla 33. RF-006: Eliminación de un fármaco del catálogo

RF-007	***Registro de un usuario***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-05, SD-04
Descripción	El sistema debe permitir crear nuevos usuarios y añadirlos al sistema de persistencia.
Prioridad	Alta

Tabla 34. RF-007: Registro de un usuario

RF-008	***Autenticación de usuarios***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-05, SD-04
Descripción	El sistema debe permitir comprobar que un usuario es quien dice ser cuando este intenta acceder al mismo.
Prioridad	Alta

Tabla 35. RF-008: Autenticación de usuarios

RF-009	***Autorización de usuarios***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-05, SD-04
Descripción	El sistema debe permitir conceder acceso a sus diferentes secciones según los permisos asociados al rol de cada usuario.
Prioridad	Alta

Tabla 36. RF-009: Autorización de usuarios

RF-010	***Asociación de un rol a un usuario***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-05, SD-04
Descripción	El sistema debe asociar un rol a un determinado usuario cuando este indica el hospital

	al que está asociado.
Prioridad	Alta

Tabla 37. RF:012: Asociación de un rol a un usuario

RF-011	***Cálculo de mezclas***
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-04, SD-01
Descripción	El sistema debe ser capaz de realizar todo el proceso matemático para obtener la mezcla que se añadirá a la bomba de infusión
Prioridad	Alta

Tabla 38. RF:011: Cálculo de mezclas

RF-012	**Cálculo de nueva dosis**
Versión	1. 0 (08/04/2022)
Dependencias	OBJ-05, SD-04
Descripción	El sistema debe ser capaz de realizar todo el proceso matemático para calcular cuál será la nueva dosis de un fármaco cuando varía la dosis de otro de los fármacos del tratamiento.
Prioridad	Alta

Tabla 39. RF:012: Cálculo de nueva dosis

3.1.7.4 Requisitos no funcionales

RNF-001	**Interfaz de usuario**
Versión	1. 0 (08/04/2022)
Tipo	Usabilidad
Descripción	El sistema debe ofrecer una interfaz de usuario intuitiva de modo que su uso resulte sencillo para los usuarios finales.

Tabla 40. RNF-001: Interfaz de usuario (usabilidad)

RNF-002	**Compatibilidad**
Versión	1. 0 (08/04/2022)
Tipo	Portabilidad
Descripción	El sistema debe ser soportado por el sistema operativo Android.

Tabla 41. RNF-002: Compatibilidad (portabilidad)

RNF-003	Fiabilidad de los cálculos
Versión	1. 0 (08/04/2022)
Tipo	Portabilidad
Descripción	El sistema debe garantizar que los cálculos que realice relacionados con el ámbito cínico deben ser correctos y estar libre de errores.

Tabla 42. RNF-003: Fiabilidad de los cálculos

RNF-004	Contraseñas cifradas
Versión	1. 0 (08/04/2022)
Tipo	Seguridad
Descripción	El sistema deberá guardar las contraseñas de los usuarios usando una función hash para garantizar la integridad de esta. El hash guardado se comparará con el hash de la contraseña introducida en el momento de autenticación.

Tabla 43. Contraseñas cifradas

3.1.8 Casos de uso

En este apartado se van a describir los casos de uso y los actores que estarán implicados en el sistema.

3.1.8.1 Diagrama de casos de uso

En el siguiente diagrama UML podemos ver de forma genérica la relación entre los distintos subsistemas, actores y casos de uso del sistema.

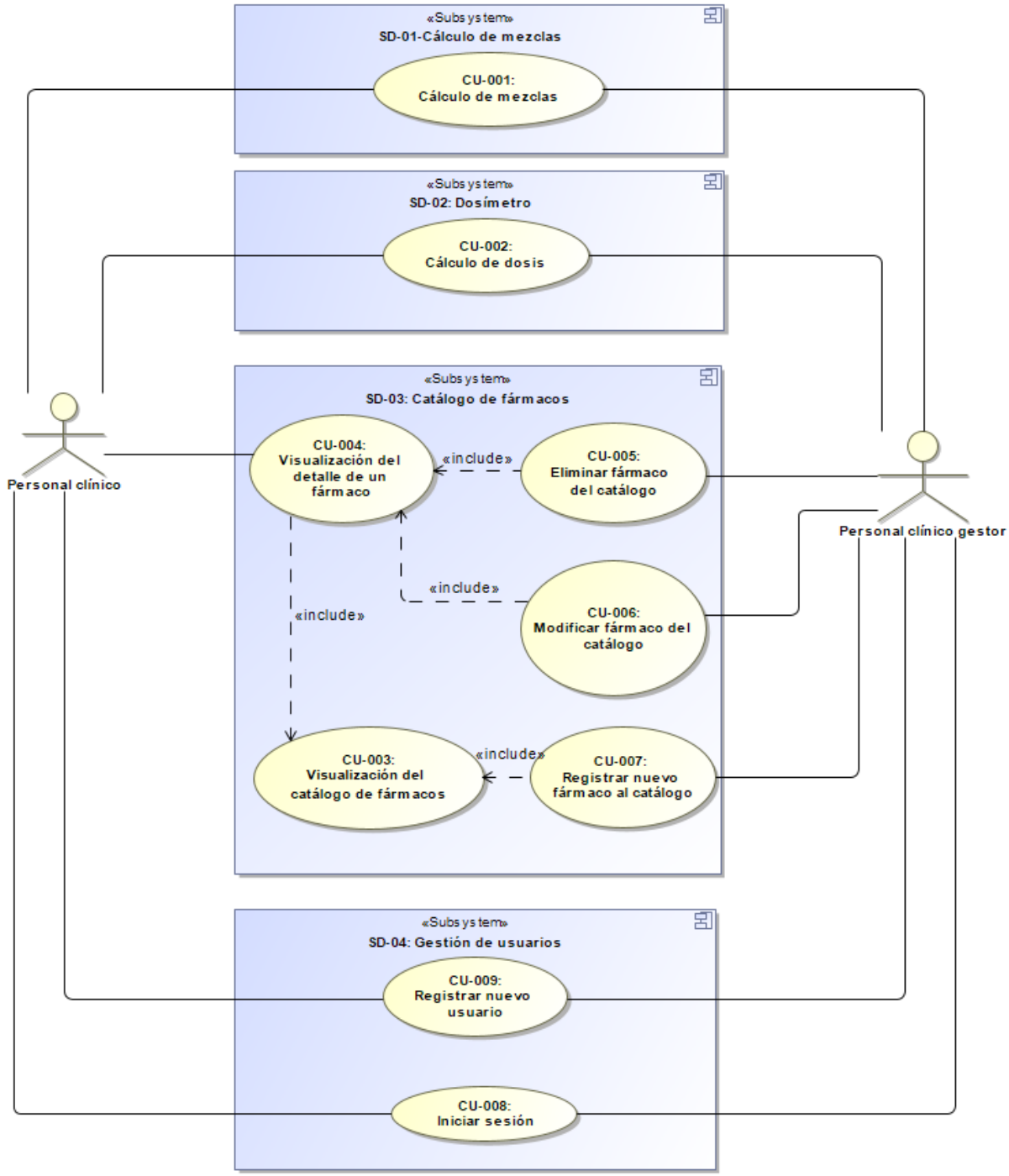


Figura 3-1. Diagrama de casos de uso del sistema

3.1.8.2 Actores del Sistema

Un actor describe el rol que desempeña un usuario externo al sistema durante una interacción con el mismo, en este caso, se ha detectado la necesidad de existencia de 2 roles que se describirán a continuación.

ACT-01	**Personal clínico**

Versión	1. 0 (08/04/2022)
Descripción	Este actor representa a cualquier profesional clínico del dolor que interactúa con el sistema con el sistema con el fin de agilizar su trabajo. No tendrá acceso a todas las funcionalidades del sistema. En concreto, no podrá realizar ninguna operación relacionada con la alteración del catálogo de fármacos del sistema.

Tabla 44. ACT-01: Personal clínico

ACT-02	***Personal clínico gestor***
Versión	1. 0 (08/04/2022)
Descripción	Este actor representa a cualquier profesional clínico del dolor que interactúa con el sistema con el fin de agilizar su trabajo y de mantener el catálogo de fármacos de un hospital actualizado. Tendrá acceso a todas las funcionalidades del sistema y será el encargado de mantener el catálogo de fármacos del hospital al que esté asociado.

Tabla 45. ACT-02: personal clínico gestor

3.1.8.3 Casos de uso del sistema

Los casos de uso describen cuál será la secuencia de acciones que garantizará el cumplimiento de un requisito funcional. En las siguientes tablas se podrán visualizar todos los detalles relacionados con los casos de uso detectados.

CU-001	***Cálculo de mezclas***	
Versión	1. 0 (09/04/2022)	
Dependencias	RINF-003, RINF-005, RINF-012, ACT-01, ACT-02	
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa y deberá haber accedido al menú principal.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite realizar un cálculo de mezclas.*	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de “Cálculo de mezclas”.
	2	El sistema solicita que el usuario rellene los campos de los datos necesarios para poder realizar el cálculo de mezclas correctamente (RINF-005)
	3	El usuario proporciona todos los campos solicitados.
	4	El sistema deberá ir validando los datos a medida que se vayan introduciendo y advertir de posibles incongruencias.
	5	El usuario deberá seleccionar la opción de “Calcular receta”
	6	El sistema deberá validar los datos indicados por el usuario.

	7	Si todos los datos son válidos:	
		7.1	El sistema realiza los cálculos pertinentes para obtener la mezcla o receta final.
		7.2	El sistema muestra el resultado final con la receta al usuario (RINF-012).
	8	Si se produce algún error en el sistema o en los datos introducidos, el sistema mostrará un mensaje de aviso.	
Prioridad	Muy alta		

Tabla 46. CU-001: Cálculo de mezclas

CU-002	***Cálculo de dosis***	
Versión	1. 0 (09/04/2022)	
Dependencias	RINF-003, RINF-006, ACT-01, ACT-02	
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa y deberá haber accedido al menú principal.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite realizar un cálculo de dosis.*	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de "Cálculo de dosis".
	2	El sistema solicita que el usuario rellene los campos de los datos necesarios para poder realizar el cálculo de dosis correctamente (RINF-006)
	3	El usuario proporciona todos los campos solicitados.
	4	El sistema deberá ir validando los datos a medida que se vayan introduciendo y advertir de posibles incongruencias.
	5	El usuario deberá seleccionar la opción de "Calcular"
	6	El sistema deberá validar los datos indicados por el usuario.
	7	Si todos los datos son válidos, el sistema realizará los cálculos pertinentes para obtener la nueva dosis buscada por el usuario y se la muestra al usuario.
	8	Si se produce algún error en el sistema o en los datos introducidos, el sistema mostrará un mensaje de aviso.
Prioridad	Muy alta	

Tabla 47. CU-002: Cálculo de dosis

CU-003	***Visualización del catálogo de fármacos***
Versión	1. 0 (09/04/2022)

Dependencias	RINF-003, RINF-011, ACT-01, ACT-02		
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa y deberá haber accedido al menú principal.		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite visualizar los fármacos existentes en el hospital al que está asociado.*		
Secuencia normal	**Paso**	**Acción**	
	1	El usuario selecciona la opción de "Catálogo".	
	2	El sistema deberá obtener el hospital al que está asociado el usuario actual de su sistema de persistencia.	
	3	El sistema deberá obtener los fármacos asociados al hospital al que está asociado el usuario actual.	
	4	El sistema deberá mostrar el listado con la información resumida de los fármacos existentes (RINF-011) al usuario y ofrecer un sistema de búsqueda de fármacos a partir de su nombre.	
	5	El usuario podrá seleccionar cualquiera de los elementos mostrados para visualizarlo en detalle (CU-004).	
	6	Si el usuario introduce alguna palabra en el buscador	
		6.1	El sistema deberá filtrar la lista de fármacos por nombre y mostrársela al usuario.
	7	Si el rol del usuario es "Gestor" (ACT-02)	
		7.1	El sistema deberá dar la opción al usuario de registrar un nuevo fármaco en el catálogo (CU-007).
Prioridad	Muy alta		

Tabla 48. CU-003: Visualización del catálogo de fármacos

CU-004	***Visualización de un fármaco de forma detallada***		
Versión	1. 0 (09/04/2022)		
Dependencias	RINF-003, ACT-01, ACT-02, CU-003		
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa y deberá estar visualizando el catálogo de fármacos (CU-003)		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite visualizar la información completa de un fármaco.*		
Secuencia normal	**Paso**	**Acción**	
	1	El usuario selecciona uno de los fármacos del catálogo.	

	2	El sistema deberá obtener la información completa del fármaco seleccionado a partir de su sistema de persistencia.	
	3	El sistema deberá mostrar el fármaco seleccionado en detalle (RINF-003).	
	4	Si el rol del usuario es “Gestor” (ACT-02)	
		4.1	El sistema deberá dar la opción al usuario de editar (CU-005) o de eliminar (CU-006) ese fármaco.
Prioridad	Muy alta		

Tabla 49. CU-004: Visualización de un fármaco de forma detallada

CU-005	***Eliminar fármaco del catálogo de fármacos***	
Versión	1. 0 (09/04/2022)	
Dependencias	RINF-003, ACT-02, CU-004	
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa con el rol de “Gestor” y deberá estar visualizando un fármaco de forma detallada (CU-004)	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite eliminar un fármaco.*	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa el botón de “eliminar” indicando el deseo de eliminar el fármaco que se está visualizando.
	2	El sistema deberá comprobar si el usuario tiene el rol de “Gestor” (ACT-002). Si es así, la operación será autorizada y procederá a eliminar el fármaco seleccionado de su sistema de persistencia.
	3	Si no se produce ningún error, el sistema redirige al usuario al catálogo de fármacos y le notifica del éxito de la operación.
Prioridad	Muy alta	

Tabla 50. CU-005: Eliminar fármaco del catálogo de fármacos

CU-006	***Modificar fármaco del catálogo de fármacos***
Versión	1. 0 (09/04/2022)
Dependencias	RINF-003, ACT-02, CU-004
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa con el rol de “Gestor” y deberá estar visualizando un fármaco de forma detallada (CU-004)
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite modificar las propiedades de un fármaco.*

Secuencia normal	**Paso**	**Acción**	
	1	El usuario pulsa el botón de "editar" indicando el deseo de modificar el fármaco que se está visualizando.	
	2	El sistema solicitará al usuario que rellene los campos con las modificaciones del fármaco (RINF-008).	
	3	El usuario debe proporcionar los datos solicitados.	
	4	El sistema deberá ir validando los datos a medida que se vayan introduciendo y advertir de posibles incongruencias.	
	5	El usuario deberá seleccionar la opción de "Aceptar" para confirmar la operación	
	6	Si todos los datos son válidos	
		6.1	El sistema deberá comprobar si el usuario tiene el rol de "Gestor" (ACT-002). Si es así, la operación será autorizada y procederá a actualizar el fármaco seleccionado de su sistema de persistencia.
		6.2	El sistema mostrará la información detallada del fármaco (RINF-003) con la información actualizada.
	7	Si se produce algún error en el sistema o en los datos introducidos, el sistema mostrará un mensaje de aviso.	
Prioridad	Muy alta		

Tabla 51. CU-006: Modificar fármaco del catálogo de fármacos

CU-007	***Registro de nuevo fármaco***		
Versión	1. 0 (09/04/2022)		
Dependencias	RINF-007, ACT-02, CU-003		
Precondición	El usuario deberá haber iniciado sesión en el sistema de manera exitosa con el rol de "Gestor" y deberá estar visualizando el catálogo de fármacos (CU-003)		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite insertar un nuevo fármaco.*		
Secuencia normal	**Paso**	**Acción**	
	1	El usuario pulsa el botón de "añadir" indicando el deseo de insertar un nuevo fármaco al catálogo de fármacos de su hospital.	
	2	El sistema solicitará al usuario que rellene los campos con los datos del nuevo fármaco (RINF-007).	
	3	El usuario debe proporcionar los datos solicitados.	
	4	El sistema deberá ir validando los datos a medida que se vayan introduciendo y advertir de posibles incongruencias.	

	5	El usuario deberá seleccionar la opción de “Aceptar” para confirmar la operación	
	6	Si todos los datos son válidos	
		6.1	El sistema deberá comprobar si el usuario tiene el rol de “Gestor” (ACT-002). Si es así, la operación será autorizada y procederá a añadir el nuevo fármaco a su sistema de persistencia.
		6.2	El sistema volverá a mostrar el catálogo de fármacos (CU-003)
	7	Si se produce algún error en el sistema o en los datos introducidos, el sistema mostrará un mensaje de aviso.	
Prioridad	Muy alta		

Tabla 52. CU-007: Registro de nuevo fármaco

CU-008	***Inicio de sesión***	
Versión	1. 0 (09/04/2022)	
Dependencias	RINF-010, ACT-01, ACT-02, CU-003	
Precondición	El usuario accede al inicio del sistema.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite ingresar dentro del sistema.*	
Secuencia normal	**Paso**	**Acción**
	1	El usuario deberá pulsar la opción de “Iniciar sesión”.
	2	El sistema solicitará al usuario que inserte sus credenciales (RINF-010)
	3	El usuario deberá proporcionar los datos solicitados.
	4	El sistema deberá comprobar si ese usuario existe en su sistema de persistencia y las credenciales proporcionadas son correctas.
	5	Si la autenticación es exitosa
		5.1 El sistema permite al usuario el acceso a su interior.
		5.2 El sistema redirige al usuario al catálogo de fármacos (CU-003)
		Si la autenticación es errónea, el sistema avisa al usuario.
	6	El sistema debe asociar una fecha de expiración para la sesión creada.
Prioridad	Muy alta	

Tabla 53. CU-008: Inicio de sesión

CU-009	***Registrar nuevo usuario***

Versión	1. 0 (09/04/2022)		
Dependencias	RINF-001, RINF-002, ACT-01, ACT-02		
Precondición	El usuario accede al inicio del sistema.		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando *el usuario solicite registrarse en el sistema*		
Secuencia normal	**Paso**	**Acción**	
	1	El usuario deberá seleccionar la opción de "Registrarse".	
	2	El sistema solicitará al usuario que inserte sus datos (RINF-001)	
	3	El usuario deberá proporcionar los datos solicitados.	
	4	El sistema deberá validar que no existe ningún otro usuario en su sistema de persistencia con la misma dirección de correo electrónico.	
	5	El sistema busca en su sistema de persistencia los hospitales que existen actualmente y solicita al usuario que seleccione uno de estos hospitales o inserte el nombre de uno nuevo (RINF-002).	
	6	El usuario elige el hospital al que adherirse.	
	7	Si el usuario selecciona un hospital que ya existe en el sistema de persistencia.	
		7.1	El sistema registrará al nuevo usuario con el rol de "Usuario genérico" (ACT-01).
		Si el usuario indica un nuevo hospital.	
		7.1	El sistema registrará al nuevo usuario con el rol de "Usuario genérico" (ACT-01).
	8	El usuario queda registrado en el sistema y se inicia sesión (CU-009)	
Prioridad	Muy alta		

Tabla 54. CU-009: Registrar nuevo usuario

3.2 Diseño del sistema

Una vez analizados y definidos los requisitos del sistema, se debe realizar la tarea de diseño que consiste en describir y crear un modelo especificando cómo deberá ser implementado el sistema. En este apartado se detallarán aspectos de diseño del modelo de datos, la arquitectura, la aplicación móvil, el servicio web y la seguridad del sistema. Además, se incluirán diagramas que permitan entender de manera visual y con un alto nivel de abstracción cómo se debe comportar el sistema ante diferentes situaciones.

