UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Finglix 2.0 - Plataforma de dosificación de precisión informada por modelos

Darwin Araujo Moraes
Ignacio Daniel Vigna López
Juan Ignacio Betarte Dabosio

Proyecto de grado presentado a la Facultad de Ingeniería de la Universidad de la República en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniería en Computación.

Tutores
Dra.Ing. Laura González
Dr.Q.F. Manuel Ibarra
Usuario responsable
Q.F. Martín Uumpierrez

Montevideo, Uruguay
Setiembre de 2024

Agradecimientos

Este proyecto ha sido posible gracias al esfuerzo de muchas personas. Por ello, queremos comenzar expresando nuestra gratitud hacia nuestras familias y amigos, cuyo apoyo constante nos acompañó tanto durante el desarrollo de este proyecto como a lo largo de toda la carrera. También queremos agradecer a Martín Umpierrez, Manuel Ibarra, Andrés Baptista y a nuestra tutora Laura González, quienes con su cooperación hicieron posible la culminación de este trabajo. Finalmente, queremos agradecer el respaldo mutuo de los integrantes de este proyecto, lo cual fue fundamental para alcanzar este objetivo.

Resumen

El presente trabajo de grado se centra en el desarrollo y mejora de una plataforma ya existente para la dosificación de precisión informada por modelos farmacocinéticos, denominada Finglix. Este proyecto surgió de la necesidad de aplicar la farmacometría, una disciplina que emplea modelos matemáticos y estadísticos, para estimar las interacciones entre un fármaco y un organismo.

En el contexto clínico la aplicación de la farmacometría es fundamental ya que construye modelos que describen los procesos por los que es sometido un fármaco dentro del organismo. Esto incluye la absorción, distribución, metabolismo y excreción del fármaco. Al comprender estos procesos, es posible optimizar la dosificación, especialmente para aquellos fármacos con márgenes terapéuticos estrechos, donde una dosificación precisa es crucial para la seguridad y eficacia del tratamiento.

Como resultado de este proyecto se incorporaron nuevas funcionalidades y se realizaron mejoras sobre las ya existentes. Uno de los principales aportes fue la inserción de una nueva capa de abstracción, que permite a Finglix integrarse a múltiples herramientas de simulación. Esta nueva capa de abstracción, también fue desarrollada con el objetivo de permitir la carga de una mayor cantidad de modelos de fármacos diferentes. Además, se agregó la posibilidad de recomendar al usuario un mejor tratamiento posible para un paciente y objetivo determinado.

El trabajo también incluyó la implementación y despliegue continuo en la nube, optimizando el ciclo de retroalimentación con los involucrados. Además, se realizaron validaciones funcionales muy valiosas con usuarios finales, que derivaron en mejoras y nuevas funcionalidades. Finalmente, se generaron manuales de usuario, configuración y despliegue. El primero para facilitar el uso de los usuarios y los otros dos para ayudar a futuros involucrados en el desarrollo de Finglix.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se brinda el contexto y la motivación del proyecto Finglix, una plataforma destinada a la dosificación de precisión informada por modelos farmacocinéticos. Se explica la relevancia de la farmacometría en el ámbito clínico, destacando su importancia para medicamentos con márgenes terapéuticos estrechos. Además, se describe el trabajo previo realizado en la plataforma, incluyendo sus capacidades iniciales y las limitaciones encontradas. Luego se establecen los objetivos principales de esta tesis, sus resultados y principales aportes.

1.1. Contexto y motivación

La farmacometría es una disciplina que utiliza modelos matemáticos y estadísticos para comprender y predecir cómo interactúan los fármacos con el organismo y cómo progresa una enfermedad. Esta disciplina ayuda en la toma de decisiones durante el desarrollo y regulación de medicamentos, promoviendo el desarrollo de tratamientos seguros y eficaces, optimizando costos asociados a la investigación clínica [2].

En la medicina de precisión, la farmacometría juega un papel crucial al permitir la dosificación precisa de medicamentos. Esto significa que se pueden individualizar las dosis para cada paciente, maximizando las probabilidades de eficacia y seguridad según sus características personales y los factores del tratamiento. La capacidad de predecir la relación dosis-exposición a un fármaco en un paciente específico, basándose en datos poblacionales previamente caracterizados, es un avance significativo en el cuidado de la salud.

En las dos últimas décadas, la industria farmacéutica ha incorporado cada vez más el uso de aplicaciones de modelado y simulación para la dosificación de medicamentos. Estas herramientas resultan fundamentales para comprender cómo el cuerpo metaboliza los fármacos administrados, permitiendo identificar objetivos farmacológicos y aportando evidencia sobre la efectividad de los tratamientos. Además, estas herramientas son valiosas para orientar estrategias en las fases preclínicas y clínicas, con el fin de optimizar el desarrollo de nuevos medicamentos de manera eficiente y reducir los costos. Un factor clave es que el desarrollo de medicamentos comienza estudiando poblaciones de pacientes limitadas. Tras su aprobación, se continúa evaluando su efectividad y seguridad en la población general, lo que puede requerir ajustar la dosis para subpoblaciones específicas. Esta práctica es de gran relevancia

en el contexto clínico, aunque no esté ampliamente adoptada actualmente, y es particularmente importante para fármacos con estrecho margen terapéutico, donde la dosificación precisa tiene un impacto significativo.

En Uruguay, dentro del Departamento de Ciencias Farmacéuticas (CIENFAR) de la Facultad de Química, se han desarrollado diversos modelos farmacocinéticos utilizando principalmente los productos de la empresa Lixoft[1]. Fundada en 2011, esta empresa se dedica al desarrollo de software de modelado y simulación para la dosificación de precisión informada por modelos, dirigido específicamente a expertos en el campo. Sus productos ofrecen soluciones robustas para el análisis de poblaciones en ensayos preclínicos, clínicos y en la individualización de tratamientos. En el ámbito de la farmacología, el producto más destacado es MonolixSuite, compuesto principalmente por Monolix y Simulx que se enfocan en la estimación y simulación respectivamente. Sin embargo, últimamente el equipo de CIENFAR está realizando pruebas con modelos de otras herramientas de estimación y simulación, principalmente de código abierto.

En el contexto del proyecto Semillero del Espacio Interdisciplinario de la Universidad de la República, se desarrolló en una tesis de grado anterior a la presente: una "Plataforma para dosificación de precisión informada por modelos" denominada Finglix. Esta tesis se logró mediante la colaboración del Laboratorio de Integración de Sistemas (LINS) de la Facultad de Ingeniería de la UdelaR con el CIENFAR de la Facultad de Química. La problemática principal que abordó la plataforma radica en la necesidad de reducir la complejidad que tiene el uso de dos de las herramientas mencionadas que conforma MonolixSuite, ya que estas deben utilizarse por separado y realizar procesos manuales con los archivos generados. Este proceso implica un alto grado de complejidad y con alta probabilidad de cometer errores requiriéndose de competencias específicas en el área, con las que no siempre se cuenta en el país. El proyecto de grado Finglix buscó automatizar y simplificar este proceso.

La plataforma desarrollada permite a partir de un modelo matemático realizar simulaciones para un paciente específico, aprovechando los modelos preexistentes para MonolixSuite desarrollados por el equipo de CIENFAR. Los modelos de fármacos ingresados en la plataforma se utilizan para realizar simulaciones de un fármaco específico con distintos tratamientos e incorporar datos de los pacientes para realizar simulaciones. La plataforma ha sido diseñada para facilitar el trabajo a los farmacéuticos y médicos, simplificando procesos y mejorando la eficiencia al momento de obtener tratamientos más adecuados.

Si bien Finglix representó un avance significativo en la integración y automatización del proceso de dosificación informada por modelos, aún presentaba ciertas limitaciones. Entre ellas, la compatibilidad restringida a una única plataforma de estimación y simulación con un fuerte acoplamiento a ésta y la falta de incorporación y prueba de una mayor variedad de modelos farmacocinéticos. Además, se identificó la necesidad de mejorar la interfaz de usuario de algunas funcionalidades, la incorporación de nuevas funcionalidades que faciliten la obtención de un tratamiento y tener una versión en la nube apta para comenzar a ser utilizada por el equipo de CIENFAR.

[1] https://lixoft.com

Este proyecto se enfoca en abordar estas limitaciones mediante la extensión en la compatibilidad con múltiples plataformas de estimación y simulación, la incorporación de nuevos modelos, la simplificación del proceso de carga de los modelos y la realización de un despliegue en la nube más apto para un entorno productivo.

1.2. Objetivos

Se propone como objetivo continuar el estudio y desarrollo de una plataforma en el marco de la dosificación de precisión informada por modelos, que ayude a farmacéuticos y médicos en la elección de un tratamiento para sus pacientes de un determinado fármaco. En particular, se apunta a aumentar la flexibilidad y alcance de la herramienta, a través de la posibilidad de incorporar otras herramientas de simulación y de la mejora de la compatibilidad con MonolixSuite (p. ej. brindando soporte a modelos con características diferentes a los originalmente soportados).

Para cumplir con este objetivo general, se plantean los siguientes objetivos específicos:

1. Estudiar y analizar el marco conceptual referido a la problemática abordada por la plataforma.
2. Relevar las nuevas necesidades de los interesados, estudiando su viabilidad y prioridad para definir el alcance del proyecto.
3. Proponer soluciones para satisfacer las necesidades identificadas, haciendo un análisis del estado del proyecto anterior para comprender el estado del cual se parte.
4. Implementar y realizar un despliegue de las soluciones en la nube que permita al equipo del CIENFAR tener acceso a la plataforma, pudiendo ver los cambios realizados.
5. Validar la implementación y solución alcanzada con los interesados, con el fin de recibir retroalimentación para realizar cambios e identificar trabajos futuros.

1.3. Resultados y aportes del proyecto

Los principales resultados y aportes del proyecto son:

Marco teórico del dominio del proyecto
Se logró ampliar el marco teórico de la tesis anterior, realizando un estudio de otras herramientas que podían sustituir o complementar a MonolixSuite. Además, se revisitaron conceptos claves que, aunque algunos ya estaban presentes en la tesis anterior, son retomados y actualizados en este trabajo para garantizar que la tesis sea autocontenida y comprensible por sí misma.

Identificación de nuevas necesidades
Se logró identificar, analizar y priorizar las nuevas necesidades recolectadas de los involucrados. Se identificó tanto la necesidad de modificar funcionalidades existentes

como de agregar nuevas funcionalidades a la plataforma.

Propuestas de solución para las necesidades identificadas
El foco estuvo en mejorar la flexibilidad de la plataforma, brindando soporte a otras plataformas de simulación, así como soporte a múltiples fármacos y modelos. Además, se incorporaron funcionalidades tales como la de realizar simulaciones de una manera distinta a la ofrecida originalmente en la plataforma: partir de cierto objetivo que se quiere alcanzar, obtener cuál es el tratamiento que pueda lograr dicho objetivo con la mayor probabilidad según las características del paciente. Complementariamente, se agregó una funcionalidad que permite observar cuánto varían los nuevos datos simulados respecto a las observaciones ingresadas del paciente, para poder evaluar la adecuación del modelo a nivel individual.

Implementación y despliegue en la nube de las soluciones propuestas.
Se implementaron las soluciones propuestas para los requisitos relevados, la mayoría de forma completa. Además, se realizó un despliegue continuo en la nube el cual permitió obtener feedback rápidamente de los interesados y finalmente un despliegue más apto para un entorno productivo que el alcanzado por el proyecto anterior.

Validación funcional
Se realizaron instancias de validación con distintos involucrados que permitieron validar la adecuación funcional de la plataforma, identificar mejoras y otras funcionalidades que no fueron implementadas pero que se podrán estudiar en un trabajo futuro.

Documentación
Se crearon manuales, con el fin de facilitar el trabajo a futuro sobre la plataforma y ayudar a los usuarios a entender cómo funciona y cómo se debe utilizar la plataforma. El primero es el manual de configuración, que explica cómo crear el ambiente de desarrollo y cómo realizar el despliegue de la plataforma. Luego, se realizó el manual de usuario que explica paso a paso, con imágenes, las funcionalidades del sistema.

1.4. Organización del documento

El resto del documento se organiza de la siguiente forma.

En el Capítulo 2 se presenta el marco conceptual que permitirá el entendimiento del dominio, las decisiones y soluciones alcanzadas.

El Capítulo 3 describe el relevamiento y análisis de requisitos del sistema. En particular, se realiza un análisis de las necesidades del proyecto identificando de donde proviene cada una de ellas. Luego se describen cada uno de los requisitos funcionales y no funcionales, y finalmente se especifica la definición de alcance del proyecto.

En el Capítulo 4 se aborda el diseño de la solución, se describe la arquitectura general del proyecto, se presenta el modelo de dominio utilizado, las decisiones y soluciones de los principales requisitos que impactaron en la arquitectura.

El Capítulo 5 corresponde principalmente a la etapa de implementación. Se describen las tecnologías utilizadas y la decisión respecto a qué tecnologías se utilizaron en esta nueva versión de la plataforma. Luego, se explican los procesos involucrados en la construcción del *frontend* y el *backend* de las funcionalidades más relevantes.

En el Capítulo 6 se detalla el proceso de pruebas y validaciones realizadas. Se describen las pruebas unitarias y las validaciones con usuarios efectuadas para verificar y validar el funcionamiento del sistema.

En el Capítulo 7 se especifican las etapas y la gestión del proyecto. Se muestra el tiempo aproximado y en qué orden se desarrollaron las etapas. También, se explica cómo fue la metodología empleada.

Finalmente, en el Capítulo 8 se presentan las conclusiones obtenidas a partir del desarrollo del proyecto. Se resumen los resultados alcanzados, se evalúa el cumplimiento de los objetivos y se reflexiona sobre posibles mejoras y trabajos futuros que podrían realizarse.

Capítulo 2

Marco conceptual

En este capítulo se presentan los principales conceptos que serán utilizados a lo largo de este documento, que buscan establecer una base conceptual del dominio del problema para el lector.

2.1. Farmacología

El principal objetivo del tratamiento farmacológico es alcanzar una respuesta clínica favorable en cada paciente, provocando los efectos buscados y minimizando los efectos adversos [3]. Sin embargo, la variabilidad en la respuesta a los medicamentos dentro de una población dada dificulta esta tarea. La dosificación de precisión, en contraposición al enfoque tradicional de talla única (enfoque de una única posología para los pacientes independientemente de sus características), busca adaptar los regímenes de tratamiento a las características individuales que afectan la absorción, disposición y respuesta del medicamento. La dosificación de precisión informada por modelos (MIPD) emplea modelos matemáticos y estadísticos, no lineales de efectos mixtos, combinados con datos del paciente para guiar las decisiones clínicas, asegurando la administración del fármaco correcto, en la dosis adecuada, al paciente indicado. Además, los modelos farmacométricos permiten cuantificar el impacto de diversos factores, como el tamaño corporal, la edad y la función de los órganos, en la dosificación del medicamento, lo que facilita una optimización más precisa.