3.2.1 Diseño del modelo de datos

[21] Un modelo de datos permite especificar de forma clara los datos con los que se deberá trabajar, las asociaciones entre ellos, su semántica asociada y las restricciones de integridad. Según el nivel de abstracción en el que se manejan los datos, los modelos de datos se clasifican en dos grupos:

1. **Modelos lógicos de datos.** Este tipo de modelado se usa para describir los datos a nivel conceptual y externo. Se caracterizan por su capacidad expresiva, su flexibilidad, su simplicidad y su formalidad. Existen diferentes técnicas de modelado lógico entre las que destacan: el modelo E-R (Entidad-Asociación) y el modelo orientado a objetos.
2. **Modelos implementables de datos.** Este tipo de modelado se usa para describir datos a nivel conceptual e interno. Se caracterizan por su capacidad de especificar la estructura lógica global y de describir el modelo a nivel de implementación (almacenamiento, restricciones …). Existen diferentes técnicas de modelado implementable entre las que destacan: el modelo jerárquico, el modelo en red y el modelo relacional.

3.2.1.1 Modelo lógico de datos: modelo E-R

En el presente trabajo se ha optado por describir el modelo lógico de datos a partir del modelo E-R. El modelo E-R (Entidad-Asociación o *Entity-Relationship*) es un mecanismo formal para representar y manipular información de manera sistemática y genérica. Este modelo cumple con las siguientes características:

- Refleja la existencia de datos, no lo que se hace con ellos.
- Es independiente del sistema operativo y del SGDB (Sistema Gestor de Base de Datos) que se empleen posteriormente.
- Es independiente de las restricciones de almacenamiento y tiempo de ejecución.
- Permite la especificación de restricciones.

Por otra parte, el modelo E-R se suele representar gráficamente a partir de los siguientes elementos:

- Entidades: se trata de objetos concretos o abstractos que pueden distinguirse de cualquier otro objeto.
- Atributos: corresponden a las propiedades que caracterizan un conjunto de entidades.
- Dominio: es el conjunto de valores permitidos para un determinado atributo.
- Asociaciones: definen las relaciones entre las distintas entidades.
- Cardinalidad de una asociación: expresa el número máximo de entidades que puede asociarse con otra entidad. Existen relaciones “muchos a muchos” (m:n), “uno a muchos” (1:m) y “uno a uno” (1:1).
- Clave primaria: atributo o conjunto de atributos que permiten identificar de manera unívoca una entidad.

En el diagrama de la *Figura 3-2 s*e podrá observar el modelo E-R que representa el modelo lógico de datos del sistema que se está diseñando. Este modelo se ha diseñado a partir de los requisitos de información y de las reglas de negocio definidas en la sección de *Análisis y definición de requisitos.*

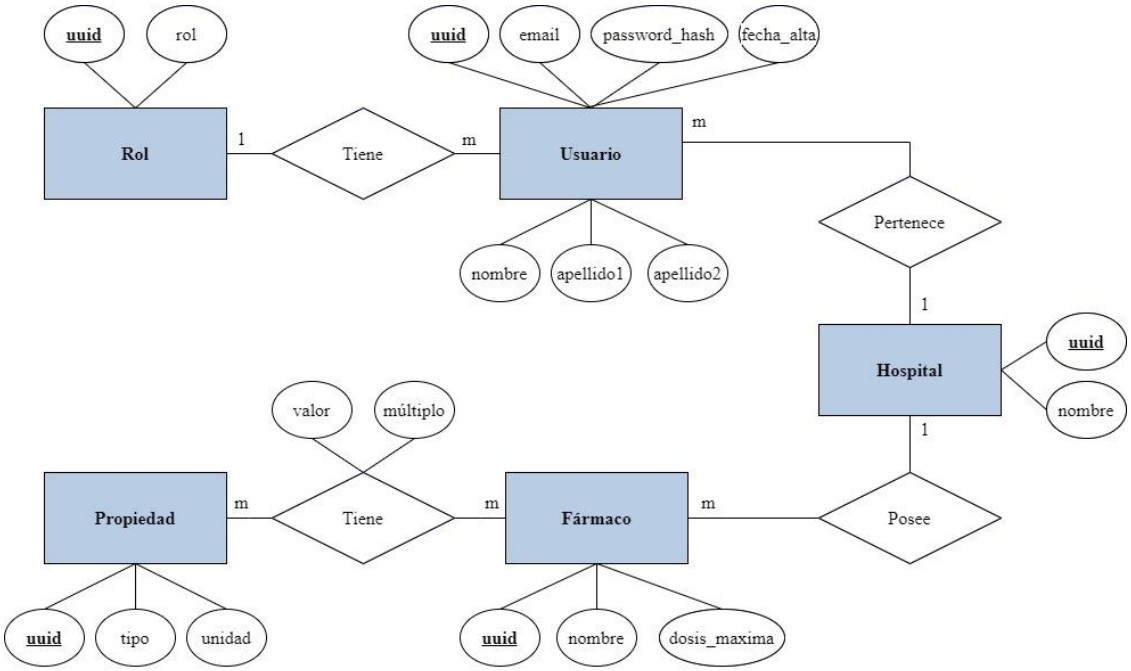


Figura 3-2. Diagrama modelo E-R del sistema

3.2.1.2 Modelo implementable de datos: modelo relacional

En relación al modelo implementable, se ha optado por describirlo a partir del modelo relacional que consiste en organizar los datos mediante tablas relacionadas unas con otras y se puede construir a partir del modelo E-R.

Tomando de base el diagrama de la *Figura 3-2,* se ha diseñado el modelo relacional del sistema que se está analizando, teniendo en cuenta los siguientes aspectos:

- Cada entidad del modelo E-R pasa a ser una tabla del modelo relacional manteniendo sus atributos y su clave primaria.
- Cada asociación del modelo E-R pasa a ser una tabla del modelo relacional. Sin embargo, los atributos de la tabla serán las claves primarias de las entidades asociadas. Por otra parte, la clave primaria de la tabla variará según la cardinalidad de las asociaciones. En concreto, estará compuesta por: los atributos propios de la asociación (opcionalmente), la unión de las claves primarias de las entidades involucradas en la asociación (si la cardinalidad es muchos a muchos), la clave primaria de la entidad que participa como "muchos" en la asociación (si la cardinalidad es muchos a uno), o la clave primaria de una de las entidades asociadas según convenga (si la cardinalidad es uno a uno).
- Es posible simplificar aquellas tablas que acaban teniendo la misma clave primaria uniéndolas.

De este modo, aplicando todo lo mencionado con anterioridad, se obtiene el modelo relacional simplificado de la *Tabla 55* que gráficamente se representa a través de la *Figura 3-3.*

P_ROL (**id_rol**, cd_rol, ds_rol)

D_HOSPITAL (**id_hospital**, nombre)

P_PROPIEDAD (**id_propiedad**, cd_tipo, ds_tipo)

R_FARMACO_PROPIEDAD(**id_farmaco** , **id_propiedad** , valor, unidad)

D_USUARIO (**id_usuario**, nombre, apellido1, apellido2, email, password_hash, fecha_alta, id_rol , id_hospital)

D_FARMACO (**id_farmaco**, nombre, id_hospital)

Tabla 55. Modelo relacional del sistema

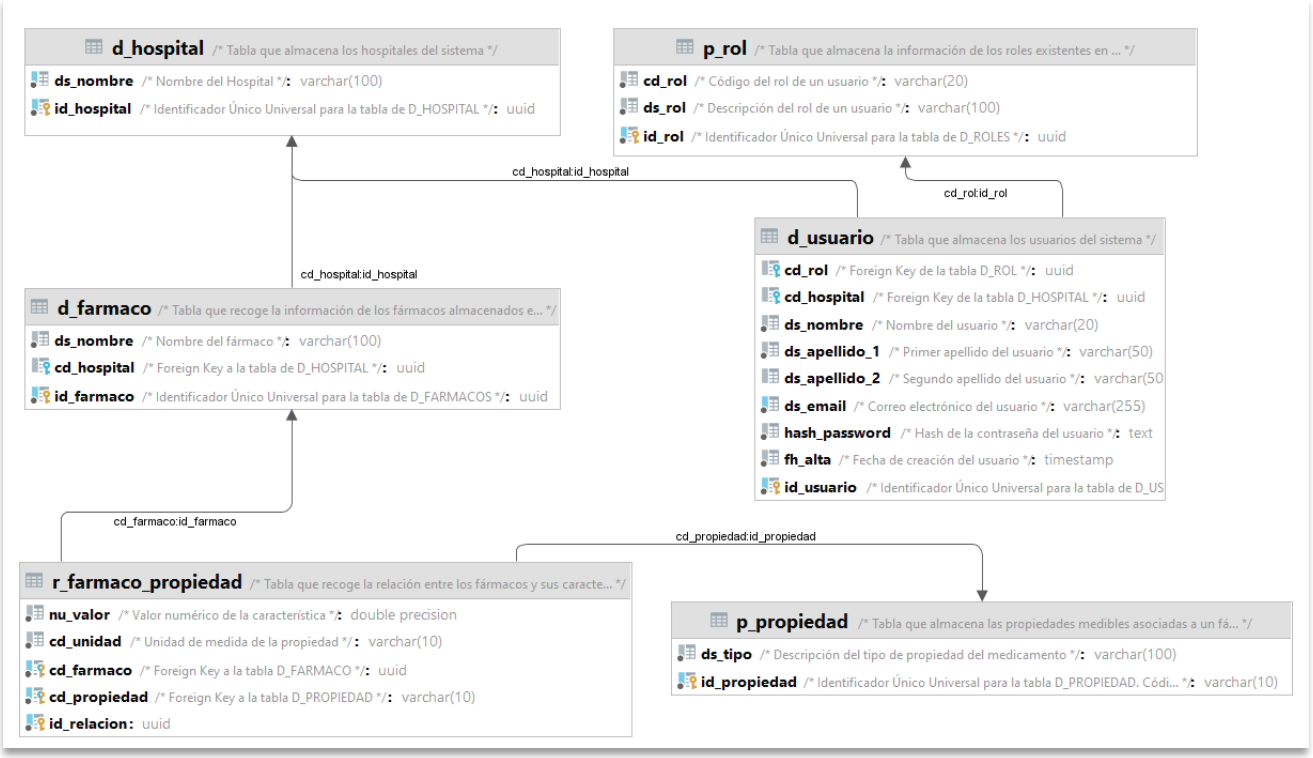


Figura 3-3. Diagrama del modelo relacional del sistema

3.2.2 Arquitectura del sistema

La arquitectura de un software define con un alto nivel de abstracción los componentes que van a formar parte del sistema, sus tareas y la comunicación entre ellos. Debe definirse teniendo en cuenta el conjunto de requisitos existentes. En este proyecto se ha optado por una arquitectura Cliente/Servidor siguiendo el modelo en tres capas que son las siguientes [22]:

- **Capa de presentación:** es la interfaz de usuario, es decir, el lugar donde el usuario final interactúa con la aplicación. Así, esta capa se encarga de entregar y solicitar información al usuario y de transformar sus solicitudes en acciones dirigidas a la capa de negocio.
- **Capa de negocio:** es el núcleo del sistema ya que se encarga de procesar la información recopilada en el nivel de presentación y de generar un resultado. También deberá interaccionar con el sistema de persistencia de forma transparente para la capa de presentación.
- **Capa de datos:** es donde se almacenará y gestionará la información persistente.

Una de las grandes ventajas de la arquitectura en 3 capas es la independencia entre las mismas. Las capas deben comunicarse a través de una interfaz bien definida pero lo que ocurra internamente en el interior de cada una de ellas no afectará al resto. De este modo, los sistemas con esta arquitectura serán reutilizables, mantenibles, escalables y flexibles.

Para el sistema que estamos modelando, la división por capas será implementada en los siguientes componentes:

- **Capa de presentación.** Estará representada por una aplicación móvil con sistema operativo Android. Se ha elegido este sistema operativo ya que la mayoría del personal médico de los hospitales andaluces dispone de una Tablet Android para el uso de aplicaciones como la que se va a desarrollar.
- **Capa de negocio.** Estará representada por un servidor web. Este servidor se implementará mediante el framework de Spring y publicará un servicio API RESTful que permitirá la comunicación con la capa de presentación. Así mismo, hará uso de la API de Persistencia JPA para interaccionar con la capa de datos.

- **Capa de datos.** Estará representada por el gestor de bases de datos PostgreSQL ya que ofrece numerosas ventajas para manejar modelos de datos relacionales.

En la *Figura 3-4* se puede observar de manera gráfica la arquitectura detallada en esta sección.

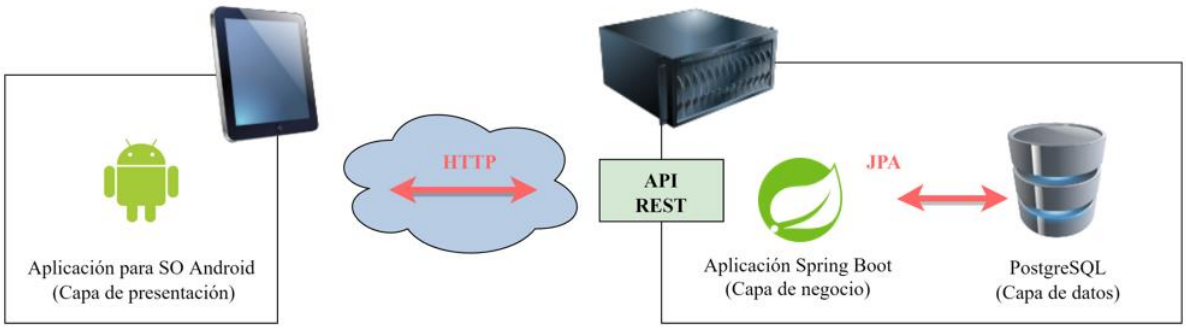


Figura 3-4. Diagrama arquitectura cliente/servidor en 3 capas

3.2.3 Diseño de la aplicación REST

El objetivo principal de esta aplicación, que se implementará usando el framework de Spring, es ofrecer un servicio REST que permita al cliente Android obtener y manipular datos del sistema de persistencia, además de conseguir los resultados requeridos para poder ofrecer cada una de las funcionalidades del sistema.

3.2.3.1 Estructura del Proyecto Spring

En primer lugar, se comenzará describiendo cuál deberá ser la estructura a nivel de paquetes del proyecto (*Figura 3-5*):

- **Controller:** deberá incluir todas las clases que actúen como controlador en Spring, es decir, aquellas clases cuya función sea responder a las peticiones que llegan al servidor. Dichas clases deberán validar los datos entrantes a nivel de formato y delegar cualquier operación relacionada con la lógica de negocio en su servicio asociado.
- **Service:** deberá incluir todas las clases que actúen como servicio en Spring, es decir, aquellas clases que contengan la lógica para satisfacer las peticiones que llegan al controlador. Dichas clases deberán validar los datos entrantes a nivel de negocio y usar la interfaz JPA ofrecida por las clases de tipo repositorio para interactuar con la base de datos.
- **Repository:** deberá incluir todas las clases que actúen como repositorio o DAO, es decir, aquellas clases que heredan de JPA con el fin de ofrecer una interfaz que permita la comunicación con la base de datos.
- **View:** contendrá el conjunto de clases que serán devueltas por el controlador en respuesta a una petición.
- **DTO:** contendrá el conjunto de clases que representan los datos de entrada provenientes del sistema que está consumiendo la API REST.
- **Entity:** contendrá el conjunto de clases que representan las tablas del modelo relacional de la base de datos como clases Java.
- **Validation:** contendrá aquellas clases y anotaciones cuyo fin sea validar que un conjunto de datos cumple con la lógica de negocio.
- **Function:** contendrá aquellas clases que permitan convertir un objeto de un tipo a otro. La conversión de "entity" a "view" sería un ejemplo de este tipo de clases.
- **Exception:** contendrá aquellas clases que modelen excepciones del sistema.
- **Security:** contendrá aquellas clases que permiten incorporar seguridad al sistema.

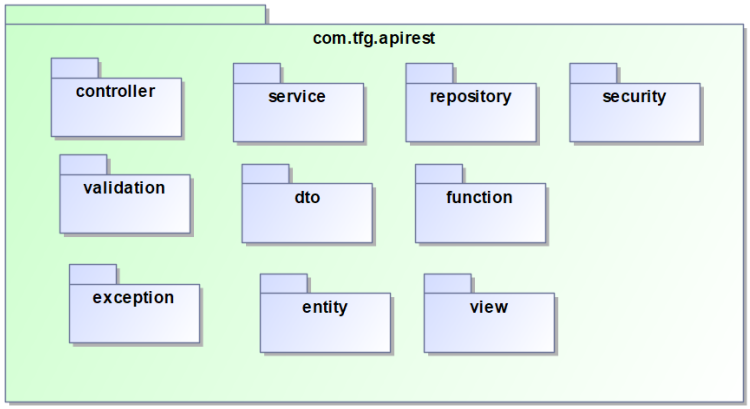


Figura 3-5. Diagrama de paquetes Servidor REST

3.2.3.2 Definición de la API REST

Se continuará describiendo cuáles deberán ser los diferentes endpoints ofrecidos por el servicio REST y que, por tanto, podrán ser accedidos desde cualquier sistema externo.

Método	URL	Acción y usuarios autorizados	Controlador
POST	**/auth/signin**	Inicio de sesión y obtención de token JWT.	**AuthenticationController**
POST	**/auth/signup**	Registro de nuevo usuario y obtención de token JWT.	
POST	**/calculo/mezclas**	Calcula mezcla de la bomba de infusión. **Authorities:** USER, GESTOR	**CalculadoraController**
POST	**/calculo/dosis**	Calcula nueva dosis de un fármaco. **Authorities:** USER, GESTOR	
GET	**/farmacos**	Obtiene el listado con la información resumida de los fármacos existentes en el hospital del usuario que realiza la consulta. **Authorities:** USER, GESTOR	**FarmacoController**
GET	**/fármacos/{id}**	Obtiene la información detallada de un fármaco concreto. **Authorities:** USER, GESTOR	
POST	**/farmacos**	Registra un nuevo fármaco en el hospital del usuario que realiza la petición. **Authorities:** GESTOR	
PUT	**/farmacos/{id}**	Modifica un fármaco concreto **Authorities:** GESTOR	
DELETE	**/farmacos/{id}**	Elimina un fármaco concreto **Authorities:** GESTOR	

GET	**/farmacos/nombres**	Obtiene el listado de nombres de todos los fármacos existentes en el hospital del usuario que realiza la consulta. **Authorities:** USER, GESTOR	
GET	**/hospitales**	Obtiene el listado con los nombres de hospitales existentes.	**HospitalCotroller**

Tabla 56. Definición de los endpoints del servicio REST

3.2.4 Diseño de la aplicación móvil

El objetivo principal de la aplicación móvil de Android es ofrecer una interfaz de usuario intuitiva y que permita realizar todas las operaciones expuestas en la toma de requisitos.

3.2.4.1 Estructura del Proyecto Android

En cuanto a la estructura del proyecto Android a nivel de paquetes, cabe destacar que todas las aplicaciones se crean con un árbol de directorios predefinido por Android. El código fuente y el resto de los recursos se encuentran en la carpeta src (sources) y, a continuación, se describirá brevemente su contenido Figura 3-6.

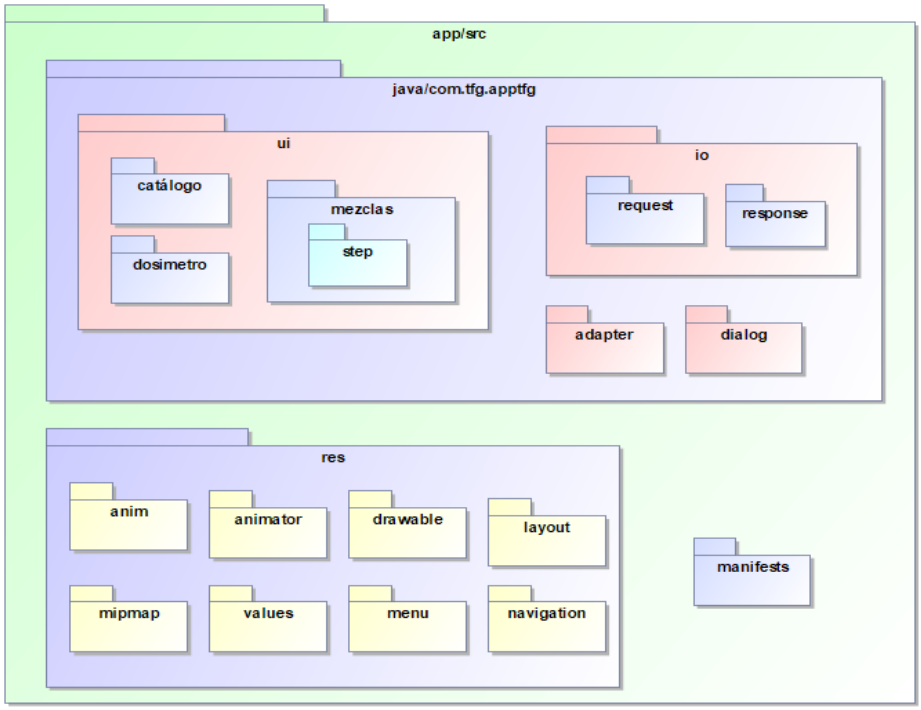


Figura 3-6: Diagrama de paquetes proyecto Android

- **Directorio Manifests:** contendrá el archivo *AndroidManifest.xml* donde se describe la información esencial para la creación de los diferentes elementos de la aplicación.
- **Directorio Java:** incluirá el código fuente de la aplicación:
- **UI:** contendrá aquellas clases relacionadas con el funcionamiento del menú lateral de la aplicación y sus secciones: dosímetro, cálculo de mezclas y catálogo.
- **IO:** contendrá las clases de entrada y salida de las peticiones a la aplicación REST, así como la interfaz y el adaptador que permitan llamar al servidor.
- **Adapter:** aquí se incluirán clases que permitan realizar adaptaciones como la visualización de una lista

en un RecyclerView.

- **Dialog:** aquí se incluirán clases que sirvan para administrar los diálogos que se vayan a mostrar en la aplicación.
- **Directorio Res:** incluirá todos los recursos que usará la aplicación para incluir ciertas características (interfaz, color, estilos, …).

3.2.5 Diseño de la seguridad

Con el fin de proteger los recursos que ofrecerá el servicio de API REST y de proporcionar un sistema de acceso controlado al sistema, se deberá usar JWT para autenticar y autorizar a los usuarios. El funcionamiento de este mecanismo está descrito en el apartado 2.2.1.6, deberá seguir el mismo patrón en el sistema a implementar.

3.2.6 Diseño del funcionamiento del proyecto

La intención de este apartado es mostrar el funcionamiento que debe tener el sistema ante diversas situaciones y desde diferentes puntos de vista. Esto será muy útil en la fase de implementación ya que servirá de base y de guía para escribir un código cuyo comportamiento sea el esperado.

3.2.6.1 Interacción entre los componentes de la aplicación de Spring (obtención listado fármacos):

En primer lugar, se mostrará (*Figura 3-7*) un diagrama de secuencia con un nivel de abstracción media. En este diagrama se detalla cómo se relacionan entre sí los diferentes componentes del servicio REST cuando se recibe una petición de un recurso por parte del cliente y se tiene que interaccionar con la base de datos para obtener la información solicitada. En concreto, se está representando la secuencia para obtener todos los fármacos existentes en el hospital del usuario que realiza la petición. Sin embargo, la secuencia en la que se ejecutan el resto de las operaciones CRUD (Create, Request, Update, Delete) sigue la misma filosofía.

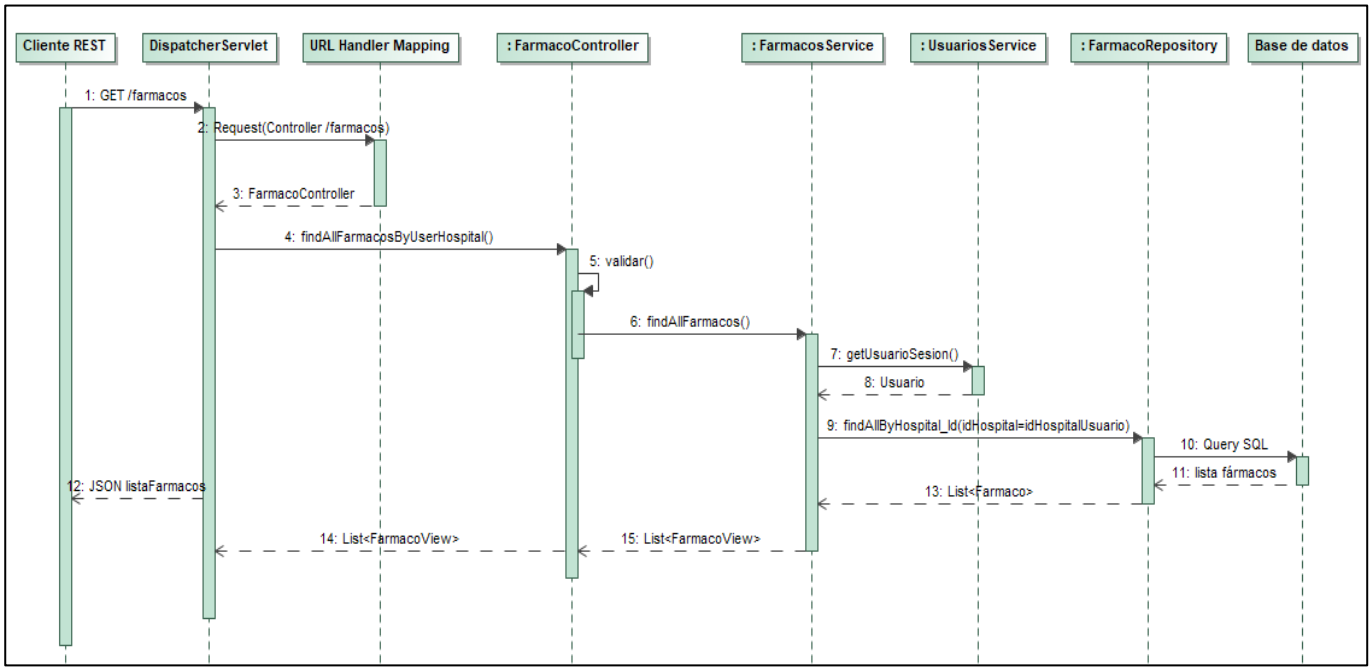


Figura 3-7. Diagrama de secuencia: Interacción entre componentes Spring (obtención listado fármacos)

3.2.6.2 Inicio de sesión en el sistema:

Se continuará con la descripción del funcionamiento del inicio de sesión (*Figura 3-8*). En este caso, se ha elegido un diagrama de actividad cuyos participantes son el usuario y los tres principales componentes del sistema: la aplicación móvil, la aplicación Spring y la base de datos. En esta representación podemos ver cómo el usuario interactúa con la interfaz de Android, la aplicación delega la autenticación en el servicio REST y se obtiene un

token JWT que permitirá acceder a los recursos protegidos de la API.

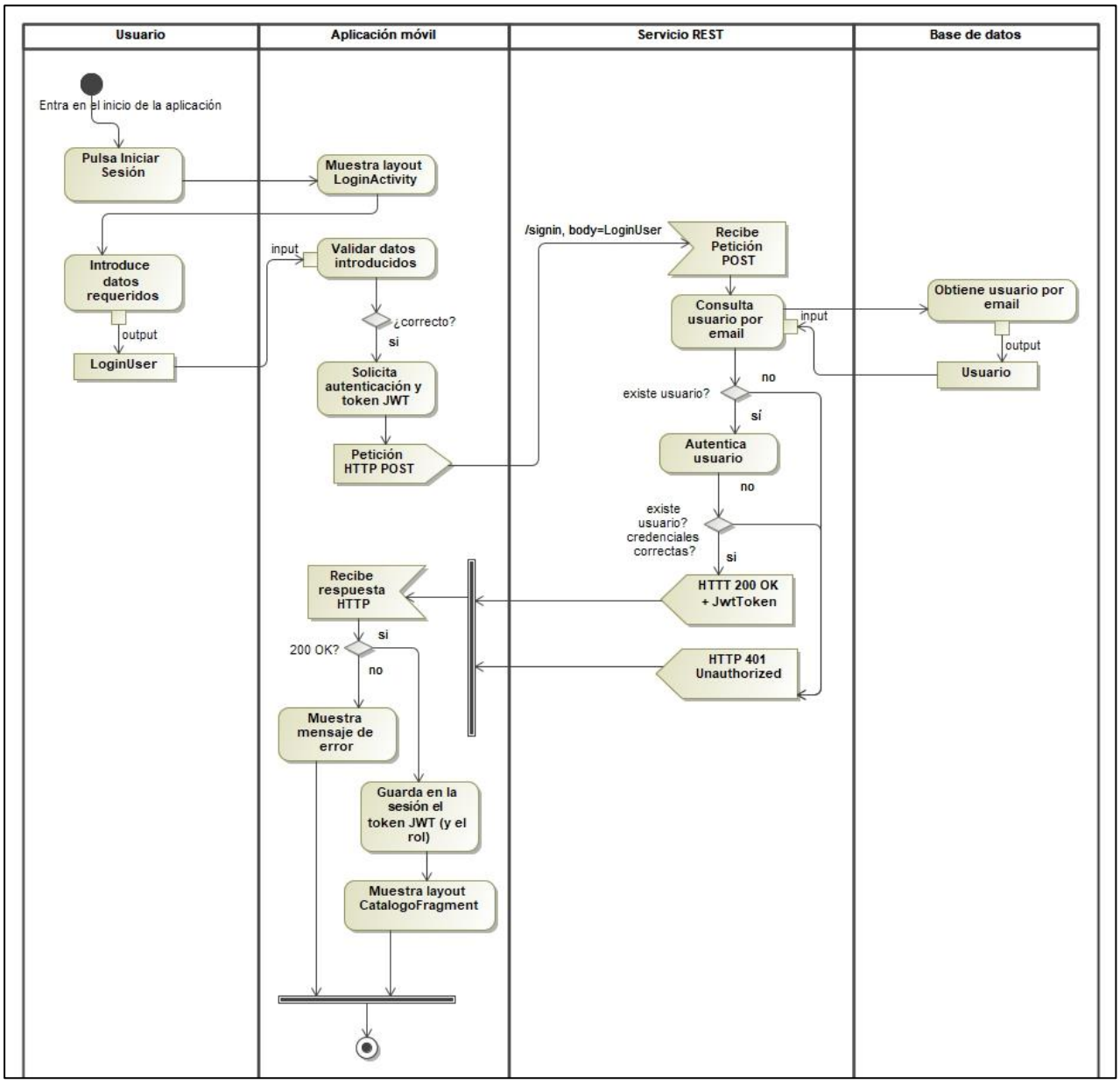


Figura 3-8. Diagrama de actividad: Inicio de sesión en el sistema

3.2.6.3 Catálogo de fármacos

El flujo de actividades relacionado con ofrecer las distintas funcionalidades del subsistema "Catálogo de fármacos" también resulta interesante que sea descrito. Se describirán las interacciones que permiten mostrar el listado de fármacos existentes en el hospital del usuario que realiza la petición (*Figura 3-9)*. El flujo de acciones a realizar será muy similar para el resto de las funcionalidades.

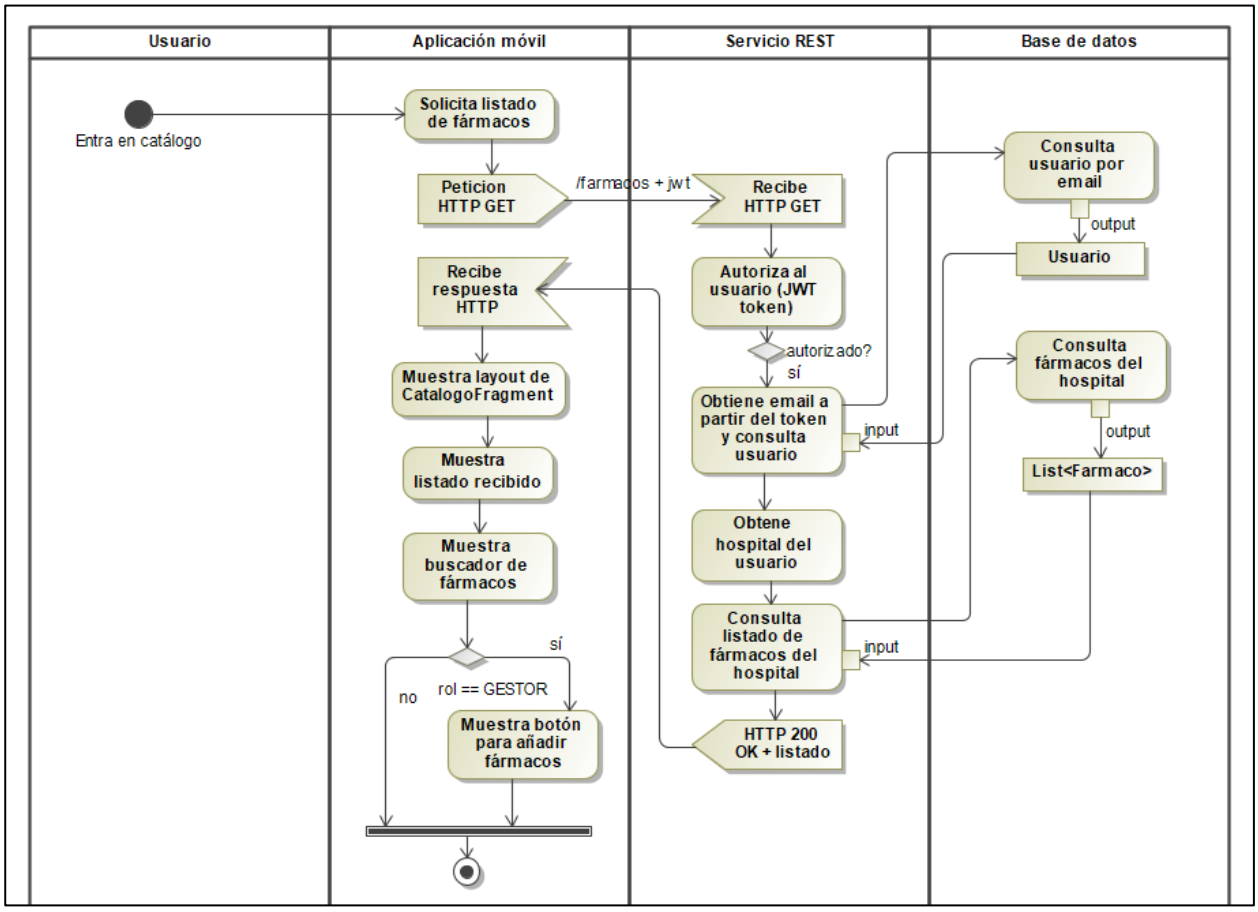


Figura 3-9. Diagrama de actividad: Mostrar catálogo de fármacos

3.2.6.4 Interacción entre los componentes de la aplicación de Android (cálculo de nuevas dosis):

La interacción entre los diferentes componentes de la aplicación móvil también conviene que sea definida. En el diagrama de secuencia de la (*Figura 3-10*) se puede observar cómo las acciones del usuario sobre la pantalla que está visualizando son respondidas por la actividad o el diagrama asociado al layout de la pantalla. Son estos componentes los que se encargan de validar lo que se introduce y de responder a los distintos eventos.