La implementación de MIPD ofrece la posibilidad de reducir la brecha entre la eficacia observada en ensayos clínicos y la efectividad en la práctica clínica real, lo que resulta especialmente relevante para poblaciones especiales que pueden no estar adecuadamente representadas en los estudios clínicos [4]. En resumen, MIPD representa una herramienta prometedora para mejorar la precisión y eficacia del tratamiento farmacológico, adaptándolo a las necesidades individuales de cada paciente.

2.1.1. Farmacocinética

La farmacocinética es una disciplina que estudia los procesos por los que un fármaco se somete a través del paso por el organismo [5]. Se encarga de estudiar qué sucede con el medicamento desde su administración hasta su eliminación del cuerpo.

2.1.2. Modelo y sistema

Un modelo se define como una estructura matemática que representa el sistema, empleado para examinar su estructura y comportamiento [3]. Esta estructura matemática consiste en un conjunto de entradas que relacionan variables independientes (como el tiempo y la dosis administrada) y características individuales (como la edad del paciente) con una variable de interés (como la concentración del fármaco o el efecto del fármaco), así como variables que cuantifican procesos relevantes (cómo se distribuye el fármaco en el sistema).

El sistema, por su parte, se refiere a un conjunto de elementos individuales que operan en conjunto para formar componentes más complejos como una célula, un tejido, un órgano o un organismo completo, incluyendo su interacción con el fármaco [3][4].

Es fundamental resaltar que un modelo es una versión simplificada del sistema, que frecuentemente incorpora varios supuestos necesarios para su construcción, basados en el conocimiento y la información disponible.

2.1.3. Farmacodinámica

La farmacodinámica es una disciplina que estudia los efectos bioquímicos y fisiológicos de los fármacos, así como sus mecanismos de acción sobre el organismo [6]. Se han desarrollado diferentes modelos que simplifican los numerosos procesos que tienen lugar entre el organismo y el fármaco. Los modelos de farmacodinámica se abrevian como modelos PD mientras que los modelos de la farmacocinética se abrevian como PK. Por lo general, se los estudia en conjunto, en los modelos PK/PD, dado que tienen una relación directa entre sí.

2.1.4. Dosificación de precisión informada por modelos

La dosificación de precisión informada por modelos se define como la personalización de los regímenes de tratamiento farmacológico en función de las características individuales del paciente que pueden afectar la absorción y disposición del principio activo del medicamento y la respuesta del organismo [7]. Tradicionalmente, en el caso de fármacos con un margen terapéutico estrecho, la dosis se ajusta según la exposición del paciente al fármaco, medida a través de la concentración en sangre y comparada con un rango terapéutico poblacional. Los modelos permiten avanzar más allá, aprovechando de manera más efectiva los datos específicos de cada paciente.

2.1.5. Modelo farmacocinético poblacional

La farmacocinética poblacional analiza la variabilidad interindividual en las concentraciones plasmáticas de los fármacos cuando se administran regímenes de dosificación estándar a un grupo de pacientes con características clínicas específicas [7]. Factores como las características demográficas, fisiopatológicas y terapéuticas, incluyendo el peso corporal, las funciones metabólicas y excretoras, y la coexistencia de otras terapias, pueden modificar la relación entre la dosis y la concentración del fármaco.

Entre los factores fisiopatológicos y clínicos que pueden afectar las características farmacocinéticas se encuentran:

- Demográficas: edad, peso o superficie corporal, género, raza, etc.
- Ambientales: tabaquismo, dieta, etc.
- Fisiológicas-fisiopatológicas: insuficiencia renal (depuración de creatinina), insuficiencia hepática, hipertensión arterial sistémica, etc.
- Variabilidad por el medicamento: formas farmacéuticas, vías de administración e interacciones farmacológicas.

Una de las principales aplicaciones de los estudios de farmacocinética poblacional es identificar las variables predictoras que influyen en el comportamiento cinético del fármaco, destacando el género, la edad y el peso, así como poblaciones especiales [7].

Estos modelos proporcionan una guía inicial para desarrollar regímenes de dosificación que permitan alcanzar y mantener una concentración plasmática adecuada en cada paciente.

2.1.6. Modelo de covariables

Una covariable es cualquier variable específica de un individuo que puede influir en la farmacocinética y/o farmacodinamia de un fármaco [7]. Estas pueden incluir indicadores demográficos (como sexo, peso corporal, etnia), valores de laboratorio (como suero, creatinina, bilirrubina, albúmina), parámetros de la enfermedad (como el estado general, la duración de la enfermedad, insuficiencia renal), y características terapéuticas (como la administración de medicamentos, alimentación/ayuno), entre otros. Además, estas variables pueden ser continuas (como el peso corporal, el aclaramiento de creatinina), dicotómicas (como el sexo), categóricas ordenadas (como los puntajes de salud), y categóricas no ordenadas (como el genotipo, la etnicidad).

Identificar covariables y cuantificar su impacto en parámetros específicos del sistema es uno de los objetivos principales de un análisis poblacional, ya que permite explicar la variabilidad interindividual (e interocasional) [7]. Esto mejora el rendimiento predictivo de un modelo, ya que se pueden utilizar características específicas del individuo para aumentar la precisión en la predicción de parámetros individuales. Además, ciertas poblaciones especiales pueden ser caracterizadas cuantitativamente.

2.2. Aspectos farmacocinéticos

Esta sección trata aspectos de la farmacocinética que se consideran necesarios para entender de forma introductoria el dominio, y principalmente para ayudar a comprender mejor los análisis y soluciones que se realizarán en los próximos capítulos.

2.2.1. Posología

La posología [8] [9] es la definición de la cantidad dosis a administrar, incluyendo el intervalo de administración. Además, establece la duración total del tratamiento para asegurar su eficacia y seguridad. Para definir una posología adecuada se pueden considerar diversos factores como el peso, la edad, el estado físico y fisiológico del paciente, y la forma en que se presenta el medicamento (como cápsulas, comprimidos, gotas, etc.).

2.2.2. Compartimentos

En farmacocinética, los individuos suelen dividirse por sitios fisiológicos (fluidos, tejidos, órganos, etc) que comparten un comportamiento similar de absorción y distribución de un fármaco, es decir que las concentración de ese fármaco están en equilibrio en esas zonas. Esta categorización se denomina compartimentos [10] y comúnmente se consideran los siguientes tres compartimentos [11] :

- Compartimiento Central: Agua plasmática e intracelular fácilmente accesible y tejidos bien irrigados como el corazón, pulmón e hígado.
- Compartimiento Periférico Superficial: Agua intracelular poco accesible y tejidos menos irrigados como la piel, grasa, músculo y medula ósea
- Compartimiento Periférico Profundo: Depósitos tisulares a los que el fármaco se une más fuertemente y de los que por lo tanto se libera más lentamente

A su vez, los modelos son también categorizados por cantidad de compartimentos dependiendo de como se comporte la distribución del fármaco en los individuos (monocompartimentales, bicompartimentales y tricompartimentales).

2.2.3. Estado estacionario

El estado estacionario [12] es la condición en la que la velocidad media de ingreso de un fármaco al cuerpo es igual a la velocidad media de eliminación, calculada dentro del intervalo de administración. En este estado, aunque las concentraciones del fármaco pueden fluctuar, lo hacen de manera constante y predecible si se mantiene la misma dosis y frecuencia de administración. Esto significa que, aunque las concentraciones del fármaco varíen dentro del intervalo interdosis, la velocidad media de ingreso y eliminación es equivalente, permitiendo así un equilibrio en el sistema.

2.2.4. TSS (*Time to Stationary State*)

El TSS [12] es el período estimado en el cual se espera que un fármaco alcance niveles estables en el organismo después de dosis repetidas y regulares. Se calcula principalmente a partir de la semivida de eliminación del fármaco.

2.2.5. Métricas farmacocinéticas

Las métricas se utilizan para medir la exposición a un fármaco y son de gran utilidad para definir objetivos de exposición.

AUC

El área bajo la curva [13] (**AUC**: *area under the curve*) representa el área bajo la curva de la concentración de un fármaco para un intervalo de tiempo dado, donde es tomado como punto de partida el momento en el que se realiza la primera administración en estado estacionario y como punto final el momento al qué corresponde la administración de la dosis siguiente. Como se puede observar en la Figura 2.1, el intervalo en el que se mide el área bajo la curva coincide con el intervalo de administración del tratamiento.

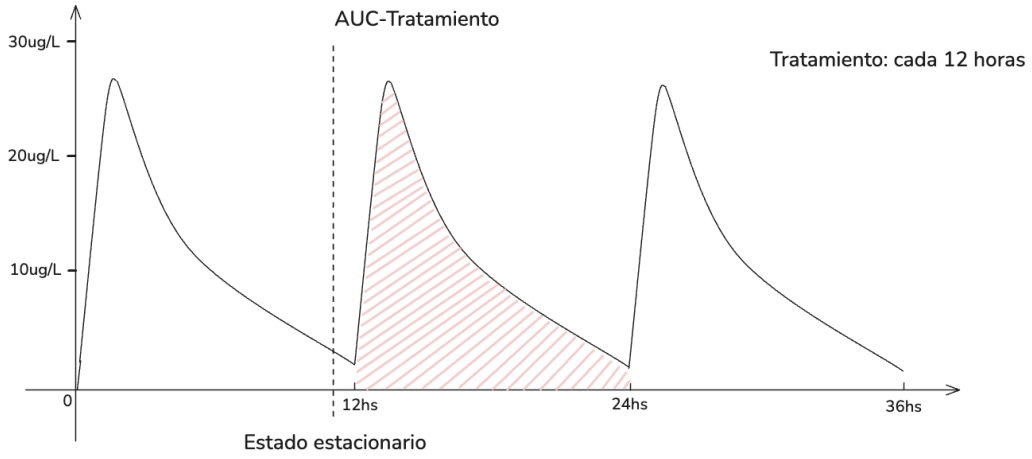


Figura 2.1: Gráfica AUC

De manera similar, se define **AUC (X)**, pero a diferencia del AUC a secas, el intervalo es fijo (X), no depende del intervalo de administración utilizado en el tratamiento. Como se visualiza en la Figura 2.2, en este ejemplo el intervalo de administración es de 12 horas, pero el intervalo en el que se mide el área bajo la curva es 24 horas, en otras palabras, se mide el AUC (24).

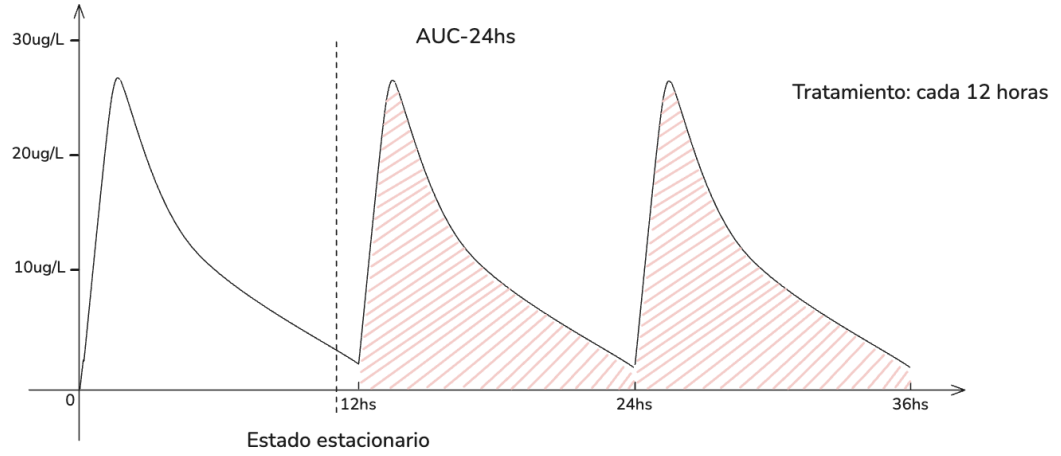


Figura 2.2: Gráfica AUC 24hs

Concentración mínima y máxima
La concentración mínima y máxima son los valores mínimos y máximos de la concentración obtenida a partir del estado estacionario.

2.3. Fármacos de interés

En esta sección se describen los fármacos para los cuales se utilizaron modelos proporcionados por el equipo del Departamento de Ciencias Farmacéuticas (CIENFAR) de la Facultad de Química.

Ciclosporina
La Ciclosporina [14] es un fármaco inmunosupresor utilizado en trasplantes de órganos para reducir la actividad del sistema inmunitario y disminuir el riesgo de rechazo del órgano trasplantado. Es un péptido cíclico de once aminoácidos producido por el hongo Tolypocladium inflatum Gams, originalmente aislado de una muestra de suelo noruego. Se ha estudiado su eficacia en diversos tipos de trasplantes, incluyendo piel, corazón, riñón, pulmón, páncreas, médula ósea e intestino.

Tacrolimus
El tacrólimus [15] [16], también llamado FK-506 o fujimicina, es un fármaco inmunosupresor utilizado principalmente después de trasplantes para prevenir el rechazo del órgano. Actúa reduciendo la actividad de las células T y la interleucina 2 (IL-2). Se usa también en dermatitis grave, uveítis refractaria post-trasplante de médula ósea y vitíligo. Es un macrólido descubierto en 1984 en una muestra de suelo japonés que contenía Streptomyces tsukubaensis.

Vancomicina
La vancomicina [17][18], un glucopéptido producido naturalmente por Nocardia orientalis, es un antibiótico efectivo contra bacterias grampositivas como S. aureus, S. pyogenes y S. pneumoniae. Su mecanismo de acción consiste en inhibir la síntesis de la pared celular bacteriana al interferir con la transglucosidasa. Sin embargo, debido a su tamaño molecular, no es tan eficaz contra bacterias gramnegativas, ya que no puede atravesar su membrana externa.

2.4. Herramientas de simulación

En esta sección se describen las principales herramientas de simulación vinculadas al proyecto.