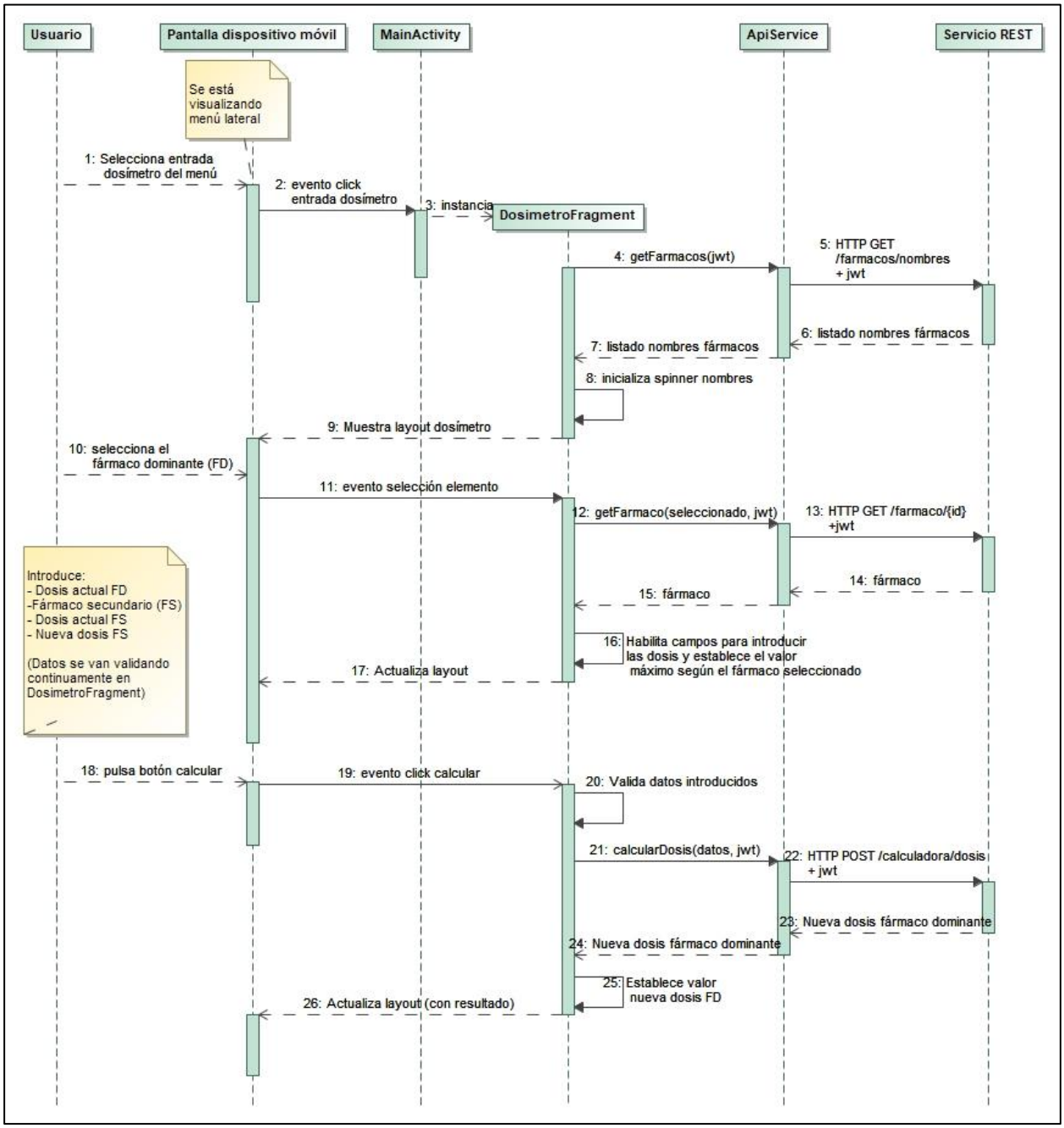


Figura 3-10. Diagrama de secuencia: Interacción entre componentes Aplicación Android (cálculo dosis)

3.3 Implementación

Una vez realizado el análisis de requisitos y el posterior diseño del sistema, se ha procedido a implementarlo. Cabe destacar que, aunque esta haya sido la última etapa, en algunas ocasiones se han tenido que modificar ciertos aspectos del diseño que no se habían definido de forma clara o que provocaban algún tipo de conflicto. En esta sección se describirá el proceso de implementación y se detallará el funcionamiento final de cada uno de los componentes del sistema.

3.3.1 Base de datos PostgreSQL

Para poner en marcha el SGDB de PostgreSQL, se ha usado Docker con el objetivo de simplificar la configuración y evitar tener que instalar el servicio y sus dependencias en el sistema operativo del ordenador que se está utilizando en el desarrollo. Además, para administrar la base de datos se creará también un contenedor

para un servidor PgAdmin. Para el despliegue de ambos servicios a través de contenedores Docker, se han seguido los pasos que se describirán a continuación.

1. Se ha descargado la imagen oficial de Postgres y de PgAdmin de Docker Hub.
2. Se ha definido la configuración que se deberá aplicar a las imágenes de Postgres y de PgAdmin, para crear los respectivos contenedores, en el fichero *docker-compose.yml*. La creación de este fichero permitirá lanzar estos dos servidores en cualquier sistema que use Docker sin tener que realizar ninguna otra configuración adicional. Esto facilitará el despliegue y mejorará la portabilidad del sistema.

```yaml
version: '3.5'

services:
  postgres:
    container_name: tfg_postgres
    image: postgres:latest
    environment:
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PW}
      - POSTGRES_DB=${POSTGRES_DB}
    ports:
      - "5432:5432"
    restart: always

  pgadmin:
    container_name: tfg_pgadmin
    image: dpage/pgadmin4:latest
    environment:
      - PGADMIN_DEFAULT_EMAIL=${PGADMIN_MAIL}
      - PGADMIN_DEFAULT_PASSWORD=${PGADMIN_PW}
    ports:
      - "5050:80"
    restart: always
```

Figura 3-11. Fichero docker-compose.yml

docker-compose.yml | .env

```
1  POSTGRES_USER=celllarod
2  POSTGRES_PW=admin
3  POSTGRES_DB=tfg_postgres
4  PGADMIN_MAIL=celllarod@alum.us.es
5  PGADMIN_PW=admin
```

Figura 3-12. Variables de entorno docker

3. Para poder conservar los datos del contenedor en caso de que este se pare, se ha creado un volumen. Esto consiste en montar una carpeta del sistema anfitrión (el ordenador de desarrollo, en este caso) en el sistema de ficheros del contenedor.
4. Se crean los contenedores con el comando *docker-compose up* (*Figura 3-13*).

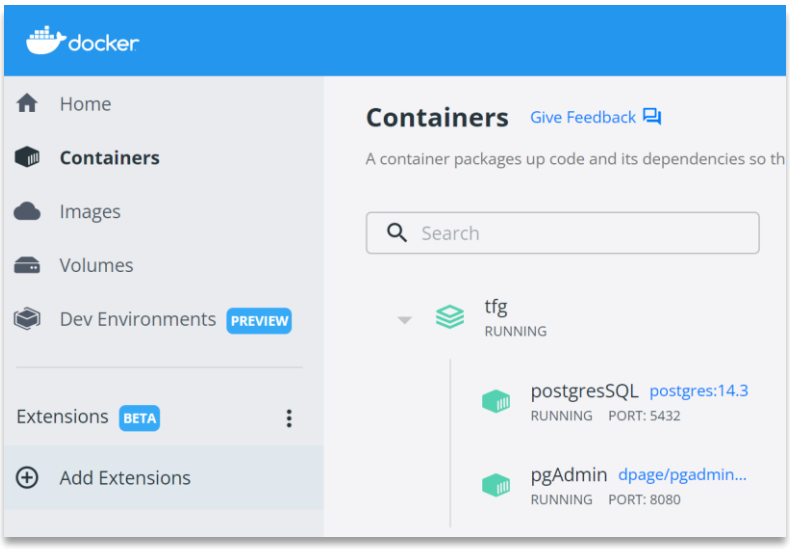


Figura 3-13. Contenedores Docker: postgreSQL y pgAdmin

Con esta configuración se consiguen ejecutar los servidores de PgAdmin y PostgreSQL. A continuación, se asocia el servidor de PostgreSQL a PgAdmin para poder visibilizar la base de datos desde el gestor y para poder interactuar con esta. Por último, se crean las distintas tablas definidas en el modelo relacional (*Figura 3-14*).

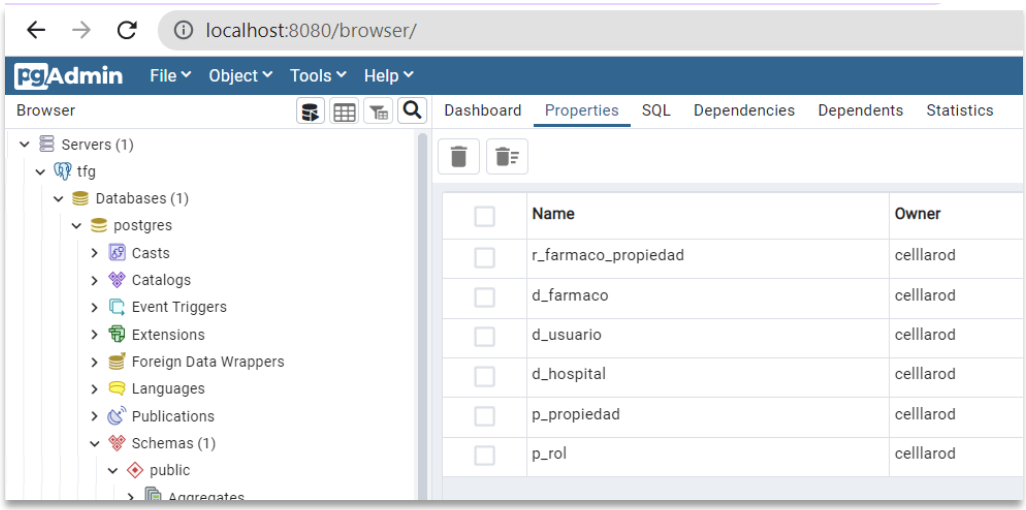


Figura 3-14. Servidores PgAdmin y PostgreSQL

3.3.2 Servicio API REST

Para la implementación del servicio API REST se ha empleado el framework de Spring con las siguientes especificaciones:

- Versión: 2.7.0.
- Lenguaje de programación Java (versión 17).
- Proyecto Maven.
- Dependencias:
  - *spring-boot-starter:* facilita la creación y la configuración de la aplicación.
  - *spring-boot-data-jpa*: permite usar la API de persistencia Java (JPA) y el motor de Hibernate para el almacenamiento de datos (SQL).
  - *spring-boot-devtools:* proporciona funcionalidades que hacen el desarrollo más ágil. Destaca la funcionalidad de reinicio automático.
  - *lombok:* permite automatizar el código que se suele repetir en cada una de las clases como los métodos getter y setter, los constructores y el patrón builder entre otros. Para ello, proporciona un conjunto de anotaciones entre las que destacan @Getter, @Setter, @NoArgsConstructor, @Builder, @Data.
  - *spring-boot-starter-test:* incluye los módulos de prueba de Spring Boot además de Junit, Mockito y otras librerías muy usadas en el ámbito del testing.
  - *spring-boot-starter-web:* incluye los módulos necesarios para implementar servicios web y el patrón MVC (Modelo-Vista-Controlador). Por defecto, proporciona Tomcat como contenedor de Servlets embebido.
  - *postgresql:* proporciona un controlador para la conexión con un sistema gestor de base de datos PostgreSQL.
  - *spring-boot-starter-security:* proporciona funcionalidades para la seguridad de la aplicación.
  - *jjwt:* incluye herramientas para manejar tokens JWT.

Con relación al entorno de desarrollo, se ha empleado el IDE de IntelliJ (versión de estudiante). Con esta

herramienta se han creado también los diagramas de clase que más adelante se incluirán con el fin de representar de manera gráfica el código de la aplicación.

El primer paso para la generación de la estructura principal del proyecto se ha realizado empleando la herramienta Spring Initializr. Esta provee una API que permite crear un proyecto indicando las especificaciones principales del mismo y sus dependencias.

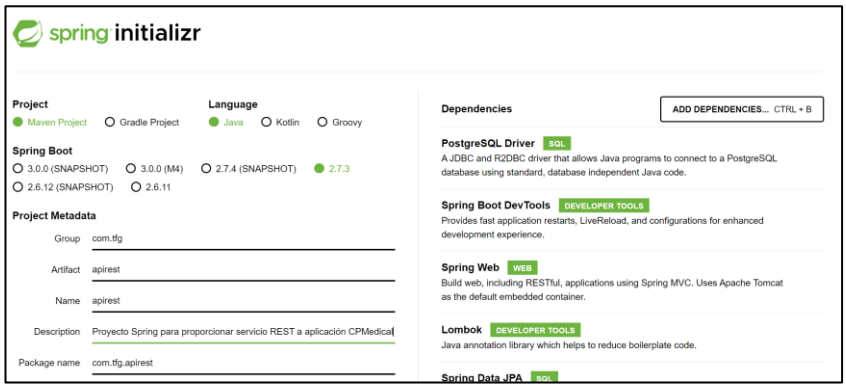


Figura 3-15. Spring Initializr

Para continuar, aunque la mayoría de la configuración de la aplicación se realizó de forma automática gracias a los diferentes "starters" de Spring, se procedió a completarla y especificarla de acuerdo con las especificaciones del sistema (*Figura 3-16*). Esta configuración se puede encontrar en el archivo *application.properties* y se ha incluido aspectos sobre el servidor, la interfaz JPA, la conexión con la base de datos y Spring Security.

application.properties

```
## Server Configuracion
server.port = 8081
server.error.include-message=always
server.error.include-binding-errors=always
server.error.include-stacktrace=never
server.error.include-exception=true

## JPA
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

## Conexión BBDD Postgres
spring.sql.init.mode=always
spring.sql.init.platform=postgres
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=celllarod
spring.datasource.password=admin

## Spring Security
spring.security.user.name=celllarod
spring.security.user.password=admin
# Clave con la que se firmará el token
cgmedical.app.jwtSecret= celllarodSecretKey
# Tiempo de expiración del token (30 días)
cgmedical.app.jwtExpirationMs= 2592000000
```

Figura 3-16. Configuración específica aplicación Spring

Con esta configuración ya se consigue conectar el gestor de base de datos a la aplicación como se puede observar en la *Figura 3-17.*

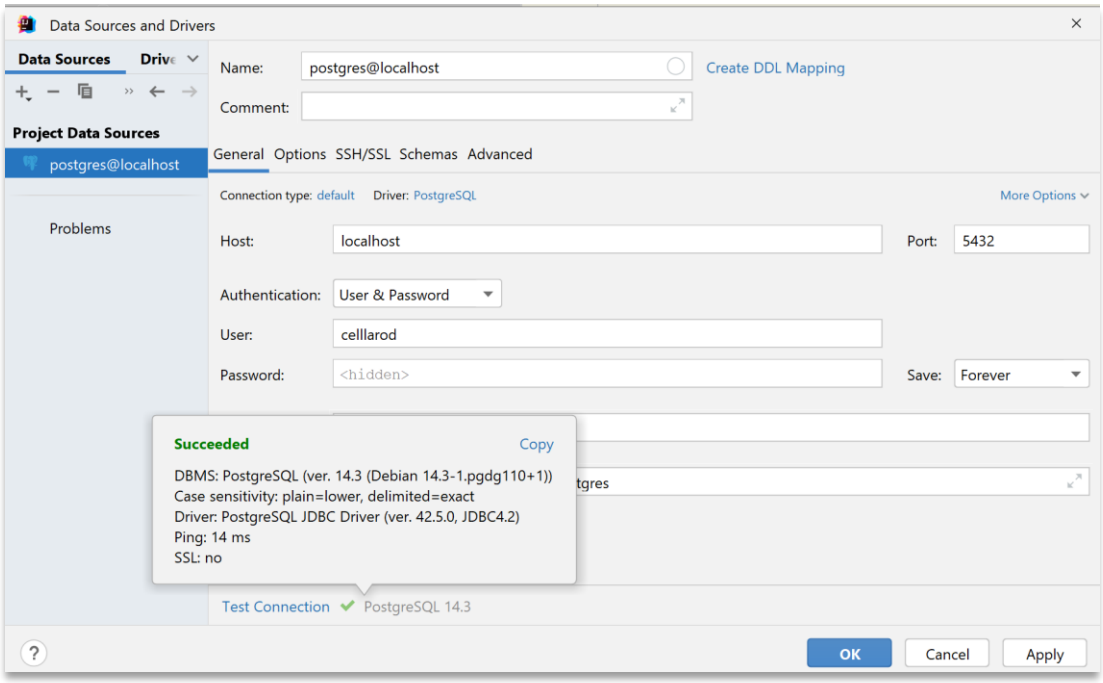
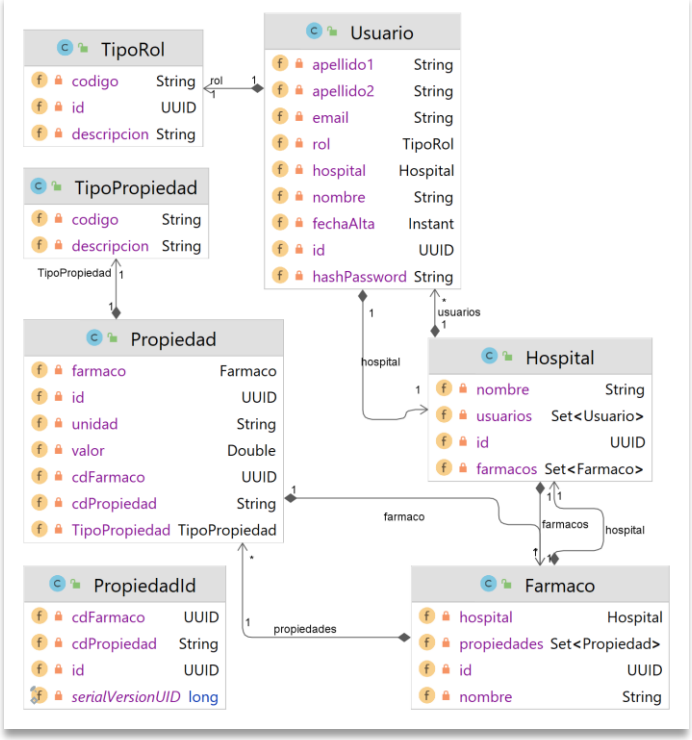


Figura 3-17. Test de conexión con la base de datos

En los siguientes subapartados se especificará cómo se ha ido desarrollando cada uno de los componentes de la aplicación.

3.3.2.1 Entidades

Una vez configurada la aplicación, se implementaron las clases de tipo "Entity" o POJO (Plain Old Java Object). Estas clases son las que permiten usar JPA para la interacción con la base de datos PostgreSQL ya que se encargan de mapear las tablas del modelo de datos a clases Java. Para realizar el mapeo se emplean anotaciones como @Entity que indica a JPA que se trata de una clase Entidad, @Id para indicar la clave primaria, @Column para la descripción de cada atributo, @Table para asociar la tabla y @OneToMany/@ManyToOne para detallar las asociaciones con otras tablas del modelo de datos.



Entity	Tabla en base de datos
Usuario	D_USUARIO
Propiedad	R_FARMACO_PROPIEDAD
Fármaco	D_FARMACO
Hospital	D_HOSPITAL
Tipo_Rol	P_ROL

Figura 3-18. Mapeo Entidad - Tabla en base de datos

Figura 3-19. Entidades de la Aplicación Spring

3.3.2.2 Repositorios

Se continuó con la implementación de las clases de tipo "Repository". Estas clases heredan de JPA y se encargan de definir una interfaz para acceder a los datos, es decir, implementan el patrón DAO (Data Access Object). Gracias a estas clases se hará el acceso a datos independiente del motor de base de datos que se esté empleando por lo que este se podrá cambiar en el futuro sin que se tenga que modificar nada (siempre que sea un motor compatible con JPA). Se usan las anotaciones @Repository y @EnableJpaRepositories para su configuración y será el framework quien se encargue de implementar estas interfaces.