2.4.1. MonolixSuite

MonolixSuite[1][19] es un conjunto de herramientas que pueden utilizarse de manera conjunta para la creación de modelos, simulaciones y visualización de datos de la

[1] `https://lixoft.com/products/`

empresa Lixsoft. Está compuesto por las herramientas: Monolix, Simulx y PKanalix. La Suite está disponible bajo distintos tipos de licencias, para adaptarse a las necesidades específicas de los usuarios. Se ofrecen licencias individuales para PKanalix, Monolix y Simulx, o licencia completa que incluye todas las herramientas. Las organizaciones académicas y sin fines de lucro tienen la posibilidad de obtener una licencia gratuita para actividades no comerciales, lo cual promueve el uso de la *Suite* Monolix en investigaciones académicas y proyectos educativos sin costo alguno.

Monolix
Monolix [20] es una herramienta de MonolixSuite para la creación de Modelos de Efectos Mixtos No Lineales (NLME) y permite estimar parámetros a partir del análisis de poblaciones. Otras características que ofrece esta herramienta son: la generación automática de pruebas diagnósticas, apoyos estadísticos, estimación de errores, comparación entre modelos y una interfaz gráfica de usuario que facilita su uso. Monolix también cuenta con documentación [21] completa, ejemplos y una comunidad activa de usuarios.

Simulx
El objetivo principal de Simulx [22] es realizar simulaciones de ensayos clínicos. En otras palabras, simula una población de individuos, en uno o varios grupos, utilizando diferentes escenarios. Constituye el último paso del flujo de trabajo de modelado y simulación, tras la creación de modelos NLME en Monolix. Simulx cuenta con una API [22] completa para sus funcionalidades en R, y cuenta con documentación completa y actualizada.

LixoftConnectors
LixoftConnectors es una API que permite ejecutar funciones de las herramientas de *software* Monolix [23] y Simulx [24] utilizando el lenguaje de programación R. Esta API, facilita la integración y el uso de estas herramientas dentro de los flujos de trabajo en R.

Regresores
Los regresores [25] son variables de Monolix que varían en función del tiempo y se utilizan para representar la influencia de variables externas sobre el comportamiento de un sistema. Estas variables no están directamente definidas en el modelo en todos los puntos de tiempo, pero se interpolan a partir de valores observados en momentos específicos. La incorporación de regresores en los modelos farmacocinéticos permite ajustar el modelo a datos observados y predecir cómo variarán las concentraciones del fármaco en función de diferentes condiciones y características individuales.

2.4.2. Mrgsolve

Mrgsolve [26] es una herramienta de código abierto y gratuita que permite la ejecución de simulaciones de modelos jerárquicos basados en ecuaciones diferenciales ordinarias (ODE), en especial para realizar simulaciones de cómo varia la concentración de un fármaco en el cuerpo para un tratamiento específico. Se utiliza en una variedad de aplicaciones de modelado, como farmacocinética (PK), farmacocinética/farmacodinámica (PK/PD), modelado farmacocinético basado en la fisiología

(PBPK) y farmacología de sistemas cuantitativos. El desarrollo de Mrgsolve se lleva a cabo en GitHub, con contribuciones de la comunidad de modelado y simulación farmacéutica.

2.4.3. Nlmixr2

Nlmixr2 [27] es una herramienta de código abierto diseñada para facilitar la implementación de efectos mixtos no lineales en R. Al ser de código abierto, está disponible para los usuarios de R de forma gratuita. Además, cuenta con una documentación completa que incluye ejemplos prácticos para ayudar a los usuarios a familiarizarse con su funcionamiento.

2.5. Conceptos vinculados al software

En esta sección se describen varios conceptos relacionados con el desarrollo, la gestión y el funcionamiento de software en diversas aplicaciones y contextos.

2.5.1. Frontoffice y Backoffice

El frontoffice en el contexto del *software* es la interfaz visible para los usuarios finales. Es donde ocurre la interacción directa entre el usuario y el sistema, permitiendo acciones como ingresar información, realizar transacciones o acceder a funciones específicas. Esta parte del *software* se centra en proporcionar una experiencia intuitiva y fácil de usar para los usuarios, ya que es la cara visible del sistema.

Por otro lado, el *backoffice* es la parte invisible del *software* que realiza tareas administrativas y de gestión de datos en segundo plano. Aquí es donde se procesan los datos, se gestionan las bases de datos y se automatizan los procesos comerciales [28]. Aunque no es visible para los usuarios finales, el *backoffice* es fundamental para garantizar que el *software* funcione de manera eficiente y efectiva, proporcionando el soporte necesario para las operaciones del *frontoffice*, y es normalmente gestionado por los administradores.

2.5.2. *Plugins (software)*

Un *plugin* es un pequeño programa que se añade a una aplicación principal para proporcionarle nuevas funcionalidades. Este concepto es esencial en áreas como la informática, el desarrollo web y las aplicaciones de edición audiovisual. Un *plugin* no puede funcionar solo; necesita integrarse con una aplicación o sistema principal para operar.

En el software, los *plugins* se usan de múltiples formas. Por ejemplo, en navegadores web como Chrome y Edge, los *plugins* permiten bloquear anuncios, corregir gramática y gestionar contraseñas. En plataformas de gestión de contenido como WordPress, los *plugins* ayudan a crear tiendas en línea, optimizar contenido para SEO y diseñar páginas web fácilmente. En programas de edición como Photoshop, los plugins añaden efectos avanzados. En clientes de correo como Outlook y Gmail, permiten

programar reuniones y gestionar calendarios eficientemente.

Algunas de las ventajas que ofrecen los *plugins* son:

- Añadir características que el *software* original no tiene, brindando flexibilidad y personalización.
- Su instalación y actualización son generalmente sencillas, lo que ahorra tiempo a usuarios y desarrolladores.
- La modularidad de los *plugins* facilita el mantenimiento del *software*, ya que se pueden agregar o actualizar funcionalidades sin modificar el núcleo del programa.

Sin embargo, también presenta desventajas:

- Pueden ser inseguros si provienen de fuentes no confiables, lo que puede introducir vulnerabilidades.
- El uso excesivo de *plugins* puede ralentizar el sistema, especialmente si no están bien optimizados.
- Los *plugins* también pueden entrar en conflicto entre sí o con el *software* principal, causando errores.
- La dependencia del desarrollador es un problema, ya que algunos *plugins* pueden quedar obsoletos o sin soporte, creando problemas con actualizaciones del software principal.

Para que un sistema utilice *plugins* de manera efectiva, debe cumplir ciertos requisitos. El *software* principal debe estar diseñado para soportar *plugins*, lo que implica una arquitectura modular y un sistema de API que permita la integración de terceros. Es útil tener repositorios oficiales donde los usuarios puedan encontrar, descargar e instalar *plugins* de manera segura. Además, el *software* debe ofrecer herramientas para gestionar los *plugins*, como la instalación, configuración, actualización y desactivación. También es crucial contar con buena documentación y soporte para que los desarrolladores puedan crear y mantener sus *plugins* efectivamente.
[29] [30] [31].

2.5.3. DLL (dynamic link library)

Una DLL [32] es un archivo que contiene código y datos compartidos por múltiples programas en sistemas Windows, promoviendo la modularización del código y la eficiencia en el uso de recursos. Sin embargo, su uso puede generar dependencias entre programas, causando conflictos si la DLL se actualiza o modifica, ofrecen ventajas como autocontención y control de versiones.

2.5.4. VPS (Virtual Private Server)

Un servidor privado virtual (VPS) [33] es una máquina que aloja todo el *software* y los datos necesarios para operar una aplicación o un sitio web. Se denomina virtual porque solo utiliza una porción de los recursos físicos del servidor, los cuales son gestionados por un proveedor externo. Algunos usos que se le pueden dar a un VPS son: despliegue de aplicaciones web, creación de entornos de prueba y almacenamiento secundario.

2.6. Historia Clínica Digital en Uruguay

En Uruguay, la Historia Clínica Digital [34] se basa en una plataforma tecnológica llamada Historia Clínica Electrónica Nacional (HCEN) [35]. Esta herramienta permite el intercambio de información clínica de los pacientes cada vez que tienen un evento asistencial, ya sea en prestadores de servicios de salud públicos o privados.

A través de la HCEN, la información clínica de los ciudadanos está disponible y accesible para el equipo de salud de manera oportuna, segura y en línea, sin importar el lugar geográfico ni el prestador de salud al que acuda el usuario.

La estrategia adoptada en Uruguay para integrar la información clínica de los usuarios de salud se fundamenta en un sistema federado, que sigue los estándares internacionales para el intercambio de información clínica. Este sistema incluye una plataforma central llamada Plataforma de Historia Clínica Electrónica Nacional, la cual permite el intercambio en tiempo real y de manera segura de los datos clínicos almacenados en cada institución que los generó.

La aplicación "Mi historia clínica digital" permite a todas las personas mayores de 18 años del Sistema Nacional Integrado de Salud (SNIS) acceder a su información clínica digital y gestionar permisos y privacidad desde cualquier dispositivo. Esta plataforma no almacena la información clínica, sino que facilita la comunicación entre los sistemas de los distintos prestadores de salud, quienes son los encargados de custodiar los datos según la normativa vigente.

El programa Salud.uy [36], encargado del desarrollo de la HCEN, comenzó en 2012 bajo la Agencia de Gobierno Electrónico y Sociedad de la Información y el Conocimiento (AGESIC) del Estado uruguayo. Este programa [37], cogobernado por Presidencia de la República, el Ministerio de Salud Pública, el Ministerio de Economía y Finanzas, y AGESIC, busca integrar las tecnologías de la información y comunicación en el Sistema Nacional Integrado de Salud (SNIS). Su financiación proviene tanto del Estado uruguayo como del Banco Interamericano de Desarrollo.

2.7. Tecnologías

En esta sección se describen tecnologías asociadas tanto al proyecto Finglix original, como a la nueva versión desarrollada en esta tesis.

2.7.1. Python

Python [38][39] es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su legibilidad y simplicidad. Fue creado por Guido van Rossum y su primera versión se distribuyó en 1991. Python es ampliamente utilizado en diversas áreas de la informática y la programación, tales como desarrollo web, análisis de datos, inteligencia artificial, automatización de tareas, desarrollo de *software*, y más.

Algunas características clave de Python son:

- Sintaxis Sencilla y Legible: la sintaxis está diseñada para ser fácil de leer y escribir.
- Multiparadigma: soporta varios paradigmas de programación, incluyendo programación estructurada, orientada a objetos y funcional.
- Interpretable: se ejecuta mediante un intérprete, lo que permite ejecutar el código línea por línea.
- Portabilidad: es multiplataforma, lo que significa que el mismo código puede ejecutarse en diferentes sistemas operativos como Windows, MacOS y Linux.
- Extensible: se puede extender con módulos y bibliotecas escritos en otros lenguajes como C o C++, lo que permite mejorar su rendimiento y funcionalidad.

2.7.2. Django

Es un *framework* de Python gratis y de código abierto, que respeta el patrón de diseño conocido como modelo vista controlador (MVC), donde el modelo representa los datos y la lógica de la aplicación, la vista maneja la presentación de los datos y el controlador maneja las solicitudes del usuario. Además, Django cuenta con su propio ORM (Object Relational Mapping) que facilita la interacción con la base de datos.
Adicionalmente este framework proporciona un administrador automático para gestionar los modelos de datos de manera fácil y rápida. Esto es útil durante el desarrollo y para tareas administrativas en producción.
Finalmente incluye medidas de seguridad integradas para proteger las aplicaciones contra vulnerabilidades comunes, como ataques de inyección SQL, ataques CSRF (Cross-Site Request Forgery), entre otros.

2.7.3. PostgreSQL

PostgreSQL [40] [41] es una plataforma gratuita y de código abierto para administrar bases de datos relacionales. En la implementación de la mayoría de sus funcionalidades utiliza el estándar SQL. PostgreSQL es operativo en los principales sistemas operativos. Además, cumple con las reglas ACID (atomicity, consistency, isolation, durability)

2.7.4. R

R [42] es un lenguaje de programación interpretado, diseñado específicamente para el análisis estadístico y la representación gráfica de los resultados obtenido. Actualmente, es uno de los lenguajes de programación más utilizados en la investigación científica, siendo muy popular en áreas como el aprendizaje automático, la minería de datos, la bioinformática, la investigación biomédica y las matemáticas financieras.

Las características más destacadas de R son:

- Orientado a Objetos: es un lenguaje orientado a objetos.
- Especialización en Estadística y Análisis de Datos: se diseñó específicamente para el análisis de datos estadísticos. Incluye una amplia gama de herramientas para modelos lineales y no lineales, pruebas estadísticas, series temporales, clasificación, agrupamiento y otros métodos de análisis de datos.
- Gráficos y Visualización: posee algunas capacidades gráficas. La librería ggplot2 es particularmente popular para la visualización de datos.
- Paquetes Extensibles: tiene muchos paquetes adicionales que extienden sus capacidades. Estos paquetes son desarrollados tanto por la comunidad de usuarios como por organizaciones y se pueden descargar e instalar fácilmente desde CRAN (Comprehensive R Archive Network)[2].
- Lenguaje Interpretado: es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea y no necesita ser compilado antes de ejecutarse.
- Vectores y Operaciones Vectorizadas: maneja los datos como vectores y matrices, y muchas operaciones están vectorizadas, lo que permite ejecutar operaciones en bloques de datos sin necesidad de bucles explícitos, mejorando la eficiencia.
- Plataforma Abierta y Libre: es de código abierto y está disponible de forma gratuita bajo la licencia GNU General Public License (GPL).
- Integración y Conectividad: se integra bien con otros lenguajes de programación y sistemas de bases de datos, lo que facilita la importación y exportación de datos.

2.7.5. React

React [43] es una librería de Javascript [44] para interfaces de usuario web, creada por Facebook en 2013, y es una de las más populares actualmente.

La base de React son los componentes, que son piezas reutilizables de código que encapsulan la lógica y la interfaz de usuario. Cada componente puede tener su propio estado interno y propiedades que se pueden pasar desde otros componentes. React utiliza un Virtual DOM (Document Object Model) para mejorar el rendimiento de

[2] `https://cran.r-project.org/mirrors.html`

las aplicaciones. El Virtual DOM es una representación virtual y ligera de la estructura de la interfaz de usuario en memoria. React compara este Virtual DOM con el DOM real y actualiza solo las partes que han cambiado, lo que minimiza la manipulación directa del DOM y mejora la eficiencia.

Utiliza una extensión de JavaScript llamada JSX que permite escribir código HTML dentro de archivos de JavaScript. Esto facilita la creación de componentes de React al combinar lógica y marcado en un solo lugar.