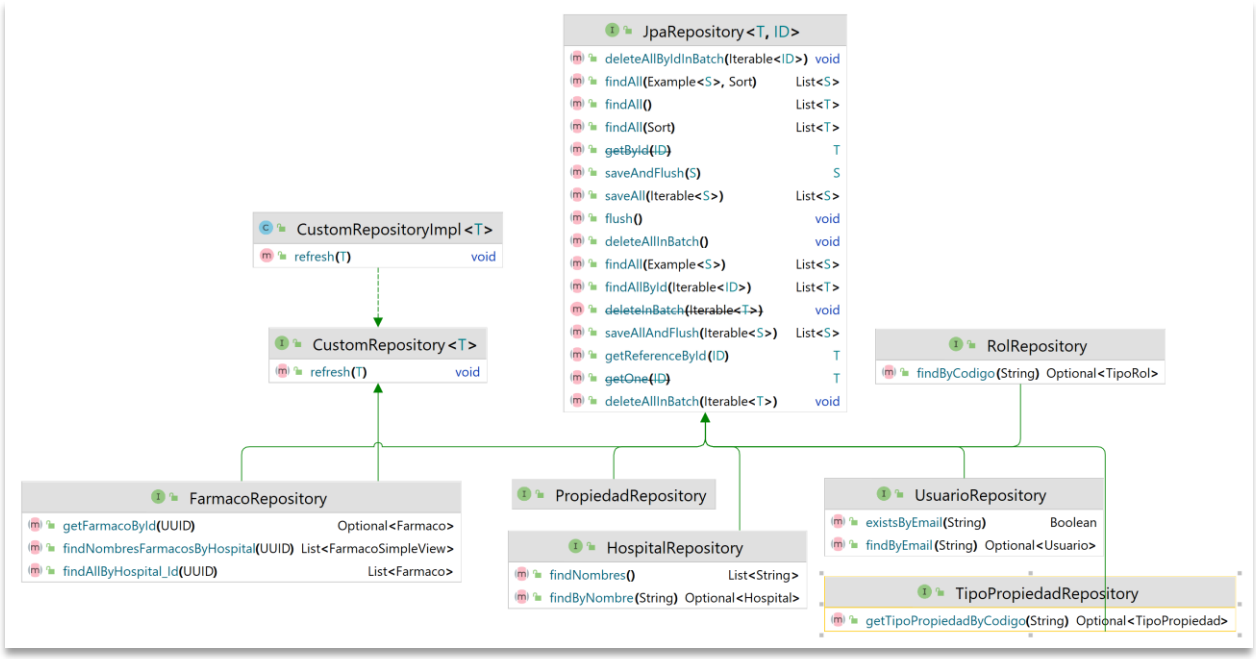


Figura 3-20. Repositorios de la Aplicación Spring

3.3.2.3 Servicios

Posteriormente, se definieron las clases "Service" que conectan los controladores y los repositorios e implementan la lógica de negocio. En el desarrollo, se han definido métodos públicos que serán llamados por los controladores y métodos privados que serán empleados desde la propia clase para el uso de los repositorios. De esta manera, se controla y aísla el acceso a los datos. Por otra parte, las anotaciones necesarias para identificar este tipo de clases serán @Service y, en caso de que fuera necesario validar algún parámetro de entrada (a nivel de negocio), @Validated.

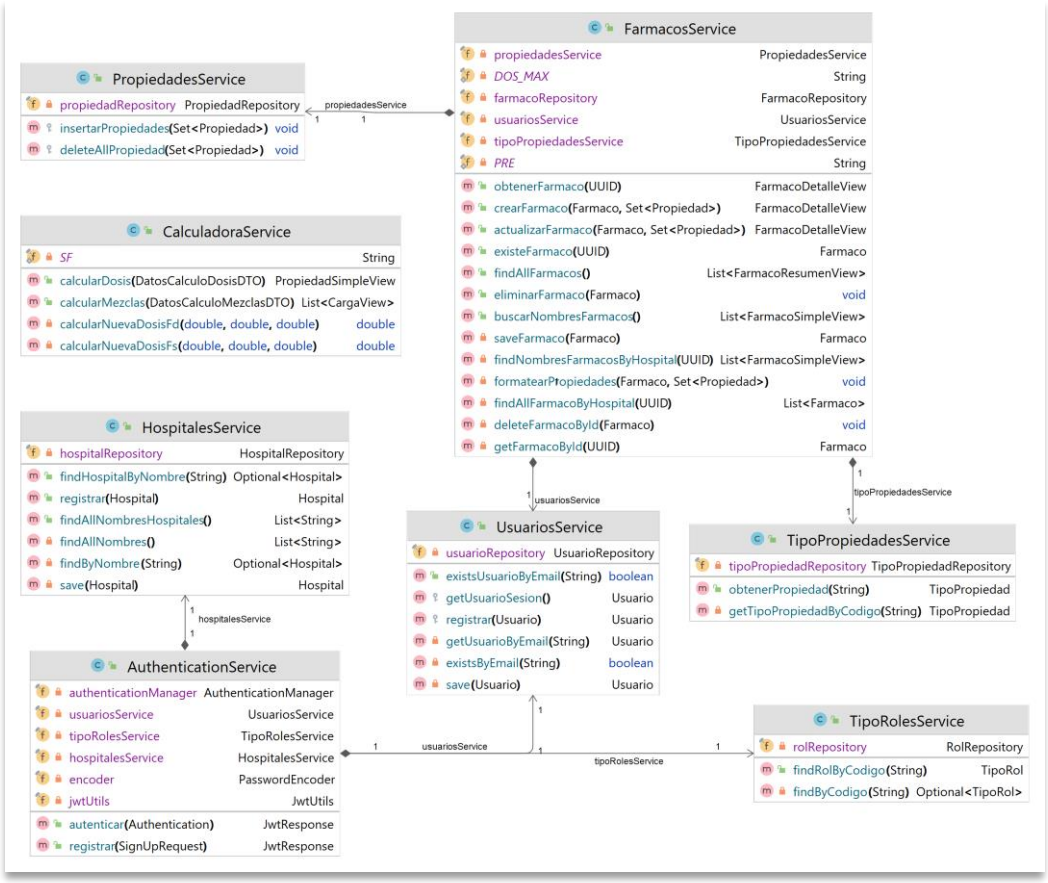


Figura 3-21. Servicios de la Aplicación Spring

3.3.2.4 Controladores

Otro de los componentes implementados es el "Controller". Estas clases son las que reciben cada una de las peticiones que llegan al servidor. En primer lugar, habrá que añadir la anotación @RestController para que siga el estereotipo correspondiente.

Además de lo anterior, habrá que mapear sus métodos para que puedan ser llamados cuando se quiera atender una determinada petición HTTP con un método específico y a una URL concreta. Para ello, se usan anotaciones como: @RequestMapping, @PutMapping, @GetMApping, @DeleteMapping, @PostMapping.

Las peticiones HTTP pueden contener variables tanto en la URL como en el cuerpo. A través de las anotaciones @PathVAriable y @RequestBody podremos especificar las variables de cada método. Cabe destacar que todos los parámetros de entrada son validados a nivel de formato gracias a anotaciones como @Validated, @Valid, @NotNull, @Size y @Pattern entre otras muchas.

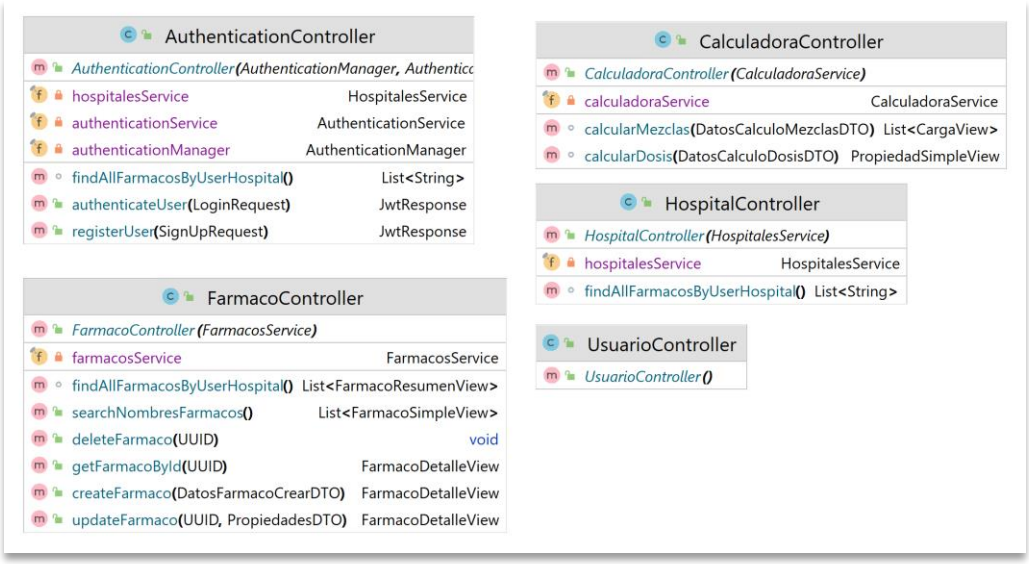


Figura 3-22. Controladores de la Aplicación Spring

3.3.2.5 Vistas

En cuanto a las vistas, corresponden a los objetos que modelan los JSON intercambiados entre el cliente y el servidor REST. Se ha realizado una distinción entre los objetos de entrada, que se han nombrado como "DTO" y los de salida, "View".

Las anotaciones empleadas en el caso de los "DTOs" son @JsonProperty, para mapear cada propiedad del JSON de entrada al atributo que lo representa; y otras para realizar validaciones de formato (@Valid, @NotNull…). En contraposición, las clases de tipo "View" se ha usado @JsonPropertyOrder para indicar el orden que deberán tener las diferentes propiedades en JSON de salida.

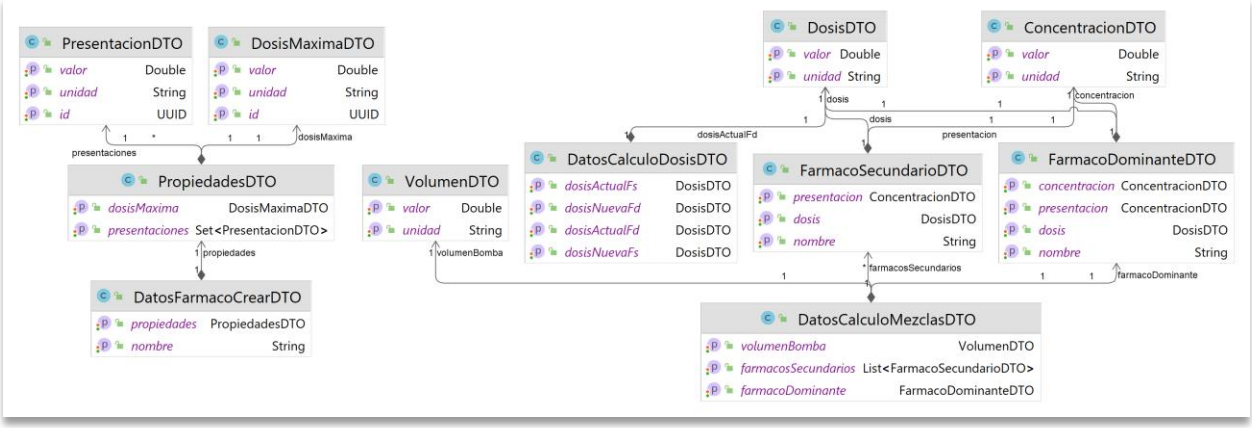


Figura 3-23. Objetos de entrada al servicio REST (DTO)

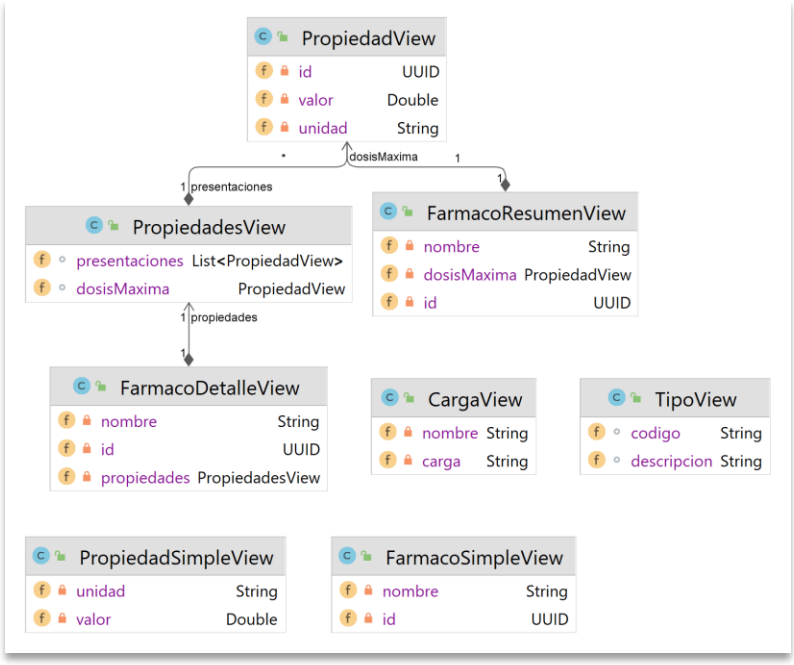


Figura 3-24. Objetos de salida del servicio REST (Vistas)

3.3.2.6 Seguridad

Desde el punto de vista de la seguridad del servicio REST, como ya se ha mencionado, se han hecho uso de las herramientas que ofrece Spring Security y JWT. En la Figura 3-25 se pueden ver las clases implicadas.

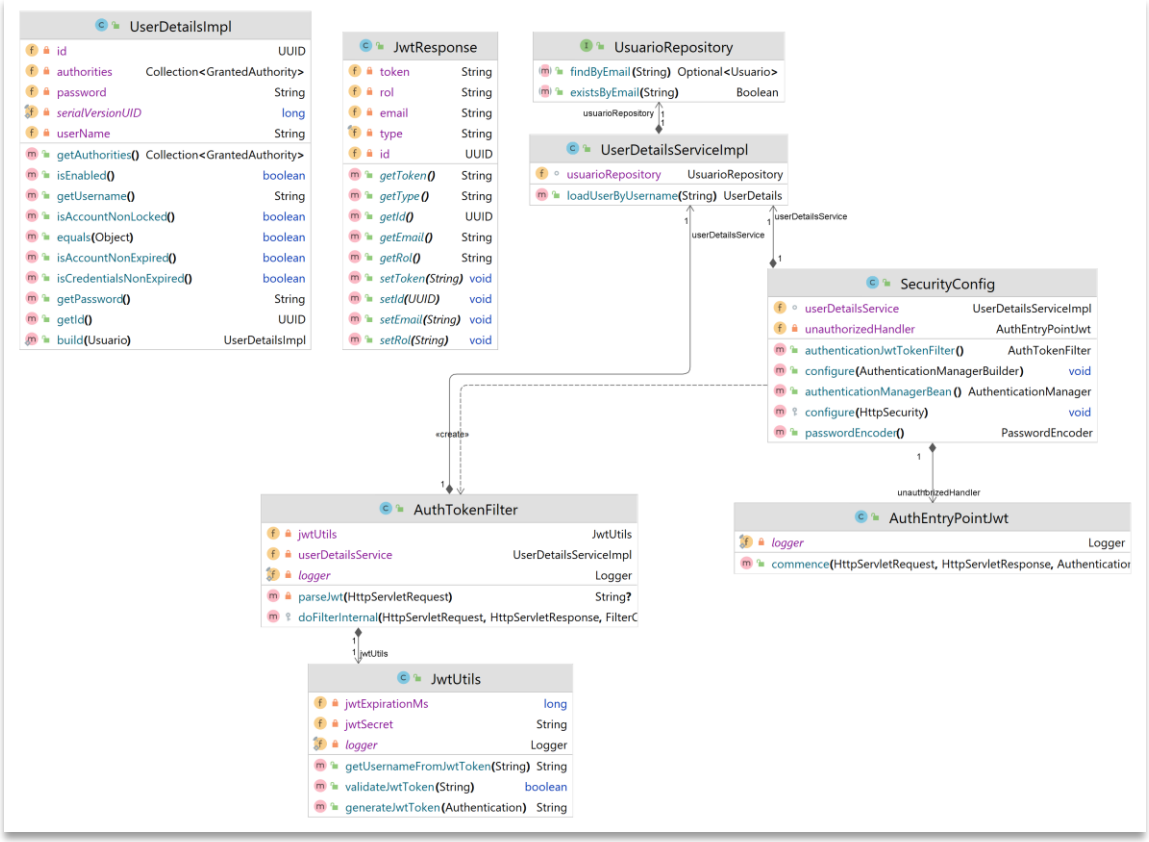


Figura 3-25. Clases que permiten la seguridad del servicio

Para el proceso de autenticación, en *SecurityConfig* se configura que todas las peticiones HTTP entrantes salvo

aquellas con destino *"/api/v1/auth/**"* tienen que pasar por el filtro *AuthTokenFilter* **antes de llegar al controlador correspondiente**. En este filtro se procede a autenticar al usuario y ver si dispone de los permisos necesarios (authorities) para acceder al recurso que está solicitando.

Todas las peticiones deberán llegar con un token en la cabecera "Authentication". A partir de ese token el sistema es capaz de extraer qué usuario está realizando la petición y qué permiso tiene (*UserDetailsServiceImpl*, *UserDetailsService*). Además, deberá comprobar que la firma del token es válida y que este no haya expirado (*JwtUtils*).

Por otra parte, para el inicio de sesión y el registro en el sistema no se pasará por el filtro de autenticación (ya que así se ha configurado en *SecurityConfig*). En el inicio de sesión se autentica al usuario mediante *AuthenticationManager*, clase de Spring Security que se ha configurado también en *SecurityConfig*. Con relación al registro de usuarios, se procederá al insertar al mismo en el sistema. La respuesta en ambos casos será el token JWT que deberá ser usado para futuras peticiones. En la Figura 3-26, se pueden observar las clases implicadas en este escenario.