2.7.6. Material UI

Material-UI [45] es una biblioteca de componentes de interfaz de usuario (UI) de React que sigue las directrices de diseño de Material Design [46]. Material-UI proporciona una amplia gama de componentes predefinidos y estilizados.

2.7.7. Supademo

Supademo [47] es una herramienta que permite crear demos interactivas de productos. Las demos son construidas con la conversión de vídeos y capturas de pantalla con anotaciones de texto. Los usuarios pueden compartir las demos a través de enlaces, incrustaciones o como archivos de vídeo. Esto permite mejorar la capacitación de uso del sistema y generando documentación de uso del sistema de una forma más simple. .

Capítulo 3

Relevamiento y análisis

En este capítulo se detalla la recopilación de requisitos para la plataforma Finglix. Este proceso se basa en tres aspectos principales como se presenta en la Figura 3.1, primero, el análisis y estudio del sistema previo, que de ahora en más se denominará Finglix 1.0. Segundo, las reuniones con el equipo del CIENFAR, que proporcionaron información valiosa sobre el dominio y uso de Finglix 1.0, además de ofrecer una lista de posibles características y mejoras. Por último, el estudio de herramientas relacionadas, incluyendo aquellas con funcionalidades similares a Finglix 1.0 y otras herramientas para integrar en la plataforma, como herramientas de simulación y estimación. Este estudio permitió identificar mejoras y nuevas características.

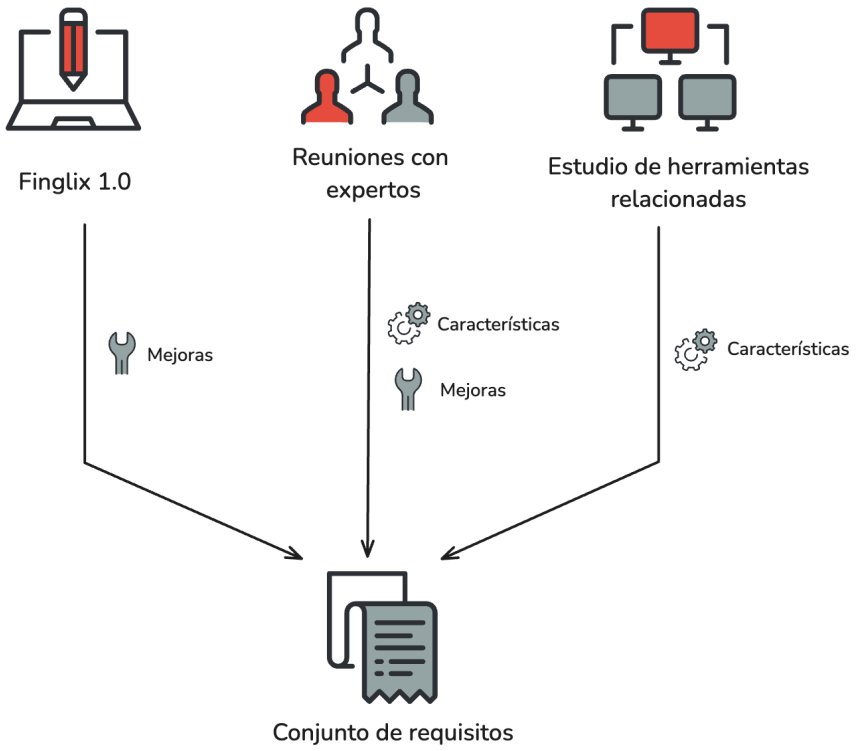


Figura 3.1: Recopilación de requisitos

3.1. Metodología de análisis

Las etapas mencionadas se llevaron a cabo de manera paralela e iterativa. Mientras se analizaban y comprendían las funcionalidades existentes en Finglix 1.0, se realizaban reuniones periódicas con el equipo del CIENFAR. Las primeras reuniones incluían a gran parte de este equipo, y a medida que se avanzaba, se efectuaban principalmente con el usuario responsable designado, debido a su amplio conocimiento del dominio y las necesidades del equipo de CIENFAR. Esto también permitió realizar reuniones más frecuentes gracias a su disponibilidad horaria. Aunque el estudio de herramientas relacionadas también se realizó en paralelo, los requisitos que surgieron de este estudio se identificaron después de comprender profundamente Finglix 1.0 y adquirir un mayor conocimiento del dominio y las necesidades del equipo.

En las primeras reuniones se identificó la necesidad de realizar simulaciones con nuevos modelos y fármacos, en particular modelos del fármaco Tacrolimus. Estos modelos presentaban algunas características distintas a las que soportaba la plataforma previamente. En las reuniones siguientes, se elaboró una lista de necesidades. Entre ellas, se consideró la más desafiante y compleja: adaptar la plataforma para integrarse con múltiples herramientas de simulación y estimación. También se identificó la necesidad de obtener un tratamiento dado un objetivo específico. Además, era importante para los farmacéuticos poder comparar visualmente cómo se comporta el modelo respecto a las observaciones agregadas. Finalmente, se identificaron algunas mejoras menores sobre las funcionalidades ya existentes.

3.2. Funcionalidades de Finglix 1.0

En esta sección se describen las principales funcionalidades de Finglix 1.0, que sirvieron de base para identificar nuevos requisitos. En la Figura 3.2, se observa gráficamente las principales funcionalidades que presenta Finglix 1.0, se resalta que, tanto las funcionalidades de simular como estimar, en esta primera versión, utilizan únicamente MonolixSuite. En las siguientes secciones se describen estas funcionalidades y algunas limitaciones que presentan.

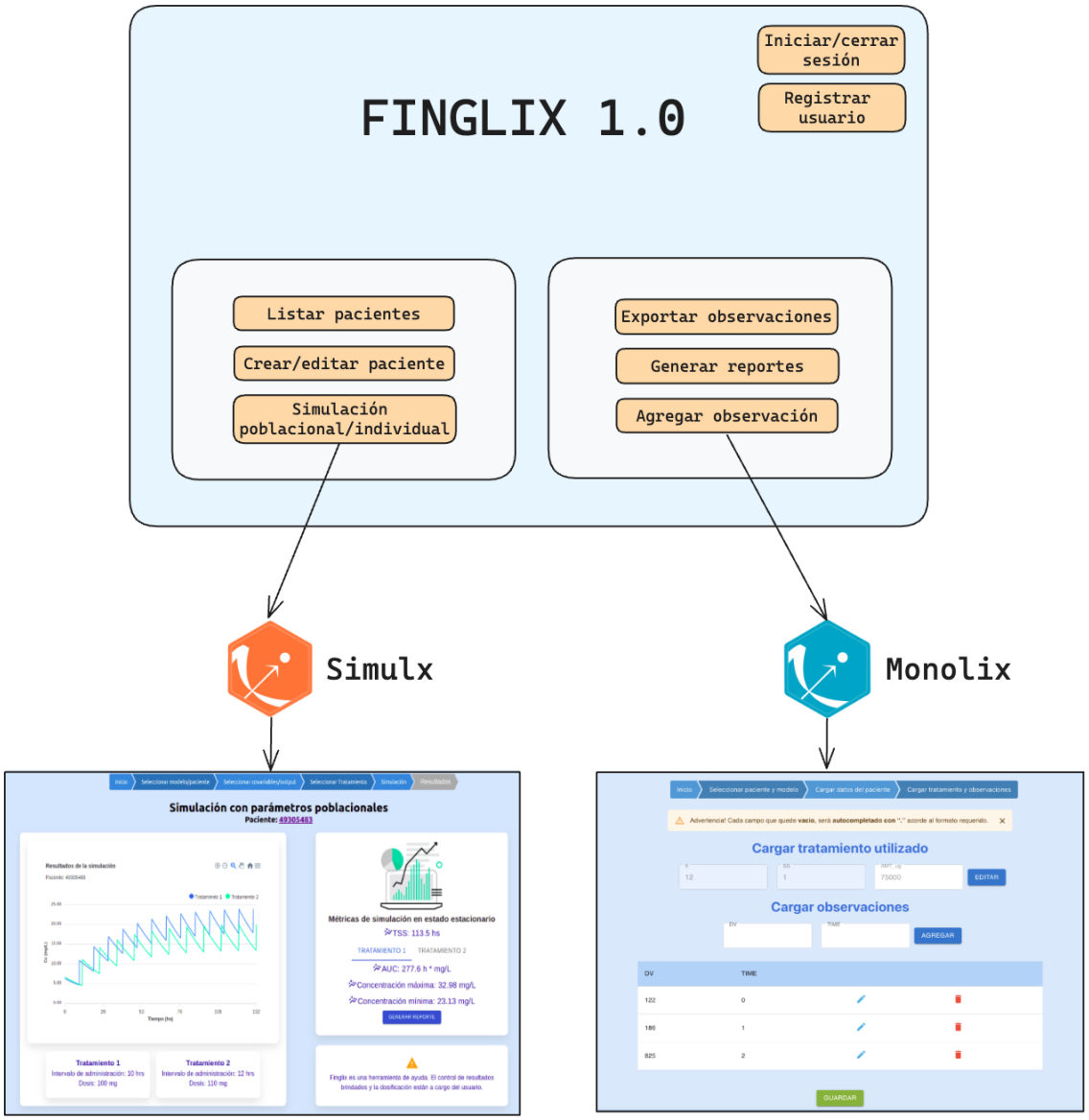


Figura 3.2: Finglix 1.0

3.2.1. Usuarios y pacientes

Finglix brinda una funcionalidad de gestión de usuarios que permite registrar nuevos usuarios y otorgarle distintos roles (Admin, Doctor y Farmacéutico). Cada rol tiene acceso a un conjunto de funcionalidades distinto. Los usuarios deben autenticarse para ingresar al sistema.

A su vez, Finglix provee un conjunto de funcionalidades relacionadas al objeto en estudio, en este caso los pacientes. El sistema permite a los administradores y farmacéuticos crear, editar y listar pacientes.

3.2.2. Cargar observaciones

Finglix permite a un médico o farmacéutico cargar observaciones de un paciente para un modelo en particular. El usuario debe ingresar las covariables del paciente,

el tratamiento utilizado (dosis e intervalo de administración) y las concentraciones observadas del paciente, junto a su tiempo de observación. Luego de la carga de estas observaciones el sistema estima nuevamente los parámetros individuales del paciente utilizando la herramienta Monolix. Esta nueva estimación permite ajustar el modelo a la variabilidad individual del paciente, lo que permite mejorar la precisión de las simulaciones posteriores para el paciente.

Además, todas las observaciones ingresadas de un paciente pueden ser exportadas en un archivo PDF.

3.2.3. Simulaciones

A partir de los datos de un paciente (covariables) y un tratamiento se puede realizar una simulación con parámetros poblacionales con la herramienta Simulx y desplegar una gráfica con los datos obtenidos junto a una serie de métricas como el AUC, concentración máxima y concentración mínima.

De igual modo, previo ingreso de una observación de un paciente, en el cual se estiman los parámetros individuales, puede realizar una simulación individual. Al igual que la simulación con parámetros poblacionales, al ejecutarse se despliega una gráfica con los datos calculados junto a métricas como el AUC, concentración máxima y concentración mínima.

3.2.4. Cargar modelo

Mediante la plataforma de administración se pueden cargar nuevos modelos para un fármaco. Para cargar modelos se deben ingresar datos como la unidad, el compartimiento, entre otros. Luego deben cargarse los parámetros del modelo, que se dividen en tres tipos: observacionales, covariables y de tratamiento. Además, deben cargarse los archivos de los modelos de MonolixSuite, se necesitan tres por modelo: uno para simulación poblacional, otra simulación individual y otro para estimar parámetros. Finalmente es necesario cargar también un conjunto de datos poblacionales, que deben ser cargados en en formato CSV, el orden y significado de las columnas de este archivo dependen del modelo.

Limitaciones:

En base al uso de la herramienta se constató que el proceso de carga de modelos resultaba complejo de utilizar y comprender. En particular, se identificaron las siguientes problemáticas:

1. Los nombres de las variables en los modelos pueden tener nombres no adecuados para el usuario. Por lo tanto, existe otro campo llamado ***Show columns name*** que relaciona variables con el nombre que debe ser mostrado al usuario. Este campo espera un JSON, donde la clave es la variable en el modelo y el valor es el nombre que debe ver el usuario.

Si bien, estos campos están explicados en el manual de usuario, este procedimiento es propenso a errores, requiere entender los modelos y generar un JSON. Además, para poder identificar qué ingresar en cada parámetro, es necesario entender los modelos de MonolixSuite con bastante profundidad.

2. Se necesitan tres archivos de extensión *mlxtran* (que representan modelos en MonolixSuite): uno para cada funcionalidad (cargar observaciones, simulación poblacional y simulación individual). Estos tres archivos no forman parte del flujo natural de estimación y simulación que presenta MonolixSuite, es decir, si el farmacéutico hace este proceso manualmente sin usar Finglix, solo necesita un archivo. En cambio Finglix requiere generar tres archivos diferentes a partir de un único archivo del modelo.

   Este proceso no estaba muy detallado en la documentación, entonces fue necesario obtener ejemplos ya creados para entender cómo debían ser estructurados estos archivos.

3. Una vez ingresado un modelo en Finglix (que incluye *metadata* del modelo, las variables y los archivos), este no se podía editar, porque si se editaba las simulaciones fallaban.

3.3. Herramientas relacionadas

En esta sección se describen las herramientas de simulación que fueron estudiadas durante la etapa de análisis.

3.3.1. Mrgsolve

Mrgsolve[1] es una herramienta que ayuda a crear y simular modelos farmacocinéticos. Esta herramienta es un paquete que se puede instalar y utilizar con el lenguaje R [42]. Utiliza bloques de código organizados para definir diferentes partes del modelo, como parámetros y ecuaciones. Los modelos se encapsulan en objetos que contienen todos los elementos necesarios para realizar simulaciones. También permite trabajar con datos de eventos y de individuos para simular diferentes escenarios de dosificación y respuestas individuales.

Una de las mayores ventajas de Mrgsolve es de código abierto y su ejecución es en memoria. Mrgsolve ha sido diseñado para ser utilizado a través del lenguaje de programación R, lo que permite utilizarlo con herramientas de análisis de datos [26]. Es muy flexible y se puede personalizar fácilmente para crear modelos específicos. También puede manejar simulaciones complejas con múltiples individuos y diferentes escenarios de dosificación. Además, cuenta con ejemplos prácticos. Para usar Mrgsolve, es importante tener conocimientos básicos del lenguaje R.