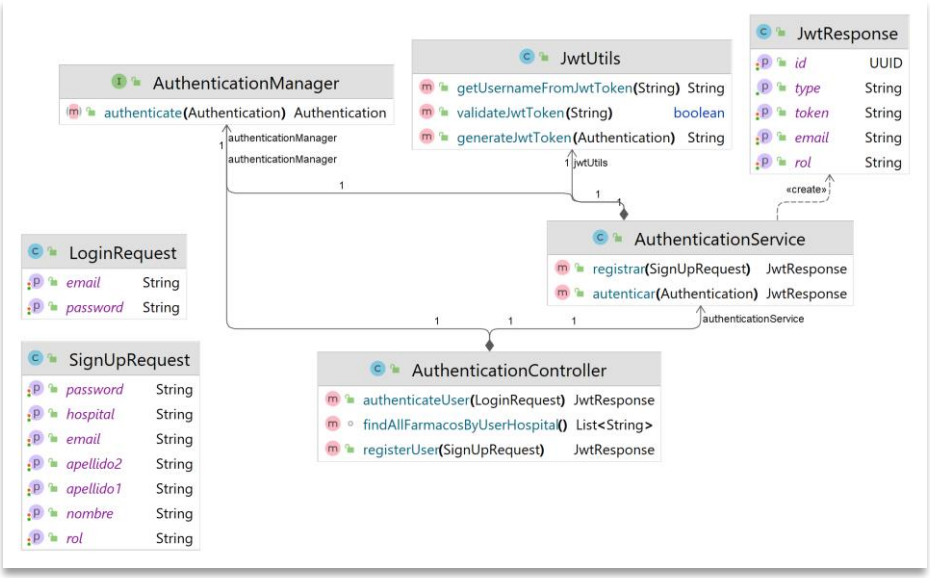


Figura 3-26. Clases implicadas en inicio de sesión y registro de nuevo usuario

3.3.2.7 Otros componentes

Además de todo lo anterior, cabe destacar el uso de la clase GlobalExceptionHandler cuyo objetivo es manejar todas las posibles excepciones que se produzcan en el servicio REST y controlarlas creando una respuesta HTTP con el mensaje y el código de error correspondiente.

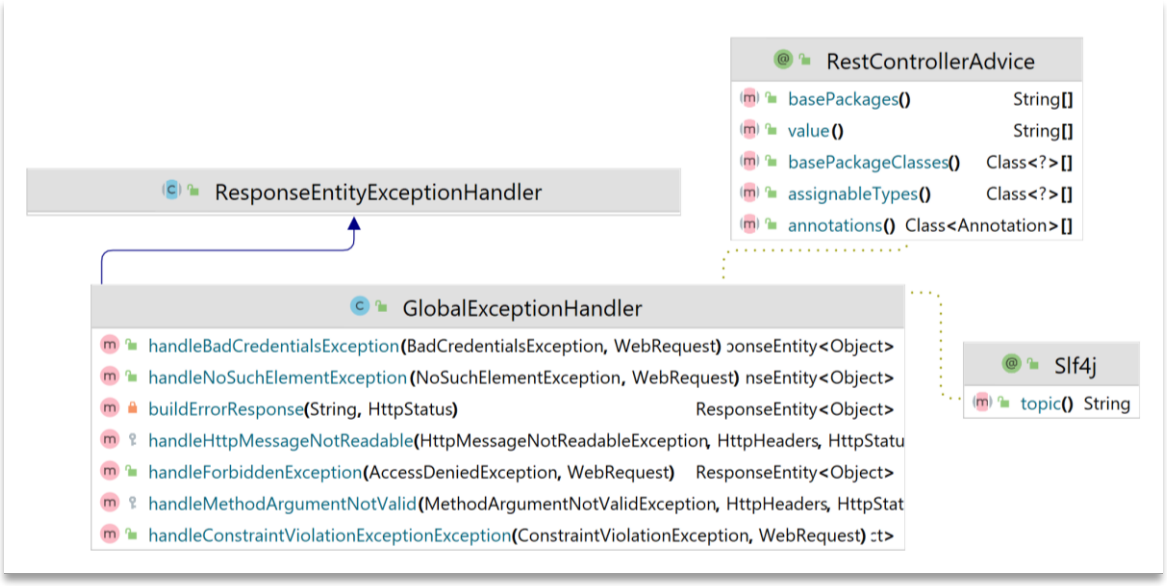
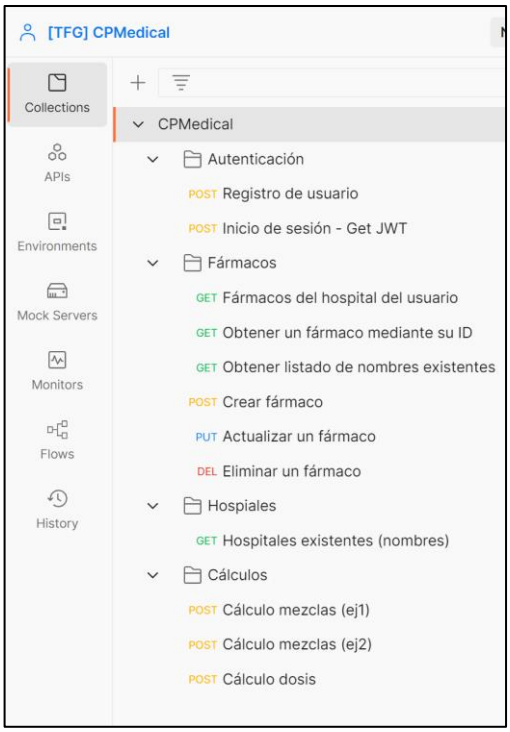


Figura 3-27. Manejador de excepciones global

3.3.2.8 Pruebas

A medida que se han ido implementando las consultas que forman el servicio REST se ha ido probando cada una de ellas a través del cliente REST Postman. Es muy importante que esto se haga para asegurar que cada petición realiza o devuelve lo esperado. De este modo, se ha creado una colección en este cliente y se han definido todas las peticiones CRUD que se pueden realizar (*Figura 3-29*).

Con relación a lo anterior, en la *Figura 3-28* se incluye un ejemplo de la petición que permite la autenticación en el servicio REST y la obtención del token JWT. Dado que para acceder al resto de recursos de la API será necesario incluir este token, al obtenerlo se ha hecho que se ejecute una función que lo guarda en una variable de la colección y permite para que se pueda acceder a él dentro de la misma (por comodidad).



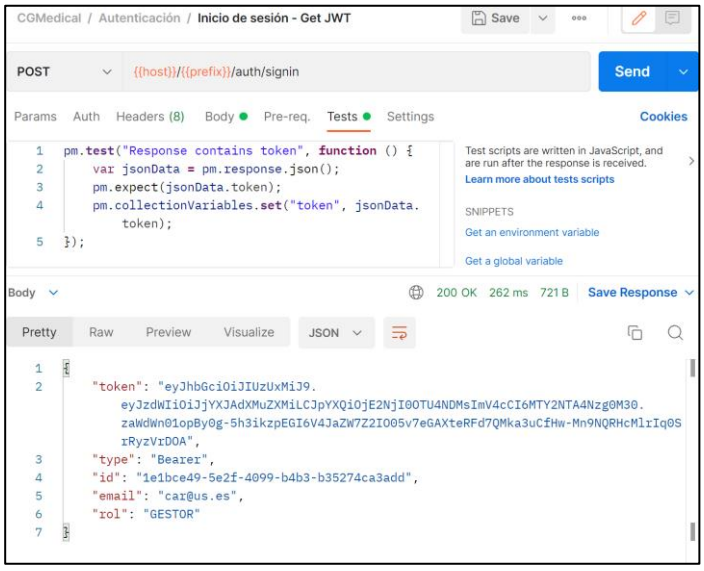


Figura 3-28. Ejemplo petición REST desde Postman

Figura 3-29. Colección Postman (Cliente REST)

3.3.3 Aplicación móvil

En este apartado se va a pasar a hablar de la implementación del componente de la aplicación móvil que tendrá las siguientes especificaciones principales:

- Lenguaje de programación: Java.
- Sistema Operativo: Android.
- Versión SDK de compilación: 32.
- Mínima versión de SDK: 27.
- Dependencias:

```
implementation 'androidx.appcompat:appcompat:1.4.1'
implementation 'androidx.legacy:legacy-support-v4:1.0.0'
// Ciclo de vida
implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.4.1'
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
// Navegacion
implementation 'androidx.navigation:navigation-fragment:2.4.2'
implementation 'androidx.navigation:navigation-ui:2.4.2'
implementation 'androidx.annotation:annotation:1.3.0'
// Constraint Layout
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
// Mateial
implementation 'com.google.android.material:material:1.6.0'
// Testing
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
// Retrofit
implementation 'com.squareup.retrofit2:retrofit:2.3.0'
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
// Logs HTTP
implementation 'com.squareup.okhttp3:logging-interceptor:3.9.1'
// RecyclerView
implementation 'androidx.recyclerview:recyclerview:1.2.1'
// CardView
implementation 'androidx.cardview:cardview:1.0.0'
// Vista imagen redonda
implementation 'com.makeramen:roundedimageview:2.3.0'
// Flex
implementation 'com.google.android.flexbox:flexbox:3.0.0'
// SwipeRefreshLayout
implementation 'androidx.swiperefreshlayout:swiperefreshlayout:1.1.0'
```

Es importante indicar que el diseño de la interfaz de usuario se ha diseñado para que se visualice en pantallas de tamaño expandido, por lo que se garantizará la correcta visualización de estas en el 94.25% de las tabletas en orientación vertical, según la documentación de Android. No obstante, se ha procurado que los componentes sean compatibles para todas las pantallas mediante diferentes técnicas como el uso de ConstraintLayout, Flex o vistas tipo Scroll. De este modo, la mayoría de los diseños existentes son responsivos, pero en algunos podemos encontrar fallas.

Con relación al entorno de desarrollo, se ha empleado el propio IDE de Android Studio.

Para explicar cómo se han implementado cada uno de los componentes de la aplicación y no perder el hilo conductor, en los siguientes subapartados se van a agrupar los diferentes elementos por funcionalidad. Se añadirá una imagen que permita visualizar el diseño y las interacciones entre pantallas para ofrecer los distintos contenidos.

3.3.3.1 Inicio de sesión y registro en el sistema

En la secuencia de pantallas de la Figura 3-30, se pueden apreciar dos flujos principales. Por una parte, se representa el inicio de sesión en el que el usuario tendrá que añadir sus credenciales correctamente para poder acceder al sistema. Intervienen las siguientes actividades y layouts.

- **Pantalla de bienvenida:**
  - **InicioActivity:** los botones tienen un listener (onClick) que hace que se inicie la actividad *LoginActivity* o *SignUpActivity*.

- **Layout activity_inicio:** se trata de un ConstraintLayout.

- **Pantalla de iniciar sesión:**
  - **LoginActivity:** el botón de enviar tiene un listener (onClick) que hace que cuando se pulse, se invoque al método "signinUser" de la interfaz *APIService* (adaptada por Retrofit). Esto desemboca en una llamada asíncrona al servicio REST para autenticar al usuario y obtener un token JWT que permita mantener la sesión iniciada. Si la respuesta que se recibe es exitosa, mediante *SessionManager* se guarda la información del usuario y su token en *SharedPreferences* para que el usuario no tenga que volver a iniciar sesión la próxima vez que use la aplicación. Finalmente, inicia *MainActivity*.
  - **Layout activity_login:** está formado por un ConstraintLayout que contiene un CardView con el formulario de inicio de sesión (TextInputLayout). Para evitar llamadas al servicio REST innecesarias, se valida a nivel de formato todo lo que el usuario inserte en los campos editables.

- **Pantalla de crear cuenta:**
  - **SignUpActivity:** el botón de siguiente tiene un listener (onClick) que hace que cuando se pulse se validen a nivel de formato todos los campos editables del formulario. Si estos son correctos se inicia *HospitalActivity* pasando la información de usuario como extra del Intent que permite iniciar la nueva actividad (ver Pantalla selección hospital).

    Layout activity_sign_up: se trata de un ConstraintLayout que engloba un CardView con scroll. A su vez, contiene el formulario registro de un nuevo usuario.

- **Pantalla selección de hospital:**
  - **HospitalActivity:** en primer lugar, realiza una llamada al servicio REST *APIService.getHospitales* para obtener el listado de hospitales existentes e inicializar el elemento seleccionable usando un ArrayAdapter. A su vez, el botón de continuar tiene un listener (onClick) que, tras validar si se ha elegido algún hospital, realiza una petición HTTP REST llamando a *APIService.signUpUser*. Si la respuesta es exitosa, se realiza lo mismo que en el caso de iniciar sesión.
  - **Layout activity_hospital:** se trata de un GridLayout que engloba un CardView con un TextInputLayout del tipo AppCompatAutoCompleteTextView que permite tanto seleccionar los hospitales con los que se cargue, como escribir un nuevo hospital.

→ **MainActivity:** se inicia siempre que el usuario entre a la aplicación. Es por ello por lo que aquí se comprueba si existe alguna sesión guardada en SharedPreferences con el token del usuario. Si es así y el token no ha expirado, inicializa y muestra el NavigationView que posibilita la existencia de un menú lateral para moverse cómodamente entre las diferentes secciones. En caso de que no exista ninguna sesión guardada con anterioridad, se inicia *InicioActivity* (pantalla de bienvenida).

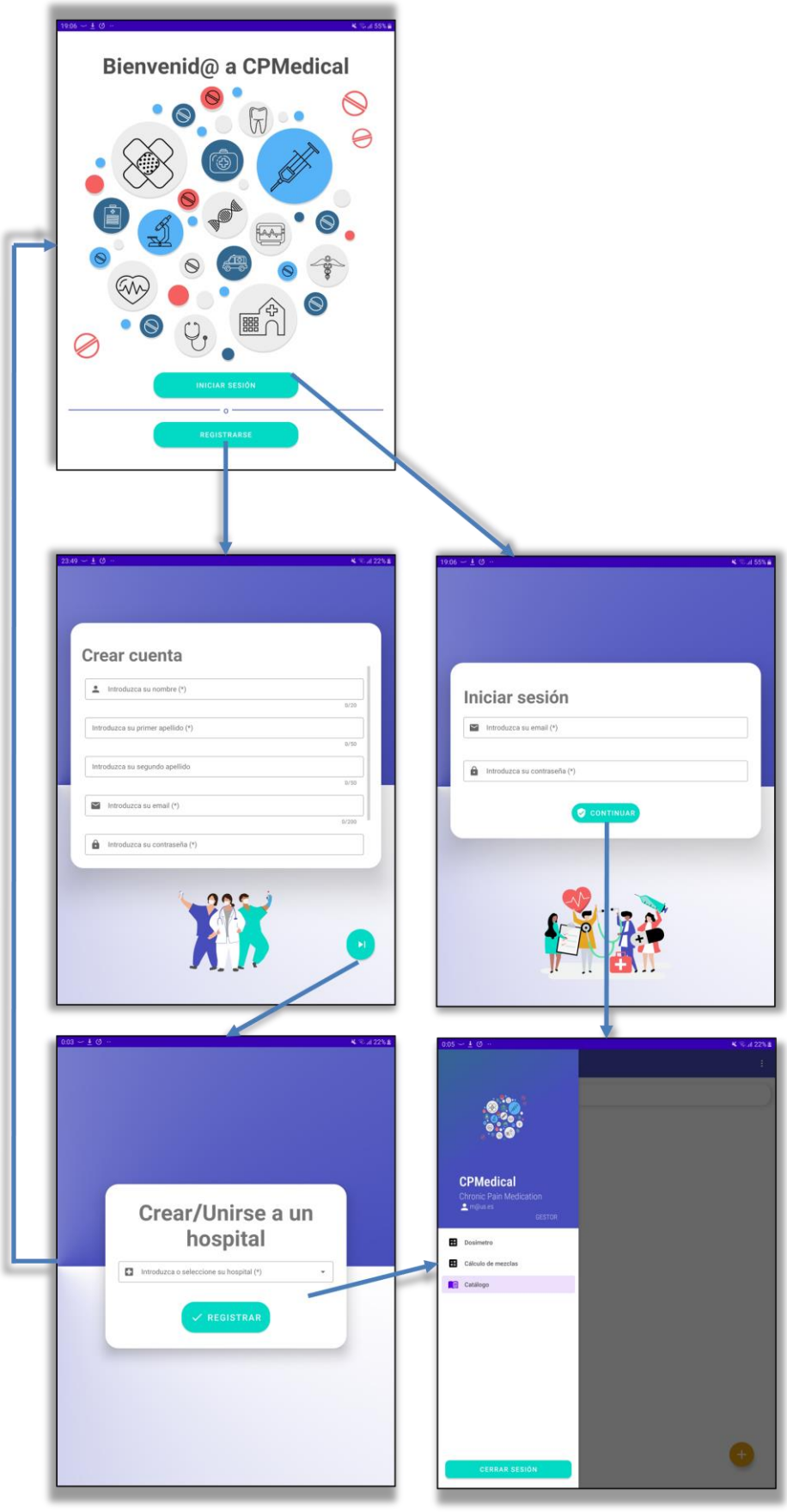


Figura 3-30. Flujo de pantallas de Inicio de sesión y registro de usuarios

3.3.3.2 Sección catálogo de fármacos

A esta sección se accede tras seleccionar "Catálogo fármacos" en el menú lateral. En la Figura 3-31 se puede observar el flujo de actividades (y fragmentos) relacionado con esta sección.

- **Pantalla de bienvenida:**
  - **CatalogoFragment:** en primer lugar, comprueba el rol del usuario actual. Únicamente si es GESTOR, mostrará el botón que permite añadir nuevos fármacos cuando se pulsa (se inicia *RegistrarFarmacoActivity* en dicho caso).

    A su vez, inicializa a través de su ViewModel asociado y del adaptador *ListaFarmacosAdapter* (el ViewModel realiza petición HTTP REST *ApiService.getFarmacos(jwt)* para obtener el listado de fármacos). Cabe añadir que este adaptador asociará un listener (onClick) a cada uno de los fármacos de la lista de modo que, si se pulsa sobre uno de ellos, se inicie *DetalleActivity* indicando como extra del Intent que permite iniciar la actividad el ID del fármaco seleccionado.

    También inicializa el buscador con un listener (onQueryTextListener) que hará que cuando se busque algo se filtre la lista de fármacos según la búsqueda haciendo uso del adapter.

    Finalmente, permite refrescar el listado de fármacos deslizando hacia arriba, lo que supondrá una nueva llamada al servicio REST).
  - **Layout fragment_catalogo:** se trata de un ConstraintLayout que incluye un SearchView y SwipeRefreshLayout que contendrá el listado de fármacos y permita deslizar hacia abajo para refrescar el listado.
- **Pantalla detalle:**
  - **DetalleActivity:** en primer lugar, comprueba el rol del usuario actual. Únicamente si es GESTOR, mostrará los botones que permiten editar y eliminar el fármaco cuando se pulsan (se muestra BottomSheetDialog si se pulsa editar o se realiza petición HTTP REST *ApiServide.deleteFarmaco(id, jwt)* si se pulsa editar).