Ventajas

[1] `https://mrgsolve.org/`

1. ***Software* libre**: es gratuito y de código abierto.
2. **Integración con R**: funciona dentro del entorno R, lo que permite usar otras herramientas de análisis de datos.
3. **Ejecución en memoria**: esto optimiza el rendimiento y velocidad de las simulaciones.
4. **Soporte y ejemplos prácticos**: proporciona ejemplos y documentación que facilitan el aprendizaje y la implementación.

Desventajas

1. **No estima parámetros**: para estimar parámetros es necesario complementar con código adicional, por ejemplo, el paquete Mapbayr (ver Sección 3.3.1.1) realiza estimaciones bayesianas a posteriori.

3.3.1.1. Mapbayest

Debido a que Mrgsolve no realiza estimaciones, se necesita algún paquete adicional que lo realice. Mapbayest es un paquete de R que realizar estimaciones bayesianas a posteriori sobre un individuo utilizando modelos definidos en Mrgsolve. Internamente, computa una función objetivo a partir del modelo y los datos, realiza optimizaciones y retorna los parámetros calculados [26] [48].

3.3.2. Nlmixr2

Nlmixr2[2] es una herramienta de código abierto con licencia GNU. Esta herramienta fue desarrollada para ajustar y simular modelos no lineales de efectos mixtos (NLMEM) en R [49]. Nlmixr2 al igual que Mrgsolve aprovecha el ecosistema de R, permitiendo a los usuarios combinar su funcionalidad con otras herramientas de análisis de datos y documentación.

Nlmixr2 actualmente requiere que los conjuntos de datos sean compatibles con el formato de eventos RxODE, lo que puede requerir un preprocesamiento adicional y ser un obstáculo para la integración directa de ciertos conjuntos de datos. [27].

Ventajas

1. ***Software* libre**: es gratuito y de código abierto.
2. **Integración con el ecosistema de R**: facilita el uso de otros paquetes de análisis de datos, visualización y documentación.
3. **Estima parámetros**: ofrece múltiples algoritmos para estimar parámetros [50], entre ellos, regresión linear tradicional, optimización libre de gradiente, aproximación estocástica de expectativa-maximización.

Desventajas

1. **Requisitos de datos específicos**: los conjuntos de datos deben ser compatibles con el formato de eventos RxODE, lo que puede requerir preprocesamiento adicional.

[2] `https://nlmixr2.org/`

3.3.3. Shiny App

En una de las reuniones con los miembros del CIENFAR uno de los integrantes introdujo una *Shiny App*[3] la cual mostraba la concentración en función del tiempo para un modelo del fármaco Vancomicina. Si bien esta herramienta no se la estudió con el mismo propósito que las anteriores ya que no es una herramienta que permitía integrarse, lo interesante de esta aplicación es que permitía comparar la curva de concentración en función del tiempo generada por el modelo contra la observación cargada del paciente. Esto es, en el gráfico mencionado, se resaltan las observaciones mediante un punto. Si el modelo está prediciendo de forma correcta, se espera que la distancia entre los puntos a la curva que estima el modelo sea pequeña. En Finglix, cuando se cargan datos de observaciones a un paciente, se realiza una reestimación de los parámetros individuales que ajusta el modelo a la observación del paciente cargado. Resulta de interés para el equipo del CIENFAR poder ver cómo queda esta curva en comparación con los datos de observaciones reales, para ver qué tan bien se comporta el modelo. Por lo tanto, la Shiny App inspiró este nuevo requisito.

3.4. Requisitos Finglix 2.0

En esta sección se enumeran y describen los requisitos relevados para abordar en esta nueva versión de la plataforma.

3.4.1. [RQ-01] Integración con múltiples plataformas de simulación y estimación

El sistema actual permite estimar parámetros y realizar simulaciones solamente utilizando la suite de Monolix. Esto presenta algunas limitaciones ya que, como se mencionó previamente, este conjunto de herramientas es de pago.

Por lo tanto, resulta conveniente que la plataforma tenga la flexibilidad de soportar múltiples herramientas de simulación y estimación. Particularmente, la mayor motivación surge de utilizar herramientas de código abierto.

3.4.2. [RQ-02] Mejorar el proceso de carga de modelos

Como se mencionó en la sección 3.2.4, la funcionalidad de carga de modelos presenta algunas limitaciones principalmente en usabilidad y extensibilidad.

Usabilidad: se desea poner el foco en la especificación de variables del modelo, ya que actualmente se dividen en varios campos y cada uno con una forma particular de ingresarlos.

También se busca mejorar la usabilidad mediante la definición del tipo de dato de las variables que se definen en la carga del modelo (enteros, strings, booleanos, arreglos, etc), limitando los posibles valores que puede ingresar el usuario y reduciendo así los posibles errores en la carga de datos. Además, para las variables se requiere poder

[3] `https://tdmx.shinyapps.io/vancomycin/`

asignarles un nombre comprensible para el usuario (por ejemplo, LBW significa masa magra corporal).

Finalmente, en los modelos se suelen utilizar enteros para representar datos como el sexo. Entonces, resulta de utilidad poder asociar esos valores a otros que el usuario conozca, por ejemplo, si el usuario selecciona sexo femenino, que el sistema lo asocie al valor "1".

Extensibilidad: se necesita que la funcionalidad permita el soporte a distintos modelos de diferentes fármacos y no a uno en concreto como es el caso del modelo de la Ciclosporina (modelo con el que se probó Finglix 1.0). Además, se necesita que se pueda interactuar con múltiples herramientas de simulación y estimación, y no con una en concreto.

3.4.3. [RQ-03] Visualizar precisión en las predicciones

Actualmente, cuando el farmacéutico carga una nueva observación simplemente se reestiman los parámetros individuales del paciente y se retorna al usuario al inicio de Finglix.

Se requiere que luego de la carga de la observación, se tenga la posibilidad de comparar la simulación individual (estimando nuevamente los parámetros individuales) respecto a la observación recién cargada. Esto permite al farmacéutico ver si sus modelos están siendo precisos o si la observación se comporta muy distinto a lo esperado.

3.4.4. [RQ-04] Perfiles de simulación

Algunas herramientas cuando estiman nuevos parámetros individuales, pueden retornar conjuntos de parámetros individuales adicionales aparte del óptimo, que bajo ciertos escenarios pueden tener una mejor precisión. Realizar simulaciones con estos parámetros se lo denomina perfiles de simulación. MonolixSuite es una de las herramientas que soportan esta funcionalidad.

Entonces, si la herramienta de simulación lo permite y el usuario lo desea, se quiere contrastar las observaciones ingresadas respecto a la simulación individual, la simulación poblacional y los perfiles de simulación.

3.4.5. [RQ-05] Simulación objetivo

Actualmente, cuando el doctor o farmacéutico se encuentra estudiando cuál es el mejor tratamiento para un paciente (dosis e intervalo de administración), debe primero determinar el tratamiento y luego realizar simulaciones para predecir cómo el paciente responderá. Por lo tanto, interesa lograr el procedimiento inverso, es decir, dado un valor objetivo en determinada medición, se desea recomendar al usuario cuál es el tratamiento que mejor aproxima al objetivo marcado. Algunos objetivos posibles son: AUC(X) y concentración mínima.

Es importante tener en cuenta que se debe discretizar tanto el intervalo de administración como la dosis. Imaginemos que como resultado se obtiene que el tratamiento ideal es suministrar una dosis de 0.81mg cada 11 horas 24 minutos y 18 segundos. Los fármacos generalmente vienen en comprimidos y a lo sumo el comprimido se puede partir a la mitad, para asegurar una precisión aceptable. Si en el ejemplo dado, el fármaco se presenta como unidad mínima en comprimidos de 0.5mg, a lo sumo se podría asegurar con cierta precisión que el paciente ingiera 0.75mg (administrándole un comprimido y medio). Similarmente sucede con el intervalo de administración, es inviable que un médico o enfermero logre suministrar a un paciente una dosis con tanta exactitud. Más viable es al menos asegurar que el intervalo sea medido en horas. Incluso, en la práctica se utiliza un conjunto mucho más finito, en múltiplo de 4 horas (cada 4, 8, 12, 16, 20 o 24 horas).

3.4.6. [RQ-06] Mostrar nuevos datos en las simulaciones

Se debe mostrar al usuario con qué fármaco, modelo y herramienta de simulación se realizó la simulación.

Otro dato que aporta valiosa información al farmacéutico luego de una simulación es el AUC. Actualmente el AUC se calcula como el área bajo la curva dentro de la gráfica de simulación tomando como intervalo la duración del intervalo de administración del tratamiento.

Sin embargo, en algunos fármacos, se suele estudiar a los pacientes para un intervalo fijo, que no dependa del tratamiento. Por ejemplo, la Ciclosporina se estudia en base al AUC en 24 horas. Por lo tanto, es necesario, a partir de un intervalo fijo indicado para el fármaco, que se aporten ambos datos luego de las simulaciones (el AUC que depende del intervalo del tratamiento y el AUC X que es fijo para el fármaco).

3.4.7. [RQ-07] Soporte a modelos que utilizan regresores

Se requiere soportar modelos de MonolixSuite que utilicen regresores. Los regresores son un tipo de variable del modelo que tienen un comportamiento particular.

MonolixSuite trata a los regresores como una función x que depende del tiempo y que no se encuentra en el modelo, comúnmente se define en un archivo auxiliar y luego es utilizada dentro del modelo. Si la función no se puede computar para todos los valores t_i, entonces Monolix realiza una interpolación.

3.4.8. [RQ-08] Flexibilizar ingreso de posologías

Actualmente, solo se permite indicar un tratamiento con un intervalo de tiempo de administración de dosis fijo. Pero en la práctica, existen casos en los cuales el tiempo de administración entre dosis varía. Por ejemplo, que se intercalen dosis cada 8 y 12 horas.

También, existen fármacos que se suministran por vía intravenosa y de manera constante, donde el intervalo de tiempo es continuo.

3.4.9. [RQ-09] *Model averaging y model selection*

Un enfoque basado en un único modelo podría dar lugar a recomendaciones de dosis inadecuadas, lo que conduciría a predicciones subóptimas.

Por esta razón, existen dos tipos de algoritmos que utilizan múltiples modelos para alcanzar una predicción más precisa. *Multimodel selection algorithm (MSA)* y *multimodel average algorithm (MAA)*. MSA es la predicción del modelo candidato con la ponderación más alta, mientras que MAA es el promedio de predicciones de todos los modelos candidatos. En la Figura 3.3 se describe con un diagrama ambos algoritmos.

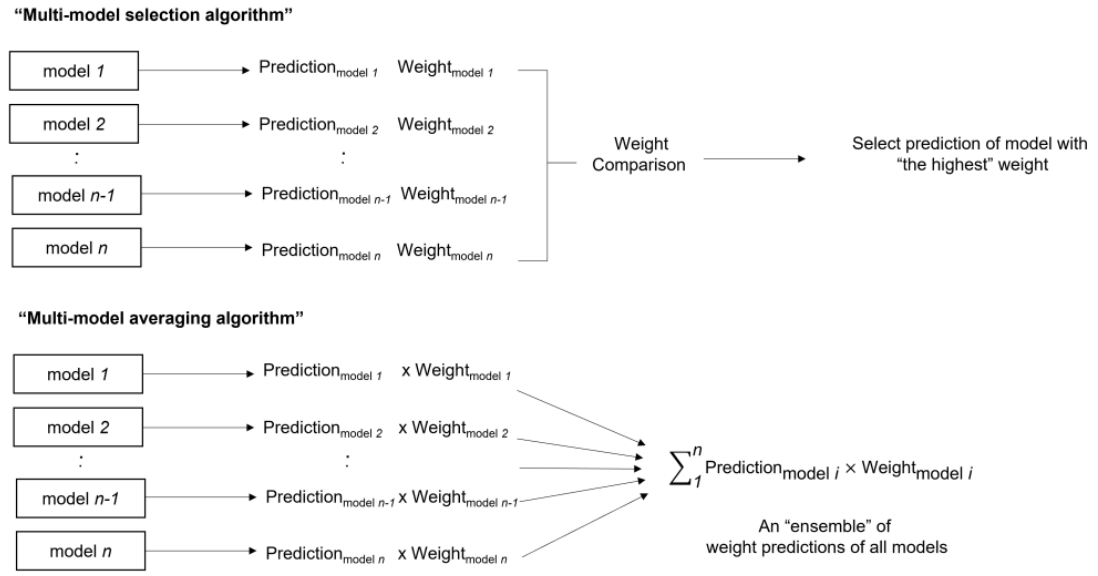


Figura 3.3: Comparación entre MSA y MAA. Figura extraída de [1]

Un ejemplo de un algoritmo MAA es: ***Bayesian multi-model average (BMA)*** [51], en el cual el peso de cada modelo es calculado a partir de la probabilidad a posteriori del modelo. Esta probabilidad se calcula de la siguiente manera:

$$P(M_j \mid D) = \frac{P(D \mid M_j) \cdot P(M_j)}{\sum_{k=1}^{K} P(D \mid M_k) \cdot P(M_k)}$$

Donde:

- $P(M_j \mid D)$ es la probabilidad a posteriori del modelo M_j dado el conjunto de datos D.
- $P(D \mid M_j)$ es la probabilidad de que el modelo M_j explique correctamente a los datos observados.
- $P(M_j)$ es la probabilidad a posteriori del modelo M_j
- El denominador $\sum_{k=1}^{K} P(D \mid M_k) \cdot P(M_k)$ es una constante que asegura que la suma de todas las probabilidades a posteriori de todos los modelos candidatos dé 1.

3.4.10. [RQ-10] Compartimentalización del acceso al software por servicio médico

Permitir que los pacientes y modelos puedan ser compartimentados por servicio médico, es decir que estos datos sean independientes. Por ejemplo, que el servicio de Nefrología no pueda visualizar los pacientes de Hematología.

3.4.11. [RQ-11] Integración con historia clínica:

El sistema debe poder integrarse con sistemas de historia clínica electrónica, lo que facilita el intercambio de información y la interoperabilidad con otros sistemas de salud utilizados.

3.5. Requisitos no funcionales Finglix 2.0

En esta sección, se presentan los principales requisitos no funcionales identificados. Estos requisitos apuntan a que el sistema cumpla con los atributos de calidad y el comportamiento del sistema esperado.

- **Interfaz gráfica y usabilidad:**
  El sistema debe aportar una interfaz gráfica para que los usuarios interactúen con el sistema. Se espera que esta interfaz sea fácil de usar.
- **Extensibilidad:**
  El sistema debe poseer la propiedad de ser extensible, tanto para la integración con múltiples herramientas de simulación y estimación como con la posibilidad de ingresar modelos de múltiples fármacos.

Luego de identificar los atributos de calidad fundamentales como la usabilidad y extensibilidad, es crucial también considerar otros requisitos no funcionales más específicos que impactarán directamente en la implementación y operación del sistema.

- **Soporte para *software* de simulación de código abierto:** Implica que el sistema debe soportar al menos un *software* de simulación de código abierto para la realización de simulaciones y estimaciones. El soporte con software de código abierto facilita la colaboración y la transparencia, permitiendo a los usuarios adaptar y mejorar las herramientas según sus necesidades específicas.
- **Despliegue:** El sistema debe ser desplegado en una infraestructura que cumpla con las normas de seguridad de los datos vigentes en Uruguay. Esto implica asegurar que todos los datos almacenados y procesados por el sistema estén protegidos contra accesos no autorizados, pérdidas y brechas de seguridad. Además, el despliegue del sistema debe ser realizado sin costo, esto incluye tanto los recursos físicos como servidores y redes, como los servicios necesarios para su operación continua y segura.

3.6. Definición del alcance

Para la definición del alcance se comenzó por la recolección de los requisitos. Para ello, se revisó el estado del proyecto anterior y se organizaron múltiples reuniones con el equipo de CIENFAR, lo que permitió una mayor comprensión de las necesidades y expectativas por parte de ellos. Estos requisitos fueron posteriormente evaluados en términos de dificultad, incertidumbre, impacto en la arquitectura del sistema, tiempo necesario para su implementación, y su aporte al proyecto de grado.

A partir de estas evaluaciones, se asignaron puntuaciones a cada requisito detallados en el Cuadro 3.1, lo que facilitó la priorización de los elementos que compondrían la propuesta de alcance inicial. Este enfoque permitió generar un plan inicial que reflejaba tanto los objetivos del proyecto como las restricciones y oportunidades identificadas.

A medida que el proyecto avanzaba, el alcance se ajustó dinámicamente en respuesta a nuevos conocimientos y desafíos que surgieron. Si bien se logró cumplir con la planificación inicial, el alcance se fue modificando conforme se obtenía más información y se realizaban pruebas. Este enfoque flexible permitió una evolución constante del alcance, con el objetivo de que el proyecto se mantuviera alineado con las necesidades y objetivos del equipo de farmacología.

Código	**Requisito**	**Incertidumbre**	**Dificultad**	**Aporte al proyecto**	**Impacto en arquitectura**	**Tiempo de implementación**
RQ-01	Integración con múltiples plataformas de simulación y estimación	10	10	10	10	8 semanas
RQ-02	Mejorar el proceso de carga de modelos.	10	10	6	8	4 semanas
RQ-03	Visualizar precisión en las predicciones.	3	6	8	4	2 semanas
RQ-04	Perfiles de simulación.	5	2	5	5	1 semanas
RQ-05	Simulación objetivo.	6	8	7	4	4 semanas
RQ-06	Cargar nuevos datos en las simulaciones.	3	5	7	3	1 semanas
RQ-07	Soporte a modelos que utilizan regresores.	9	7	5	3	2 semanas
RQ-08	Flexibilizar ingreso de posologías.	7	7	5	6	3 semanas
RQ-09	*Model averaging.*	6	6	5	7	3 semanas
RQ-10	Compartimentalización del acceso al software por servicio médico.	1	1	3	3	1 semanas
RQ-11	Integración con historia clínica.	10	8	4	4	2 semanas

Cuadro 3.1: Requisitos y sus valoraciones para Finglix 2.0

A continuación, se explica brevemente el significado de cada categoría utilizada en la tabla 3.1.

- Incertidumbre: Mide cuán claro o definido está el requisito.
- Dificultad: Estima el nivel de complejidad técnica para implementar el requisito.
- Aporte al proyecto: Representa el valor o beneficio que el requisito aporta tanto para el equipo de CIENFAR, como para este proyecto de grado.
- Impacto en arquitectura: Describe cómo el requisito afecta la estructura del sistema. Un alto impacto implica que el requisito podría requerir cambios importantes en la arquitectura existente o en las decisiones de diseño.
- Tiempo de implementación: Es la estimación del tiempo, medido en semanas, necesario para completar el requisito.

Primero, se estimó la incertidumbre, dificultad e impacto en la arquitectura de cada requisito. Y a partir de esto se estimó el tiempo de implementación. Finalmente, se definió el aporte al proyecto de cada requisito.

Se ordenaron los requisitos de mayor a menor por aporte al proyecto y luego se seleccionaron todos los requisitos posibles a realizar en aproximadamente veinte semanas de implementación. Como resultado se definieron las siguientes funcionalidades a implementar:

- RQ-01 Integración con múltiples plataformas de simulación y estimación
- RQ-02 Mejorar el proceso de carga de modelos.
- RQ-03 Visualizar precisión en las predicciones.
- RQ-04 Perfiles de simulación.
- RQ-05 Simulación objetivo.
- RQ-06 Cargar nuevos datos en las simulaciones.
- RQ-07 Soporte a modelos que utilizan regresores.

Adicionalmente relacionado al RQ-01, se planteó la posibilidad de abstraer un modelo de la herramienta, esto es, si un modelo como los que se tiene de la Ciclosporina podía utilizarse directamente con otras herramientas aparte de MonolixSuite. Esto no es posible con la forma convencional de ejecutar modelos que tienen las herramientas estudiadas. Algunas difieren ampliamente en la forma de definir y ejecutar los modelos. Por ejemplo, MonolixSuite utiliza su propia extensión de archivo de un modelo y una estructura muy particular, lo que resulta muy complejo adaptar a cualquiera de las otras opciones. Sin embargo, existe la fundación Ddmore[4] que trabaja en lograr un estándar en la definición de modelos. Si bien todavía las herramientas utilizadas no lo han adoptado, quizás en un futuro todos sigan este estándar. Por lo tanto quedó por fuera del alcance de esta tesis separar los modelos de la herramienta de simulación.

[4] `https://www.ddmore.foundation`

Capítulo 4

Diseño de la solución

En este capítulo se describe la descripción general de la solución y la evolución de la arquitectura respecto a Finglix 1.0. Además, se explicita el modelo de dominio y diagramas de componentes/despliegue. Finalmente, se detalla la solución de las funcionalidades que tuvieron mayor impacto en el desarrollo de la solución y arquitectura del sistema.

4.1. Descripción General

Como se puede observar en la Figura 4.1, Finglix 2.0 cuenta con todas las funcionalidades que incluía la versión anterior (en amarillo), algunas de ellas incluyen ajustes y mejoras (en azul), agrega nuevas funcionalidades (en verde) y también da soporte a otras herramientas de simulación, desacoplándose de la herramienta MonolixSuite. Este soporte dinámico y no limitado a una única plataforma es posible ya que Finglix 2.0 permite cargar plugins, estos plugins son programables y realizan una interfaz de operaciones que el sistema luego invoca para llevar a cabo los distintos casos de uso.

Además, en la Figura 4.1 se puede apreciar la arquitectura del sistema, que está compuesta por un *frontend* que incluye un *frontoffice* y un *backoffice*. El primero es la interfaz que el usuario final utiliza (por ejemplo, los médicos), el segundo es utilizado por el administrador del sistema y proporciona una interfaz para gestionar algunos recursos, por ejemplo, la carga de plugins y modelos. Por otra parte, el *backend* incluye una API REST y un *message broker*, este ultimo se utiliza para gestionar colas de mensajes que luego un *worker* procesa. Tanto los *workers* como la API interactúan con el sistema de archivos del sistema operativo y la base de datos.

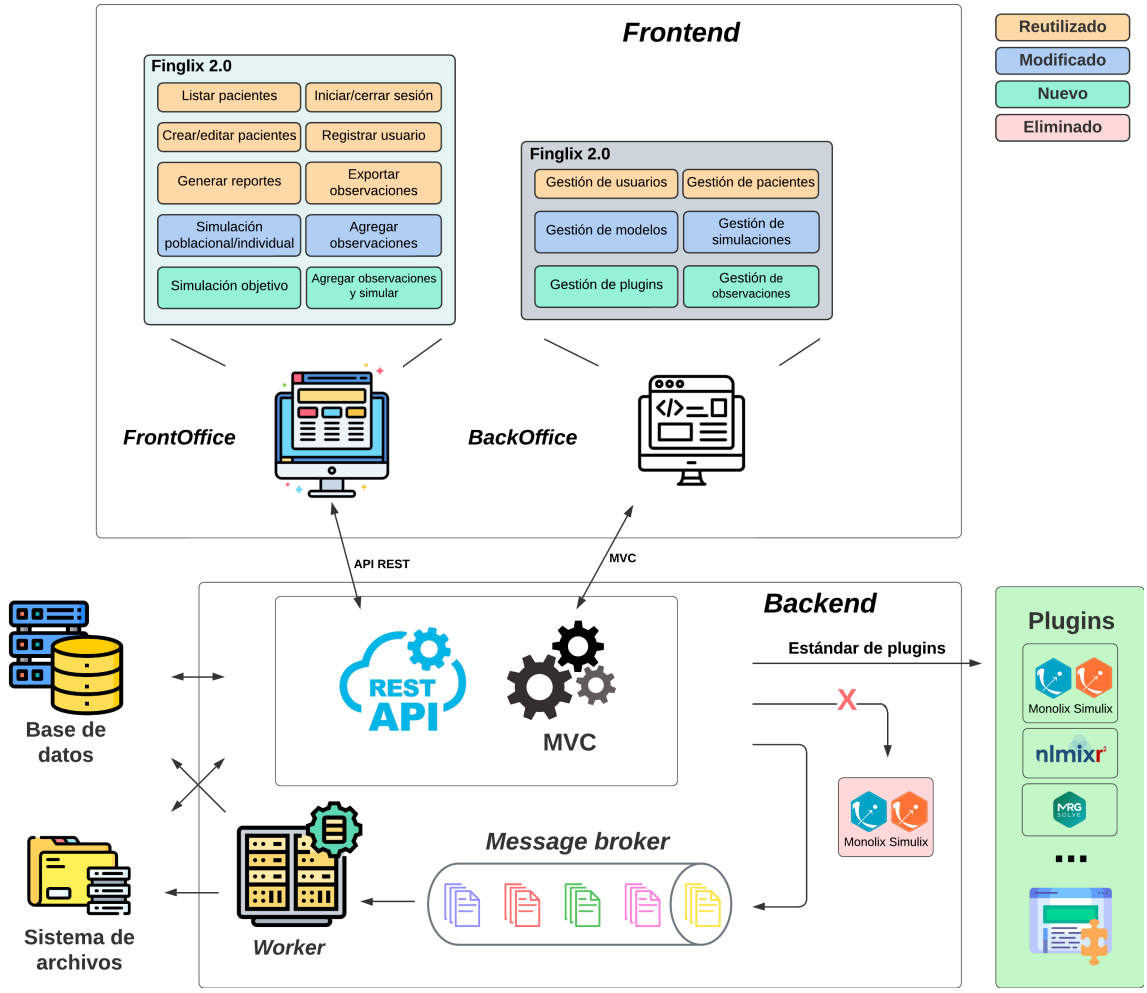


Figura 4.1: Finglix 2.0

En Finglix 2.0, en el *backend*, la API está diseñada para utilizar *plugins* que se cargan dinámicamente en el sistema, cada uno con su propia estructura de carpetas, proporcionando funcionalidades como la realización de simulaciones, la estimación de parámetros, entre otras capacidades que se desarrollarán en las siguientes secciones.

El *frontoffice* incluye funcionalidades como el inicio y cierre de sesión, registro de usuarios, la capacidad de gestionar pacientes por medio de un listado, registro y edición de pacientes a los que se le realizan las simulaciones. Además, el *frontoffice* cuenta con funcionalidades para realizar simulaciones tanto poblacionales como individuales, junto con la nueva funcionalidad de Finglix 2.0 denominada simulación objetivo. El sistema permite la exportación de observaciones y la generación de reportes. Para las funcionalidades de simulación y carga de observaciones se utiliza dinámicamente la ejecución de los plugins cargados previamente en el sistema.

En el Cuadro 4.1 se detalla qué funcionalidades, tanto a nivel de *frontend* como *backend*, fueron reutilizadas, modificadas o nuevas con respecto a Finglix 1.0.

Se define como "reutilizado" cuando no se requirió realizar cambios o solo ajustes menores. Se define como "modificado" cuando la funcionalidad ya existía en el sistema y se utilizó como base para la nueva versión de Finglix 2.0, finalmente se define

Funcionalidad	***Frontend***	***Backend***
Iniciar y cerrar sesión	Reutilizado	Reutilizado
Listar pacientes	Reutilizado	Reutilizado
Crear y editar pacientes	Reutilizado	Reutilizado
Simulaciones poblacionales	Modificado	Nuevo
Simulaciones individuales	Modificado	Nuevo
Simulación objetivo	Nuevo	Nuevo
Exportar observaciones	Reutilizado	Nuevo
Generar reportes	Reutilizado	Reutilizado
Cargar observaciones	Nuevo	Nuevo
Cargar modelos	Reutilizado	Nuevo
Registrar usuario	Reutilizado	Reutilizado
Cargar plugins	Nuevo	Nuevo

Cuadro 4.1: Reutilización de las funcionalidades en Finglix 2.0

como "nuevo" cuando la funcionalidad no existía en Finglix 1.0.

Además, el soporte a regresores, relevado en la Sección 3.4.7, no forma parte de esta tabla ya que no es una funcionalidad en sí, sino que es una mejora que involucra todos los tipos de simulaciones, cargar observaciones y cargar modelos. El soporte a regresores, desde el punto de vista de la solución, requirió agregar el tipo regresor a los tipos de parámetros de un modelo. En la Sección 5.4.4 se describe su implementación.

4.2. Arquitectura de la Solución

La arquitectura de Finglix 2.0 y la arquitectura de Finglix 1.0 son bastante similares en muchos aspectos, ya que se mantiene la mayor parte de su estructura.

El mayor cambio fue agregar una capa de abstracción. En lugar de acoplarse el *backend* directamente con MonolixSuite, se optó por un diseño que abstrae los modelos y las herramientas utilizadas. Esto llevó a la introducción de un sistema de plugins, proporcionando flexibilidad para integrarse con diversas herramientas de simulación y estimación.