    Además, inicializa los componentes de la pantalla para poder visualizar la información detallada del fármaco (previamente habrá tenido que hacer petición HTTP REST *ApiService.getFarmaco(id,jwt)*
  - **Layout activity_detalle:** se trata de un ConstraintLayout con un CardView y un ScrollView y los campos para establecer la información detallada del fármaco.
- **Dialogo editar:** se abre cuando se pulsa el botón de editar. Valida los datos que se modifican a nivel de formato y el de la dosis a nivel de negocio, no puede superar el valor de dosis máxima asociado al fármaco que se está visualizando.
- **Pantalla registrar fármaco:**
  - **RegistrarFarmaco:** inicializa el layout y asocia un listener al botón de añadir presentación que creará una vista nueva para poder insertar una presentación nueva. También le asocia un listener al botón de aceptar que hará una Petición HTTP REST *AiService.updateFarmaco(id,jwt)* únicamente si cuando valida el formulario no se encuentra ningún error. Finalmente, añadirá un listener al botón de cancelar para volver al catálogo de fármacos si se pulsa.
  - **Layout activity_registrar_farmaco:** se trata de un ConstraintLayout con un CardView y un ScrollView que incluye el formulario para registrar un nuevo fármaco (*formulario.xml*).

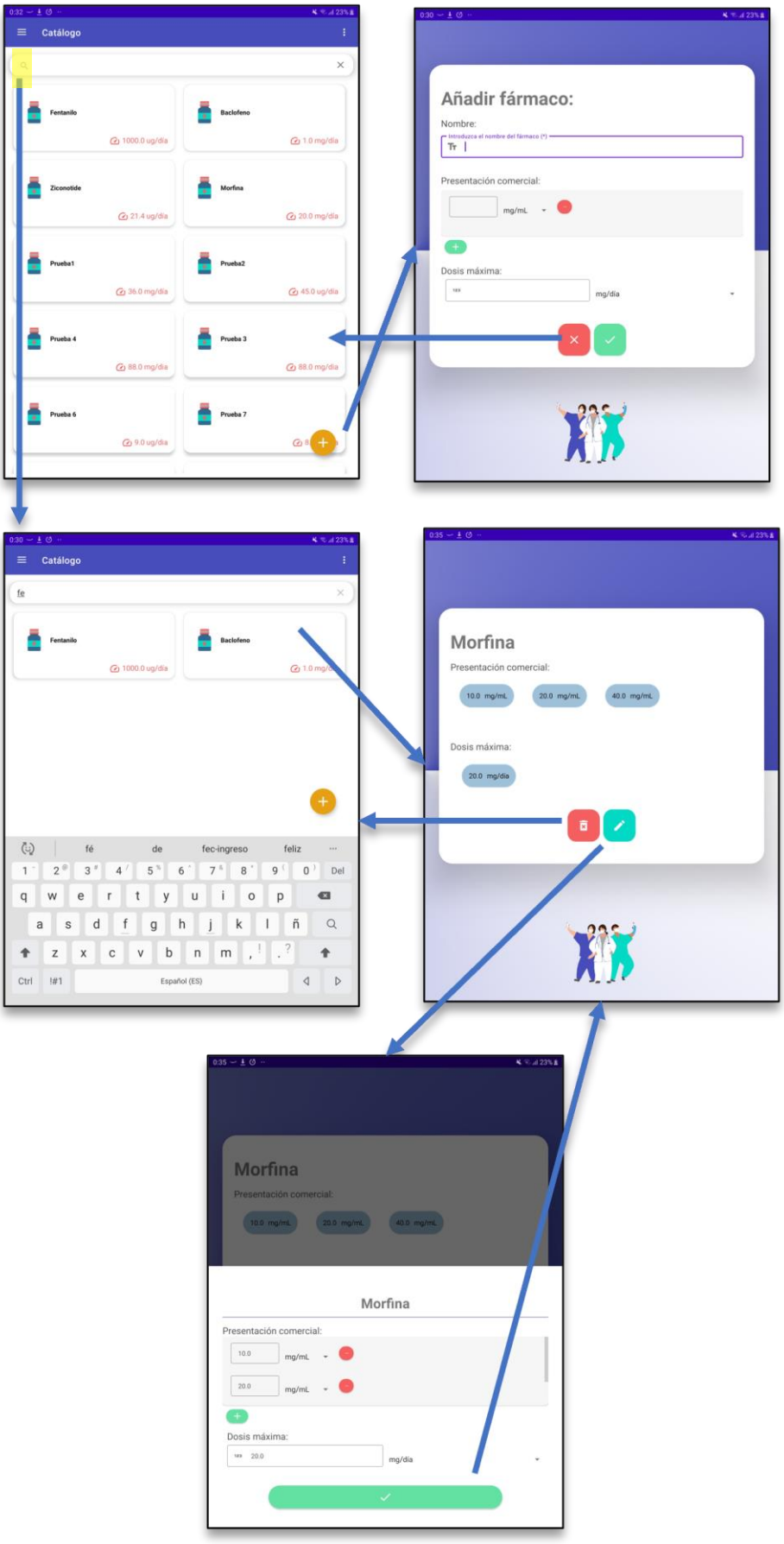


Figura 3-31. Flujo de pantallas catálogo fármacos

3.3.3.3 Sección cálculo de mezclas

A esta sección se accede tras seleccionar "Cálculo mezclas" en el menú lateral. En la *Figura 3-32* se puede observar el flujo fragmentos relacionado con este apartado.

- **Pantalla formulario paso a paso:**
  - **MezclasFragment:** se encargará de mostrar un fragmento u otro en función de los botones seleccionados, consiguiendo así un formulario en 3 pasos. Para ello, hará uso de FragmentManager.
  - **Layout fragment_mezclas:** se trata de un ConstraintLayout con un CardView y un FrameLayout en el que se irán añadiendo y reemplazando los diferentes fragmentos según el paso seleccionado. Además, incluye una serie de botones que permitirán navegar de un paso del formulario a otro.

✓ **Paso 1: elegir volumen de la bomba**

- **VolumenFragment:** manejará los eventos de selección de un volumen.
- **Layout fragment_step_volumen:** se trata de un ConstraintLayout con un RadioGroup que incluirá los botones para seleccionar el volumen de la bomba.

✓ **Paso 2: elegir fármaco dominante**

- **FarmacoDominanteFragment:** al iniciarse, realiza, a través de su ViewModel asociado, una llamada al servicio REST mediante *APIService.getNombresFarmacos* para obtener el listado de fármacos que inicializará a través de un ArrayAdapter el desplegable para seleccionar un fármaco u otro. Además, este elemento tendrá un listener (onClick) que permitirá incorporar una validación al introducir un valor de dosis, de modo que se impida indicar una dosis mayor a la máxima del fármaco seleccionado. A su vez, se inicializará el desplegable que permite seleccionar la presentación comercial con el listado de presentaciones asociadas al fármaco en el que haya hecho clic.
- **Layout fragment_step_farmaco_dominante:** se trata de un ConstraintLayout que contiene el formulario para añadir el fármaco dominante que actúa como dato de entrada para el cálculo de la mezcla de una bomba de infusión.

✓ **Paso 3: elegir listado de fármacos secundarios**

- **FarmacosSecundariosFragment:** dado que indicar un fármaco secundario es un paso opcional, se muestra un botón con un listener (onClick) que hace que cuando se pulse, se muestre un diálogo con el formulario para añadir un fármaco secundario. El funcionamiento de los diferentes elementos del diálogo es muy similar al explicado para el formulario en el que se introduce el fármaco dominante. Se podrán añadir (y eliminar) tantos fármacos secundarios como se desee y se mostrarán en una lista.
- **Layout fragment_step_farmacos_secundarios:** se trata de un ConstraintLayout que incluye un botón para añadir un nuevo fármaco secundario y de un RecyclerView que manejará el listado de fármacos secundarios añadidos. Este último elemento se incluye dentro de un NestedScrollView para que el listado tenga la opción de scroll.

✓ **Resultado: receta**

- **RecetaFragment:** validará que todos los pasos anteriores se hayan completado sin errores y, en ese caso, realizará, a través de su ViewModel asociado, una llamada al servicio REST mediante *APIService.calcularMezclas* para obtener el resultado final. Si no hay ningún error, mostrará dicho resultado, en caso contrario, mostrará un mensaje de error y cambiará el color de los botones que permiten navegar entre un paso del formulario y otro a rojo o verde según si se ha detectado o no algún error en dichos pasos.
- **Layout fragment_step_receta:** se trata de un ConstraintLayout que incluye un RecyclerView (con opción de scroll gracias a NestedScrollView) en el que se incluirá el listado de fármacos que deberán incluirse en la bomba y la carga o volumen de estos (resultado del cálculo de mezclas).

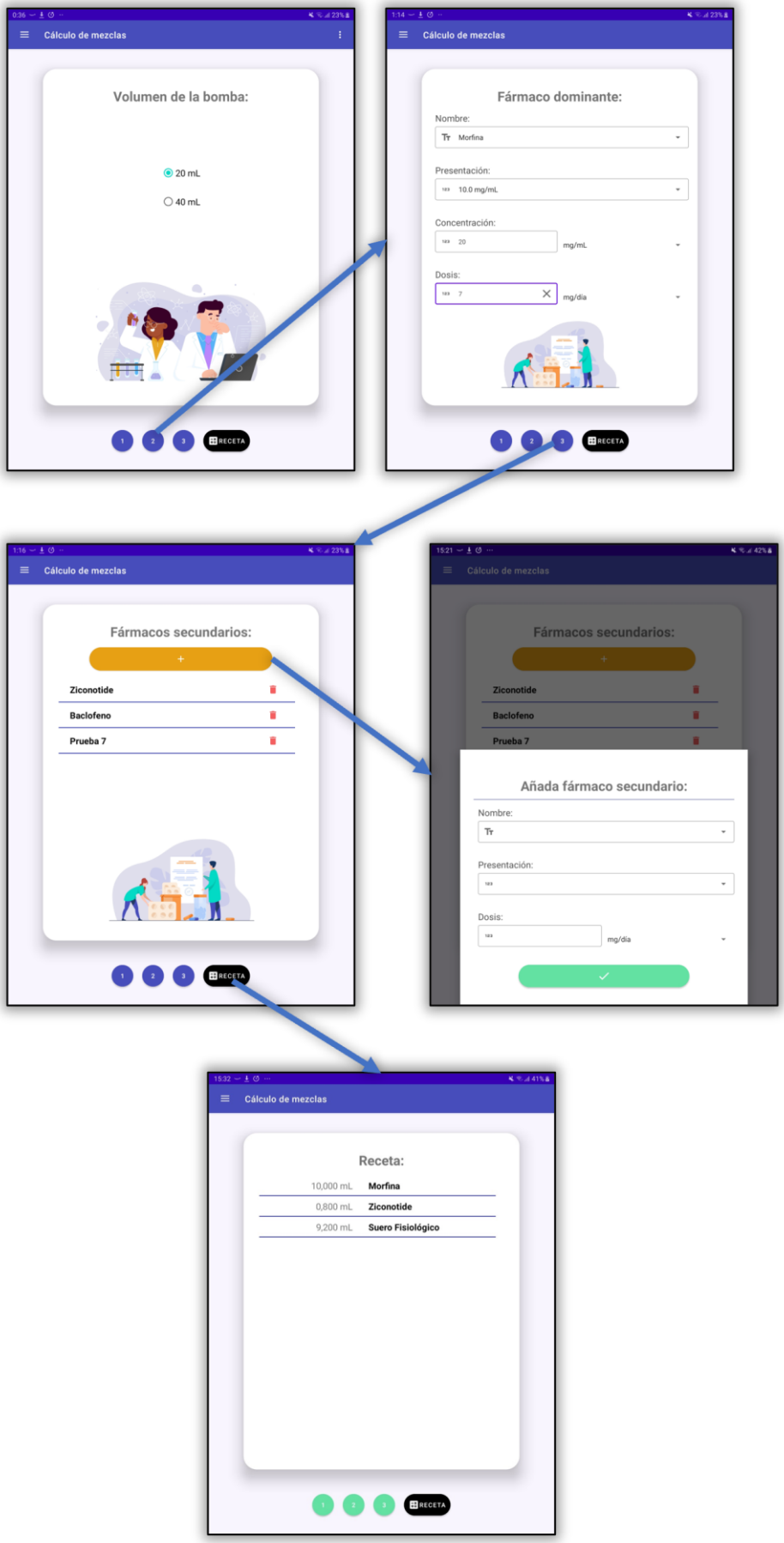


Figura 3-32. Flujo pantallas cálculo de mezclas

3.3.3.4 Sección cálculo de dosis

A esta sección se accede tras seleccionar "Dosímetro" en el menú lateral. En la *Figura 3-33* se puede observar el flujo fragmentos relacionado con este apartado.

- **Pantalla dosímetro**
  - **DosimetroFragment:** en primer lugar, solo estarán habilitados los campos para seleccionar los nombres de los fármacos. Para inicializar estos desplegables se realizará, a través de su ViewModel asociado, una llamada al servicio REST mediante *APIService.getNombresFarmacos*. Estos elementos tendrán un listener (onClick) que inicializará la validación de los campos para rellenar las dosis con el valor máximo asociado al fármaco seleccionado. En este momento, se habilitarán el resto de los campos de dosis a excepción del campo de nueva dosis del fármaco que se quiera calcular ya que será ahí donde se mostrará el resultado final.

    El botón "calcular" tendrá un listener (onClick) que hará que se validen todos los campos introducidos y que se realice una llamada al servicio REST mediante *APIService.calcularDosis* para obtener el resultado final.
  - **Layout fragment_dosimetro:** se trata de un ConstraintLayout con un CardView que contendrá dos FlexBoxLayout que permitirán redistribuir los formularios del fármaco dominante y del fármaco secundario en función del tamaño de pantalla disponible. Además, incluye botones para reiniciar el formulario, calcular el resultado final y elegir el fármaco (dominante o secundario) al que calcular la nueva dosis requerida.

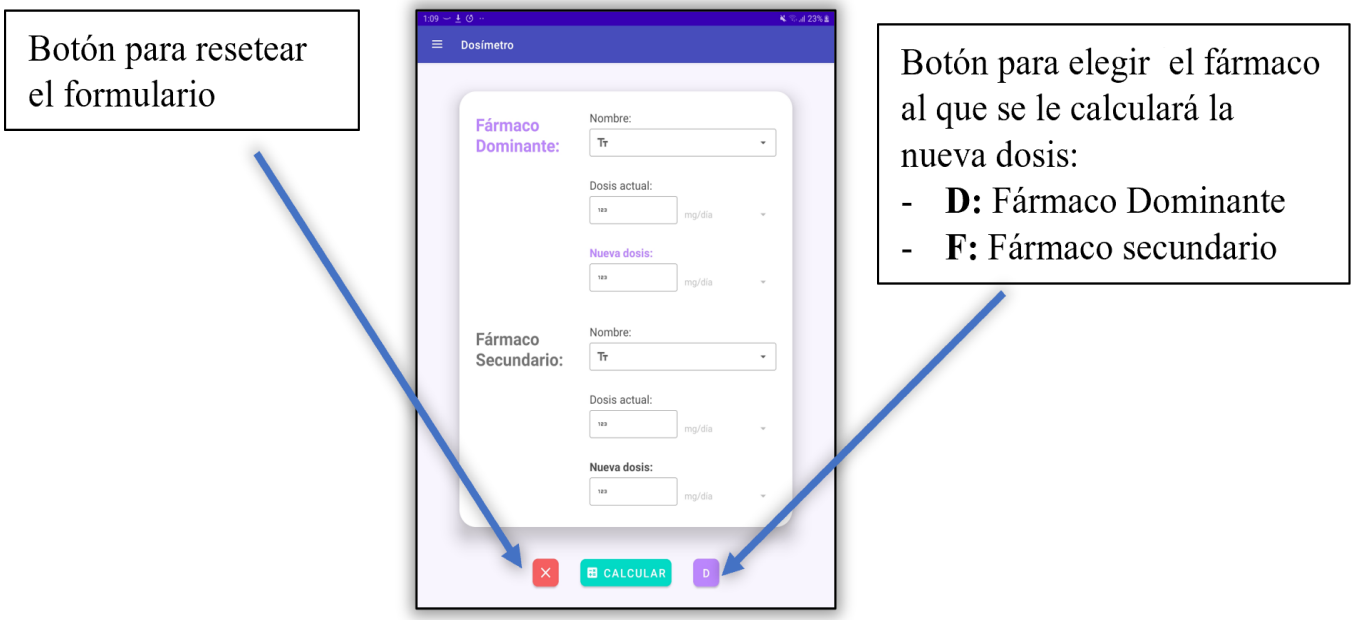


Figura 3-33. Pantalla de dosímetro

3.3.3.5 Refrofit

Para poder interactuar con el servicio REST desde la aplicación de Android, será necesario emular un cliente REST que permita realizar las diferentes peticiones HTTP. Para ello, se ha empleado la librería de Retrofit que ha permitido realizar las llamadas de red, obtener los resultados y parsearlos de forma automática a un objeto Java manejable desde el código.

La implementación ha consistido en crear los siguientes elementos:

- **APIAdapter:** se encarga de crear una instancia única de Retrofit que se configura indicándole la URL base del servidor REST, el convertidor que deberá usar para convertir las respuestas a objetos (GsonConverterFactory), el cliente HTTP que realizará las llamadas de red y recibirá las respuestas (OkHttpClient) y la API a usar (APIService) para montar cada una de las solicitudes REST.

  Cabe destacar que cada instancia de Retrofit representa una conexión con el servidor por lo que conviene que sea única. De este modo, se ha construido la clase APIAdapter siguiendo el patrón Singleton para que solo cree esta instancia la primera vez.

- **APIService:** define una interfaz Java que será adaptada por Retrofit para realizar las llamadas HTTP. Cada método de la interfaz representará una llamada asíncrona a un endpoint del servidor REST y especifica a Retrofit cómo debe realizar las peticiones mediante etiquetas (@GET(/endpoint), @Body, @Path, @Header …).

Para realizar la llamada a un endpoint desde alguna parte del código, se deberá:

1. Obtener la interfaz APIService asociada a la instancia de Retrofit mediante el método *ApiAdapter.getApiService.*
2. Llamar al método de la interfaz de APIService que represente al endpoint deseado. Este método devolverá un objeto de tipo *Call* que se encargará de enviar de enviar la petición HTTP cuando se llame a su método *enqueue* de forma asíncrona.
3. Llamar al método *enqueue* del objeto anterior para poner la petición en cola. A este método se le pasará un objeto de tipo Callback con la implementación de lo que se deberá realizar cuando se reciba la respuesta del servidor.

4 CONCLUSIONES Y LÍNEAS FUTURAS

Es difícil decir lo que es imposible, porque el sueño de ayer es la esperanza de hoy y la realidad de mañana.

- Robert H. Goddard-

En esta última sección, se procederá a realizar un resumen a modo de conclusión con todo lo visto hasta ahora que servirá de recapitulación. Además, se indicarán una serie de líneas futuras que podrían suponer la elaboración de un sistema más completo.