La arquitectura diseñada para el sistema permite desplegar los diferentes componentes de manera independiente. Es posible separar el *frontoffice* del *backoffice* y del *backend*, lo que facilita escalabilidad y mantenimiento. La base de datos también puede ser desplegada de forma separada y además los plugins están diseñados de tal forma que pueden ejecutarse tanto en el *backend* como parcialmente en servidores de terceros. Esto último se logra programando un plugin de tal forma que sea un intermediario entre Finglix y un servidor externo que se encarga de realizar las simulaciones, enviando los datos para realizar la simulación y luego retornando los resultados obtenidos a Finglix.

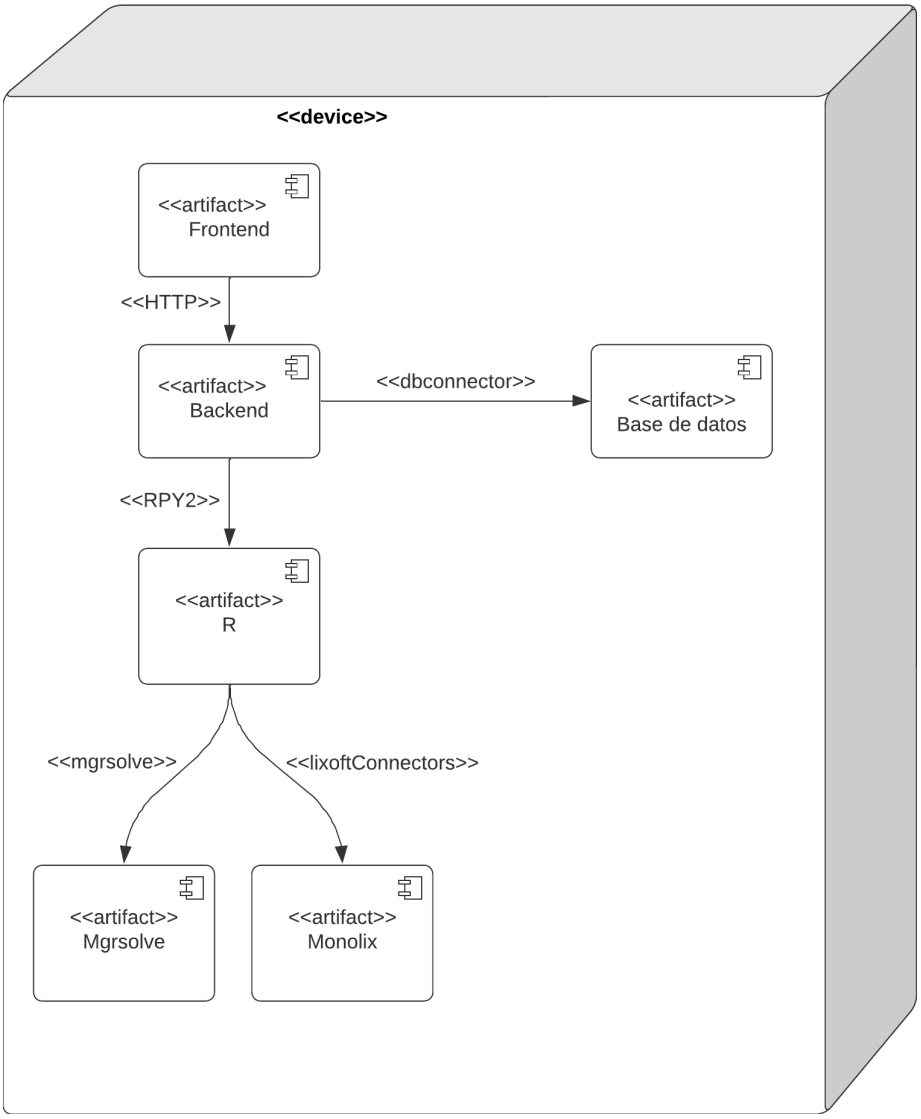


Figura 4.2: Diagrama de despliegue/componentes V1

En las Figuras 4.2 y 4.3 se observan dos estrategias posibles de despliegue, siendo la primera la que fue utilizada en este proyecto. La segunda, muestra la posibilidad de escalabilidad del sistema, ya que se tiene el *frontend*, *backend* y base de datos en distintos ambientes, y a su vez cada plugin puede potencialmente ejecutar en un ambiente totalmente separado al *backend*.

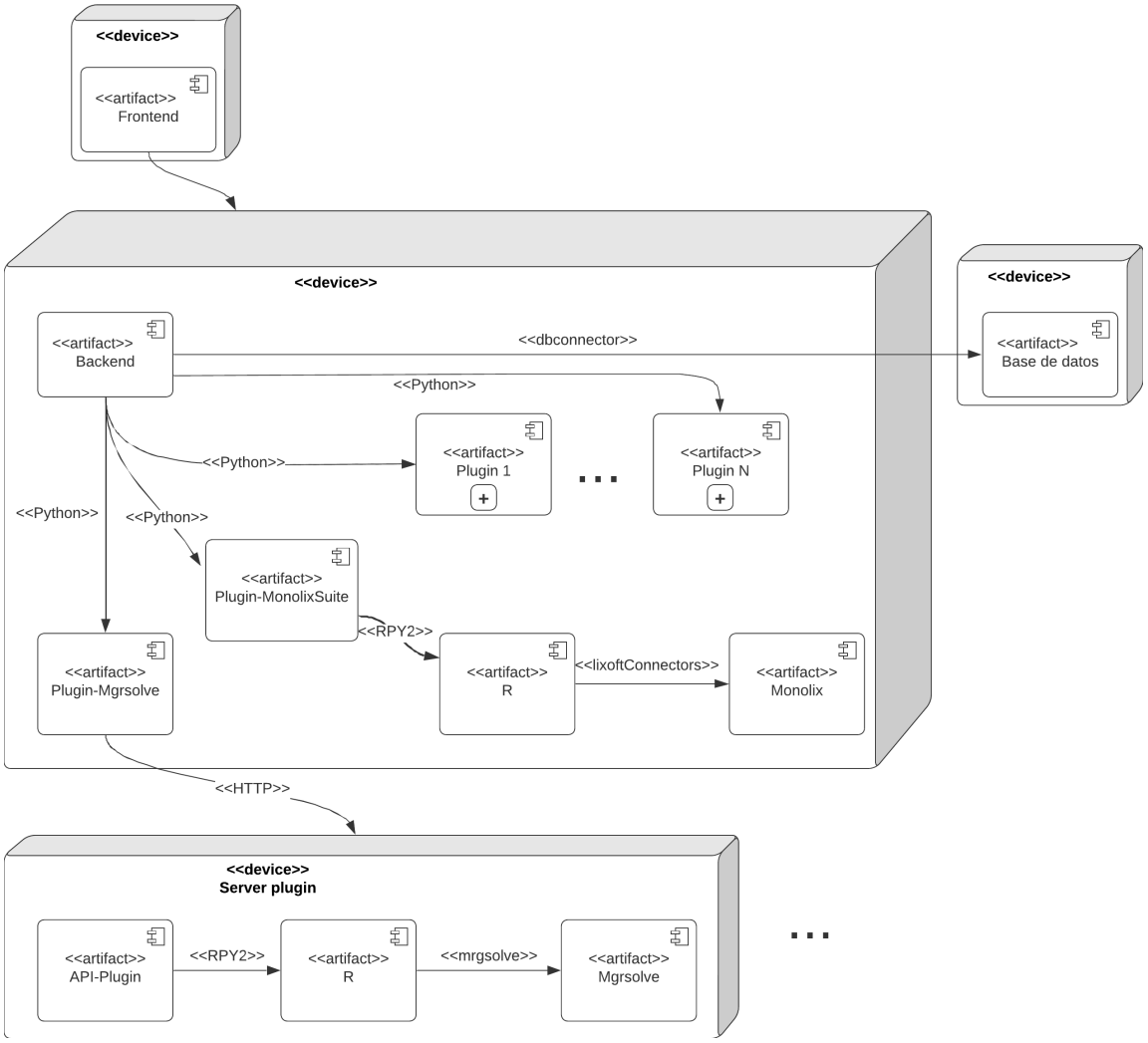


Figura 4.3: Diagrama de despliegue/componentes V2

4.3. Modelo de dominio

En esta sección se detalla el modelo de dominio de Finglix 2.0 y se comentan los cambios realizados con respecto al utilizado en Finglix 1.0. La Figura 4.4 presenta el modelo de dominio organizado en secciones.

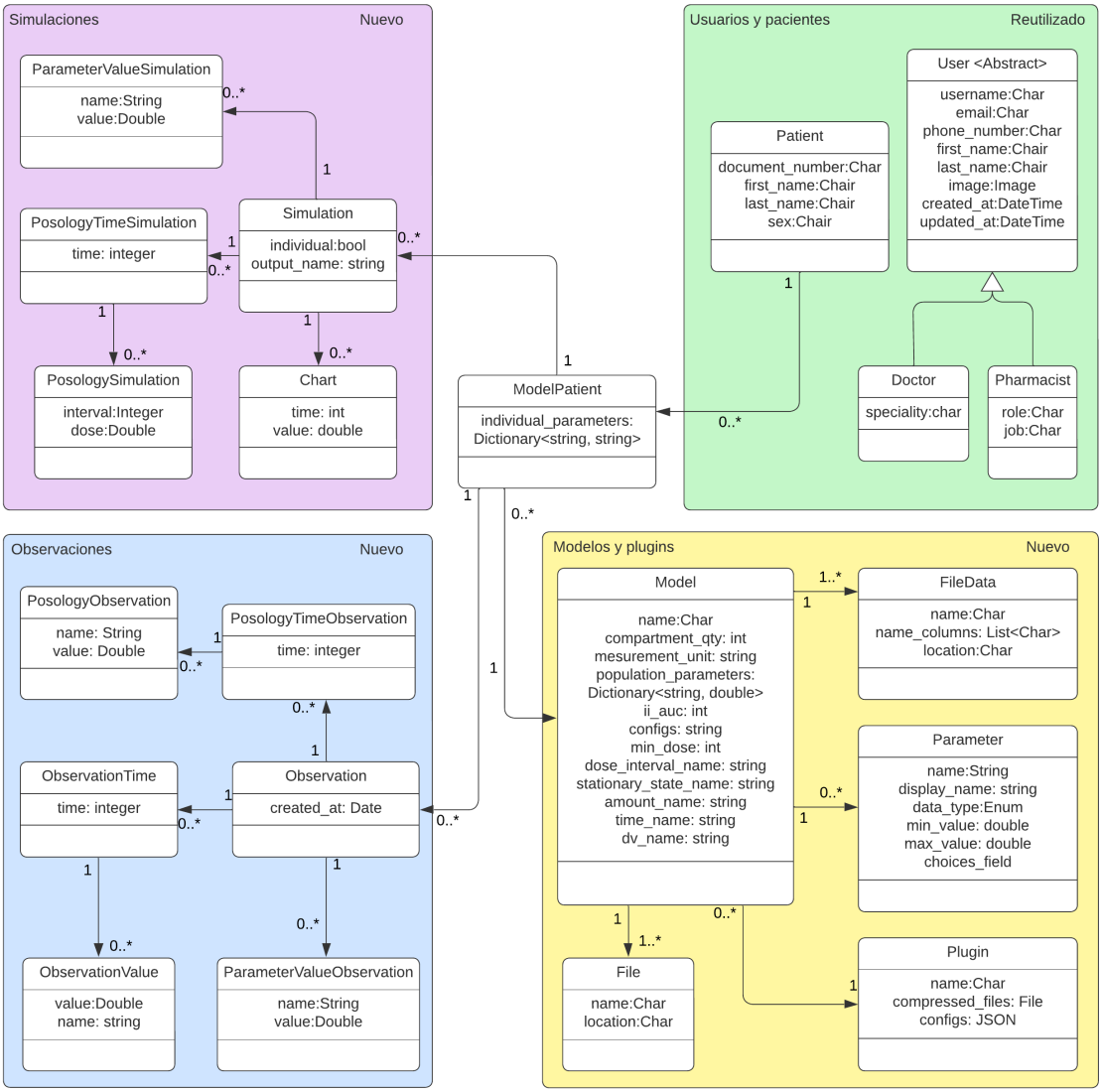


Figura 4.4: Modelo de dominio

En verde, se aprecia el concepto *User* que representa a los usuarios del sistema, quienes pueden ser médicos o farmacéuticos, cada uno con sus roles y especialidades definidas. Este concepto no fue alterado en esta nueva versión de Finglix. Otro concepto que permanece sin cambios en esta nueva versión es *Patient*.

En amarillo, se encuentra el concepto Plugin, que representa un componente de software que se carga dinámicamente en el sistema y tiene la capacidad de interactuar con herramientas de simulación. Cada plugin cuenta con configuraciones específicas. También posee un nombre y los archivos necesarios para interactuar con dichas herramientas. Luego, se encuentra el concepto de *Model*, que representa los modelos

farmacocinéticos utilizados en el sistema. Cada *Model* se relaciona con una serie de *Files* que son archivos empleados para ejecutar las simulaciones. *FileData* es otro tipo de archivo que contiene los datos poblacionales que el plugin utilizará para realizar las simulaciones. Además, cada modelo incluye una serie de parámetros específicos.

Además, el concepto *ModelPatient* se encuentra relacionado al concepto *Observation*, que representa las observaciones del paciente para un modelo específico y se aprecia en al sección azul. Cada observación se compone de una posología, que debe conformarse a los parámetros del modelo correspondiente, y una serie de parámetros y observaciones que también deben estar alineados con los parámetros del modelo. Los parámetros del modelo tienen un nombre cuyo objetivo es mostrarse en la interfaz de usuario, un tipo de dato, valores máximos y mínimos permitidos, y opcionalmente, valores predefinidos. Los modelos también incluyen datos poblacionales utilizados para las simulaciones, configuraciones que el plugin puede utilizar, y parámetros específicos del modelo como nombres de variables necesarias (como *amount*, *time*, *dv*, "estado estacionario"), la unidad de dosificación, el número de compartimentos del modelo, y el nombre del modelo.

De manera similar (en violeta), se encuentra el concepto de *Simulation*, que también está relacionada con el concepto de *ModelPatient*. Esta último se divide en dos tipos, simulaciones individuales y simulaciones poblacionales. También, modela la habilidad de poder relacionar estas simulaciones a una o varias posologías y varios parámetros, que modelan las entradas necesarias para poder realizar las simulaciones. Finalmente, el resultado de la simulación se modela a través del concepto *Simulation*, ya que se espera que el resultado de una simulación sea tal que pueda ser graficado. Comúnmente, se grafica un valor en función del tiempo, y ese valor puede estar asociado a distintos factores (por ejemplo es la concentración en sangre de un fármaco).