4.1 Conclusiones

Partiendo de la toma de requisitos a través de entrevistas con los profesionales del dolor del Hospital Virgen del Rocío, se detectaron una serie de necesidades y problemas cuya solución final implicaría la elaboración de una aplicación móvil que permitiera digitalizar algunas de sus actividades diarias implicadas con la toma de decisiones clínicas para el tratamiento del dolor crónico. Cabe destacar que el hecho de haber tenido entrevistas con estos profesionales ha permitido asimilar y comprender mejor el problema existente.

Tras formalizar y analizar los requisitos, se procedió a diseñar un sistema cuya arquitectura principal suponía crear una aplicación móvil con sistema operativo Android cuyos datos fueran persistidos en un servidor de base de datos PostgreSQL externo. Para la comunicación entre la aplicación móvil y la base de datos, habrá un servidor intermedio que proporcionará una API REST. Esta comunicación se realizará de manera controlada gracias a los tokens JWT. El resultado final ha sido una aplicación plenamente funcional que resuelve los problemas planteados por los profesionales entrevistados.

Con relación a la implementación del servicio REST, el framework de Spring, aunque supuso una alta inversión de tiempo para profundizar en la tecnología, finalmente ha simplificado enormemente las labores de configuración del servidor. Además, ha permitido generar un código más limpio, claro y reutilizable. Cabe añadir que el módulo de Spring Data y el uso de JPA han permitido que el acceso a los datos se haga a través de una interfaz con la ventaja de que será independiente al sistema gestor de bases de datos utilizado (siempre que este sea compatible con JPA).

En este caso, el modelo de datos no resultó ser del todo complejo por lo que el acceso al mismo sin este tipo de interfaces no es tan relevante. Sin embargo, al crear una aplicación, siempre se debe pensar en su futura escalabilidad y mantenibilidad y se debe implementar de modo que el cambio de algún elemento del sistema suponga el menor impacto.

Por otra parte, para implementar la base de datos el uso de Docker también ha supuesto una alta inversión de tiempo de documentación tecnológica, sin embargo, ha merecido la pena ya que se trata de una tecnología muy usada en la actualidad. Además de los beneficios que ha supuesto en mi aprendizaje, su uso para el despliegue del servidor PostgreSQL y PgAdmin ha resultado muy beneficioso ya que ha permitido la abstracción de las dependencias de cada servicio y una configuración de este relativamente sencilla. Además, gracias a docker-compose se consigue un fichero que podrá usarse en cualquier sistema con Docker para desplegar los servicios facilitando así la portabilidad de estos.

Por último, al implementar la aplicación Android se han detectado algunos problemas y complejidades. Por ejemplo, se ha invertido mucho tiempo para intentar crear un diseño responsive apto para cualquier tipo de pantalla. Sin embargo, el tiempo invertido no se ve reflejado en el resultado final ya que no se ha conseguido que cumpla este requisito completamente. De este modo, quizás se tendría que haber profundizado más en este aspecto.

Por otra parte, la elección de sistema operativo Android limita el uso de la aplicación a este SO. Si bien es cierto que este será el SO en el hospital de los profesionales entrevistados, si el sistema llegara a usarse en otros hospitales (que es para lo que está diseñado), no se tendría la garantía de que este también fuera el SO usado. Así, se podría haber optado por tecnologías multiplataforma como NativeScript, que ofrece muchas ventajas para desarrolladores de aplicaciones móviles.

Como conclusión, resaltar que este trabajo de fin de grado me ha servido para acercarme un poco más al amplio mundo de las tecnologías aplicadas al sector sanitario. Además, han sido muchos los conceptos, herramientas y metodologías de Ingeniería de Software que he podido aprender y profundizar a lo largo de la elaboración del proyecto.

4.2 Líneas futuras

El objetivo principal de este proyecto ha sido elaborar la aplicación móvil que proporcionara las herramientas requeridas por los profesionales de la salud en las entrevistas realizadas. Aunque este objetivo se ha cumplido y se ha conseguido una aplicación totalmente funcional, dado que el tiempo para su desarrollo ha sido limitado, existen muchas líneas de mejora para la misma.

Por una parte, se podrían concertar más entrevistas para definir nuevos problemas que permitieran dotar a la aplicación de nuevas funcionalidades y herramientas clínicas.

Por otra parte, como versiones futuras, sería interesante implementar o mejorar aspectos como los que se enumerarán a continuación:

- Actualmente la seguridad del sistema se basa únicamente en la tecnología de tokens JWT. Se puede mejorar a través de mecanismos de seguridad específicos y más profundos.
- La autenticación en el sistema es del tipo “usuario-password”. En este mecanismo, el nivel de seguridad depende de la complejidad de la contraseña y de la integridad de esta y es muy probable que el usuario la olvide. En la actualidad, este mecanismo está siendo desbancado por otros que son mucho más seguros y cómodos para el usuario como la biometría o la autenticación a través de certificado digital. Alguna de estas técnicas podría ser incorporadas en la aplicación desarrollada.
- Se podría diseñar un módulo que permitiera obtener estadísticas de usabilidad del sistema, así como detectar anomalías o fallos en este.
- Se ha trabajado con servicios desplegados de manera local. Para que el sistema pueda ser finalmente usado, la aplicación móvil deberá ser instalada en los dispositivos finales y los servidores REST y PostgreSQL deberán ser desplegados en la nube. Esto último podría realizarse a través de Microsoft Azure, que ofrece una opción gratuita de hosting para estudiantes y además proporciona numerosas opciones útiles en la etapa de despliegue como posibilidad de testing, BackUp y monitorización.
- Actualmente, existen dos roles para los usuarios del sistema. Solo el gestor podrá realizar operaciones relacionadas con la alteración del catálogo de fármacos del hospital al que esté asociado. La asignación de roles se realiza cuando el usuario se registra en el sistema. Si indica un hospital que aún no existe, se crea al usuario con el rol de gestor y será el responsable de los fármacos del hospital indicado. En caso contrario, si indica un hospital ya existente, su rol será el de usuario simple en ese hospital (podrá ver el catálogo de fármacos, pero no modificarlo). Hoy en día solo se podría cambiar el rol del usuario desde la base de datos (no óptimo). Sería interesante crear un módulo que permita al gestor de cada hospital ver qué usuarios están asociados al mismo y ofrecer la posibilidad de modificar el rol de los usuarios deseados.
- Dado que las herramientas ofrecidas en la aplicación están relacionadas con procesos de cálculo matemáticos que si no son bien realizados pueden suponer un gran impacto sanitario, sería importante

añadir mecanismos de testing como pruebas unitarias que verifiquen que se obtienen los resultados deseados. Estas pruebas permitirán que, si se realiza cualquier modificación en el código, se pueda verificar de forma automática que ninguno de los resultados esperados se ha visto afectados. Sin este mecanismo, estas pruebas deberán realizarse de forma manual por lo que será mucho más tedioso, menos fiable y escalable. Para el testing del servicio REST podría usarse el framework de Junit que puede asociarse al proyecto de Spring de forma sencilla mediante *spring-boot-starter-test.*

- Se podría concertar una etapa de pruebas con los usuarios finales que permita detectar posibles anomalías, fallas y posibilidades de mejora.
- Se podría intentar refactorizar y mejorar la calidad del código desarrollado para cada componente.
- Sería interesante hacer un diseño la interfaz de usuario de la aplicación móvil totalmente responsive de modo que pueda visualizarse la aplicación de manera correcta en todos los dispositivos. Además, sería interesante que la aplicación fuera compatible con los sistemas operativos más usados en la actualidad: iOS y Android.

REFERENCIAS

[1] «Bombas de infusión,» [En línea]. Available: https://www.medtronic.com/es-es/tu-salud/tratamientos-y-terapias/dolor-por-cancer/dispositivo.html#what-is-it. [Último acceso: 9 Septiembre 2022].

[2] «Documentación Android,» [En línea]. Available: https://developer.android.com/studio/intro?hl=es-419. [Último acceso: 1 Septiembre 2022].

[3] «Java,» [En línea]. Available: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n). [Último acceso: 08 Septiembre 2022].

[4] «Servicios web,» [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/web-services/. [Último acceso: 08 Septiembre 2022].

[5] «Servicios REST y SOAP,» [En línea]. Available: https://www.chakray.com/es/cuales-son-las-ventajas-de-una-api-rest/. [Último acceso: 8 Septiembre 2022].

[6] S. Framework. [En línea]. Available: https://www.campusmvp.es/recursos/post/que-son-spring-framework-y-spring-boot-tu-primer-programa-java-con-este-framework.aspx. [Último acceso: 09 Septiembre 2022].

[7] «Spring Boot,» [En línea]. Available: https://www.geeksforgeeks.org/spring-boot-starters/#:~:text=Spring%20Boot%20Starters%20are%20dependency,dependencies%20under%20a%20single%20name.. [Último acceso: 8 Septiembre 2022].

[8] «Spring Boot,» [En línea]. Available: https://www.geeksforgeeks.org/spring-boot-starters/#:~:text=Spring%20Boot%20Starters%20are%20dependency,dependencies%20under%20a%20single%20name.. [Último acceso: 8 Septimebre 2022].

[9] «Spring Security,» [En línea]. Available: https://www.arteco-consulting.com/post/securizando-una-aplicacion-con-spring-boot. [Último acceso: 8 Septiembre 2022].

[10] DAO. [En línea]. Available: http://www.cursohibernate.es/doku.php?id=patrones:dao.

[11] «MVC,» [En línea]. Available: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html. [Último acceso: 08 Septiembre 2022].

[12] «JWT,» [En línea]. Available: https://www.bezkoder.com/spring-boot-refresh-token-jwt/. [Último acceso: 2 Agosto 2022].

[13] «JPA,» [En línea]. Available: https://www.campusmvp.es/recursos/post/la-api-de-persistencia-de-java-que-es-jpa-jpa-vs-hibernate-vs-eclipselink-vs-spring-jpa.aspx. [Último acceso: 8 Septiembre 2022].

[14] «Docker,» [En línea]. Available: https://coffeebytes.dev/que-es-docker-y-para-que-sirve/.

[15] «PostgreSQL,» [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/postgresql/. [Último acceso: 8 Septiembre 2022].

[16] «IntelliJ IDEA,» [En línea]. Available: https://www.jetbrains.com/es-es/idea/features/. [Último acceso: 8 Septiembre 2022].

[17] «Documentación Android,» [En línea]. Available: https://developer.android.com/studio/intro?hl=es-419. [Último acceso: 01 Septiembre 2022].

[18] «DataGrip,» [En línea]. Available: https://www.jetbrains.com/help/datagrip/meet-the-product.html. [Último acceso: 8 Septiembre 2022].

[19] «GIT,» [En línea]. Available: https://www.atlassian.com/es/git/tutorials/what-is-version-control. [Último acceso: 8 Septiembre 2022].

[20] «MagicDraw,» [En línea]. Available: https://www.magicdraw.com/main.php. [Último acceso: 9 Septiembre 2022].

[21] J. A. T. Muñiz, Apuntes de la asignatura Diseño de bases de dato, Departamento de Ingeniería Telemática: Universidad de Sevilla, 2021.

[22] «Modelo en 3 capas,» [En línea]. Available: https://www.ibm.com/es-es/cloud/learn/three-tier-architecture#:~:text=La%20arquitectura%20de%20tres%20niveles%20es%20una%20arquitectura%20de%20aplicaciones,datos%2C%20donde%20se%20almacenan%20y. [Último acceso: 4 Septiembre 2022].

ANEXO I: PROCESOS DE CÁLCULO

4.3 Problema de cálculo de mezclas:

Las bombas de infusión que se emplean en la unidad del dolor pueden ser de dos modelos posibles en función del volumen soportado:

1. Volumen de 20 ml
2. Volumen de 40 ml

Se desea que a partir de la introducción de:

- Nombre del fármaco A
- La concentración del fármaco A (C_A)
- La presentación del fármaco A (P_A)
- La dosis del fármaco A (D_A)
- Nombre del fármaco B
- La dosis del fármaco B (D_B)
- La presentación del fármaco B (P_B)

El sistema sea capaz de calcular cuál debe ser la mezcla que se debe añadir a la bomba de infusión:

- Volumen o carga del fármaco A (V_A)
- Volumen o carga del fármaco B (V_B)
- Volumen o carga de suero fisiológico (V_{SF})

Para explicar el proceso de cálculo, se va a incluir un pequeño ejemplo con los siguientes datos de entrada:

- Volumen bomba: 40 mL
- Fármaco A = Morfina
  - C_A =20 mg/mL
  - D_A = 7 mg/día
  - P_A = 40 mg/mL
- Fármaco B = Ziconotide
  - D_B = 1 ug/día
  - P_B = 100 ug/mL

El proceso de cálculo para obtener el resultado sería el siguiente:

1. **Calcular el flujo o velocidad de la bomba para el fármaco dominante (A):**

$$Flujo = \frac{D_A}{C_A}$$

En un día …

1 ml	20 mg (C_A)	$X = \frac{7mg * 1mL}{20mg} = 0.35\frac{mL}{dia}$
x	7mg (D_A)	

Se obtiene que la bomba entrega $0.35\frac{mL}{dia}$ del fármaco A al día.

2. **Calcular la concentración del fármaco B (secundario) limitada por la velocidad del fármaco A (dominante):**

$$C_{B,marcada\ por\ A} = \frac{D_B}{Flujo}$$

La velocidad la marca la dosis del fármaco A por lo que la dosis del fármaco B tiene que ir en arrastre, es decir, en el flujo diario calculado anteriormente ($0.35\frac{mL}{dia}$) debe ir la cantidad del fármaco B necesaria.

0.35 mL (flujo)	1 ug (D_B)	$C_{B,marcada\ por\ A} = \frac{1\ mL * 1\ ug}{0.35\ mL} = 2.86\frac{ug}{mL}$
1 mL	x	

A través de la dosis de B y el flujo de la bomba, se halla la concentración de fármaco B que tiene que haber dentro de la bomba.

3. Calcular la cantidad que debe haber en la bomba del fármaco A y del fármaco B:

$CantidadNecesaria_B = V_{bomba} * C_{B,marcada\ por\ A}$

$CantidadNecesaria_A = V_{bomba} * C_A$

1 mL	2.86 ug ($C_{B,marcada\ por\ A}$)	$CantidadNecesaria_B = 40\ mL * 2.86\frac{ug}{mL} = 114.3\ ug$
40 mL (V_{bomba})	x	

1 mL	20 mg (C_A)	$CantidadNecesaria_A = 40\ mL * 20\frac{mg}{mL} = 800\ mg$
40 mL (V_{bomba})	x	

4. Calcular la carga (volumen) que irá en la bomba de cada fármaco:

Se debe tener en cuenta cómo se consigue todo esto de farmacia.

$$Carga_x = \frac{CantidadNecesaria_x}{\boldsymbol{PresentaciónFarmacia}}$$

$$Carga_{SF} = Volumen_{bomba} - Carga_A - Carga_B$$

			Resultado
BOMBA (Volumen = 40 mL)	**Fármaco A**	$\frac{800\ mg}{40\frac{mg}{mL}}$	**20 *mL***
	Fármaco B	$\frac{114.4\ ug}{100\frac{ug}{mL}}$	**1.144 *mL***
	Suero Fisiológico	$40\ mL - 20\ mL - 1.144\ mL$	**18.856 *mL***

Nota: pueden existir tratamientos con un solo fármaco o más. En principio da igual que haya 2, 3 o 4 fármacos

porque para todos se toma la referencia del fármaco dominante (A) y no interactúan entre ellos. Es decir, si tenemos los 4 fármacos, habría que hacer 3 veces el cálculo que hacemos cuando tenemos 2 fármacos.

A continuación, se incluye una tabla con otro ejemplo que puede servir de resumen:

Volumen bomba	V_{bomba}	mL	**20 mL**
Dosis A	D_A	g/dia	**5 mg/dia**
Dosis B	D_B	g/dia	**0.2 mg/dia**
Presentación A	P_A	g/mL	**40 mg/mL**
Presentación B	P_B	g/mL	**2000 ug/mL**
Concentración A	C_A	g/mL	**20 mg/mL**
Flujo	$Flujo = \frac{D_A}{C_A}$	mL/dia	5 mg/día / 20 mg/mL = **0.25 ml/dia**
Concentración B,A	$C_{B,marcada\ por\ A} = \frac{D_B}{Flujo\ A}$	g/mL	0.2 mg/día / 0.25 mL/día = **0.8 mg/ml**
Cantidad A	$V_{bomba} * C_A$	g	20 mL * 20 mg/mL = **400 mg**
Cantidad B	$V_{bomba} * C_{B_A}$	g	20 mL * 0.8 mg/mL = **16 mg**
Carga A	$Carga_A = \frac{Canb\ tidadNecesaria_A}{PresentaciónFarmacia}$	mL	400mg / 40 mg/mL = **10 mL**
Carga B	$Carga_B = \frac{CantidadNecesaria_B}{PresentaciónFarmacia}$	mL	16 mg / 2mg/mL = **8 mL**
Carga SF	$Carga_{SF} = Vbomba - CargaA - Carga\_B$	mL	20mL – 10 mL– 8 mL = **2 mL**

Tabla 57. Resumen cálculo de mezclas

4.4 Problema del cálculo de dosis:

Se desea calcular cuánto debe incrementar la dosis de un fármaco cuando varía la dosis de otro fármaco de una mezcla que ya está en la bomba de infusión.

Suponiendo que las dosis recetadas a un paciente son las siguientes:

- Fármaco A: 7mg/día
- Fármaco B: 1 ug/día

Siendo el fármaco B un nuevo fármaco que se le introduce al paciente a través de una dosis baja para ver cómo es aceptado en su organismo.

Cuando este paciente vuelve (a los 15 días, por ejemplo) habrá que subirle la dosis del fármaco B si este ha sido aceptado correctamente.

- Nueva D_B: 1.5 ug/día

La dosis del fármaco A no se varía.

- **Opción 1:**

5. Vaciar la bomba.
6. Entrar en la aplicación de mezclas.
7. Calcularlo todo de nuevo teniendo en cuenta la nueva dosis de B.
8. Hacer la mezcla nueva.

Esto no es rentable ya que estaríamos desperdiciando fármacos.

- **Opción 2:**

Realizar una simulación que calcule cuánto debe variar la dosis del fármaco A si subo la dosis del fármaco B.

Para ello se calcula el porcentaje de incremento de B y se aplica en A.

De este modo, los datos de entrada estarán relacionados con la situación actual y la nueva.

- La dosis del fármaco A 7 mg/día
- La dosis actual del fármaco B: 1 ug/día
- La dosis nueva del fármaco B: 1.5 ug/día

Cálculo:

1 ug de B	7 mg de A	$X = \frac{1.5\ ug * 7mg}{1ug} = 10.5\ mg/día$
1.5 ug de B	X mg de A	

Se obtiene que la nueva dosis de fármaco A debe ser 10.5 mg/día.