Las principales diferencias del modelo de dominio de Finglix 2.0 con respecto a la versión anterior son:

- **Incorporación de concepto de plugins:** La idea general es que el sistema solicita a los plugins que realicen simulaciones, estimen parámetros y carguen observaciones entre otras operaciones, lo que proporciona flexibilidad y permite adaptarse a futuras necesidades. Esta estructura no solo mejora la modularidad del sistema, sino que también facilita la integración con diversas herramientas de simulación y modelos.

- **Representación de modelos farmacocinéticos:** En Finglix 1.0, el concepto *ModelDrug* representaba modelos farmacocinéticos en su completitud, incluyendo el nombre del modelo, archivos para simulaciones, variables, salidas, CSVs, observaciones, parámetros poblacionales, y compartimientos. Por otro lado, *ModelDrugPatient* relacionaba los pacientes con estos modelos, poseía los parámetros individuales y las últimas observaciones de simulación. Aunque ambos conceptos cumplían el objetivo de realizar simulaciones, en Finglix 2.0 se han eliminado y desglosado en conceptos más pequeños para mejorar la flexibilidad. La nueva estructura permite soportar diferentes modelos y plugins, lo que amplía la capacidad del sistema para trabajar con múltiples herramientas de simulación.

4.4. Solución de plugins

El componente de *software* encargado de interactuar con MonolixSuite, que antes se consideraba simplemente como parte del sistema, ahora se maneja mediante plugins como una entrada externa. El sistema solicita a estos plugins que realicen simulaciones, estimen parámetros y carguen observaciones entre otras operaciones, lo que proporciona flexibilidad y permite adaptarse a futuras necesidades. Esta estructura no solo mejora la modularidad del sistema, sino que también facilita la integración con diversas herramientas de simulación y modelos, facilitando que el sistema pueda utilizar nuevas herramientas de simulación a partir de la carga de plugins.

Para la incorporación de este sistema de plugins a Finglix 2.0 fue necesario definir una interfaz de operaciones que todo plugin que desee cargarse a la plataforma debe cumplir. En esta sección se describe esta interfaz y cómo el sistema interactúan con los plugins a través de ella.

4.4.1. Interfaz de Plugins

Las operaciones de la interfaz de plugins son:

InitModel(model): Esta función inicializa un modelo para ser utilizado. Recibe un parámetro model (DataObject con la estructura definida) y retorna los parámetros poblacionales. A esta función el sistema la invoca únicamente cuando se carga o edita el modelo.

FileDatasRequired(): Esta función retorna un vector que contiene los nombres válidos para los archivos de datos de un modelo. Un ejemplo de archivo de datos podría ser un CSV que contenga valores poblacionales. El sistema invoca esta función cuando se carga o edita un modelo, dado que a través de ésta el sistema corrobora que los datos ingresados durante la carga o edición sean compatibles con lo que el plugin espera para un modelo.

FilesRequired(): Esta función retorna un vector de los nombres válidos para los archivos que se esperan en la carga de un modelo. Al igual que *FileDatasRequired*, el sistema invoca esta función cuando se carga o edita un modelo.

GetDependencies(): Esta función retorna un vector con las dependencias que se deben de instalar para el correcto funcionamiento del plugin. El sistema instala estas dependencias al momento de cargar el plugin automáticamente.

DeleteModel(model): Esta función realiza el proceso de borrar un modelo cargado previamente en el plugin. El sistema invoca esta función cuando se elimina un modelo desde el *backoffice*.

DeleteModelPatient(model, patient): Esta función realiza el proceso de borrar los datos del paciente que el modelo pueda tener asociados.

LoadConfig(config_name, value): Carga una variable de configuración (clave,

valor) en la instancia del plugin. Estas variables de configuración son cargadas en el *backoffice* en la sección "configs" del apartado plugins.

LoadModel(model): Esta función carga un modelo en la instancia del plugin. Para ello el sistema le pasa el modelo cargado previamente y se espera que el plugin inicialice las variables que corresponda.

LoadParameter(name, type_variable, datatype, value): Esta función carga un parámetro en la instancia de plugin.

LoadPosology(posology_occasion, posology_wait, posology_value): Esta función carga un tratamiento en la instancia de plugin.

LoadPatient(patient): Esta función carga un paciente en la instancia del plugin.

LoadObservations(observations): Esta función carga las observaciones en la instancia del plugin.

CalculateIndividualParameters(): Esta función calcula y retorna los parámetros individuales de un paciente luego de haber ingresado observaciones del mismo. Luego, el sistema guarda los parámetros para el paciente-modelo.

LoadTreatment(cycle_duration, dosis): Esta función carga un tratamiento que se utiliza para las simulaciones individuales y poblacionales.

ExecutePopulation(population_parameters, tss): Esta función ejecuta una simulación poblacional y retorna un vector de tratamientos.

ExecuteIndividual(individual_parameters, tss, _aux_results): Esta función ejecuta una simulación individual y retorna un vector de tratamientos.

4.4.2. Interacciones del sistema con los plugins

En esta sección se especifican los diagramas de secuencia de cargar plugins y también las funcionalidades que se vieron afectadas por la nueva capa de abstracción, como por ejemplo, cargar modelos, simular y cargar observaciones.

En la Figura 4.5 se presenta el diagrama de secuencia correspondiente al proceso de carga de un plugin en el sistema, que consiste en cargar el archivo comprimido del plugin junto a sus datos de configuración y nombre.

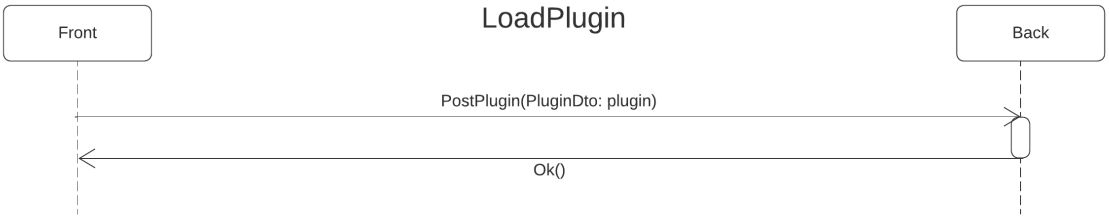


Figura 4.5: Diagrama de secuencia con la interacción para cargar un plugin

La Figura 4.6 detalla el proceso de carga de un modelo. Cuando se carga un modelo en el sistema, a partir del plugin seleccionado, el sistema solicita al plugin los datos y archivos requeridos.

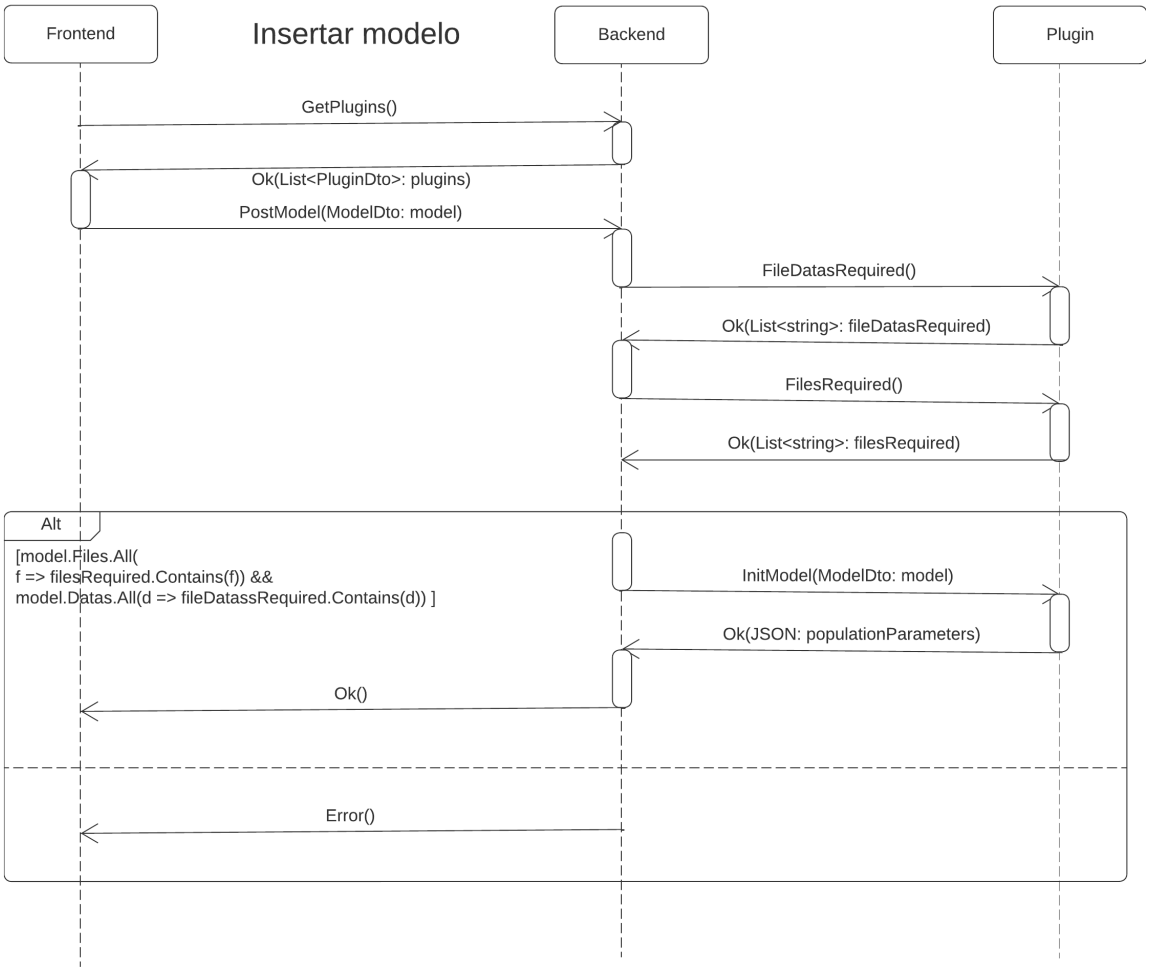


Figura 4.6: Diagrama de secuencia de interacción con plugins para cargar modelos

La realización de simulaciones se especifica en la Figura 4.7, en donde se detalla la comunicación del sistema con un plugin al realizar una simulación. Inicialmente se carga el modelo en el plugin, se cargan los parámetros, y los tratamientos a simular. Posteriormente, si es una simulación poblacional se realiza la simulación. En el caso de realizar una simulación individual se carga el paciente, las observaciones y finalmente se ejecuta la simulación individual.

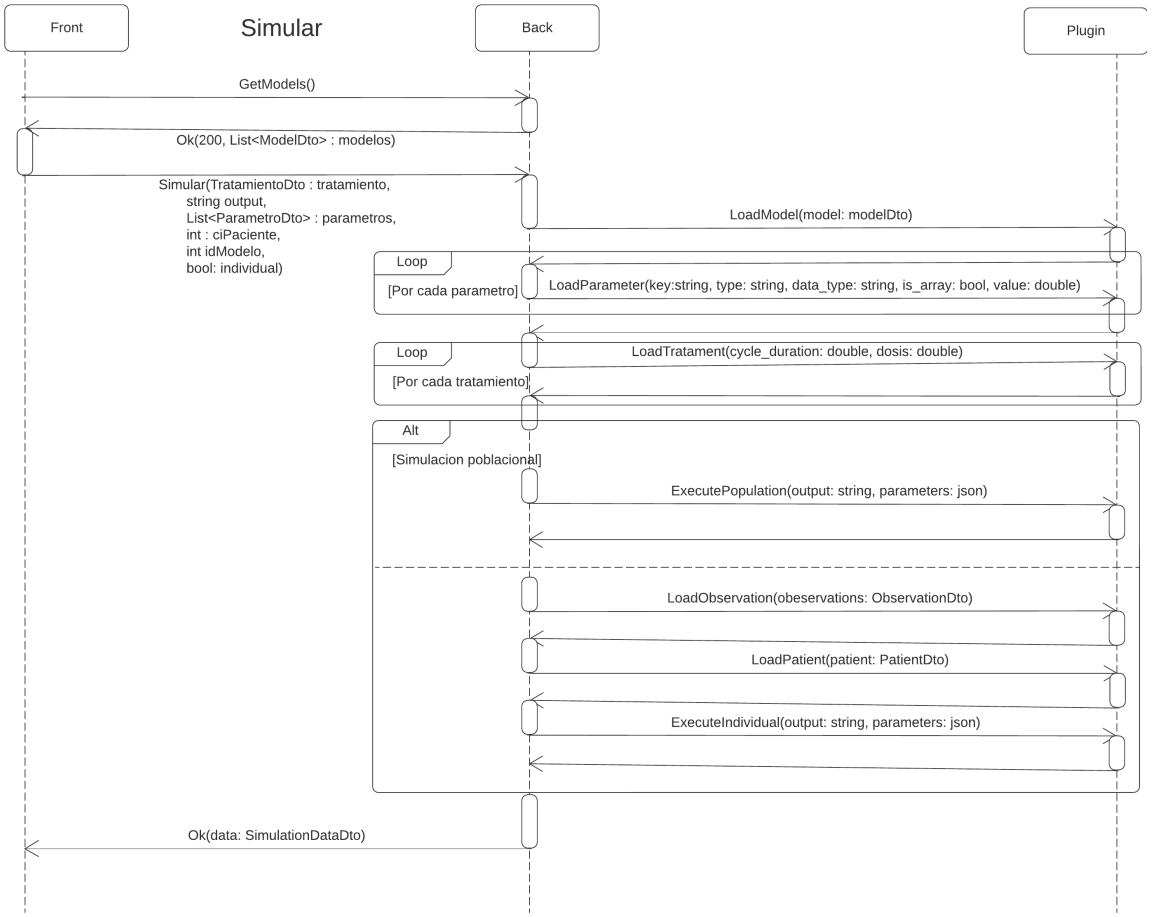


Figura 4.7: Diagrama de secuencia con la interacción con plugins para simular

En la Figura 4.8 se presenta el proceso de carga de observaciones. Inicialmente, el sistema carga las observaciones en el plugin, carga el paciente, carga el modelo y se recalculan los parámetros individuales del paciente.

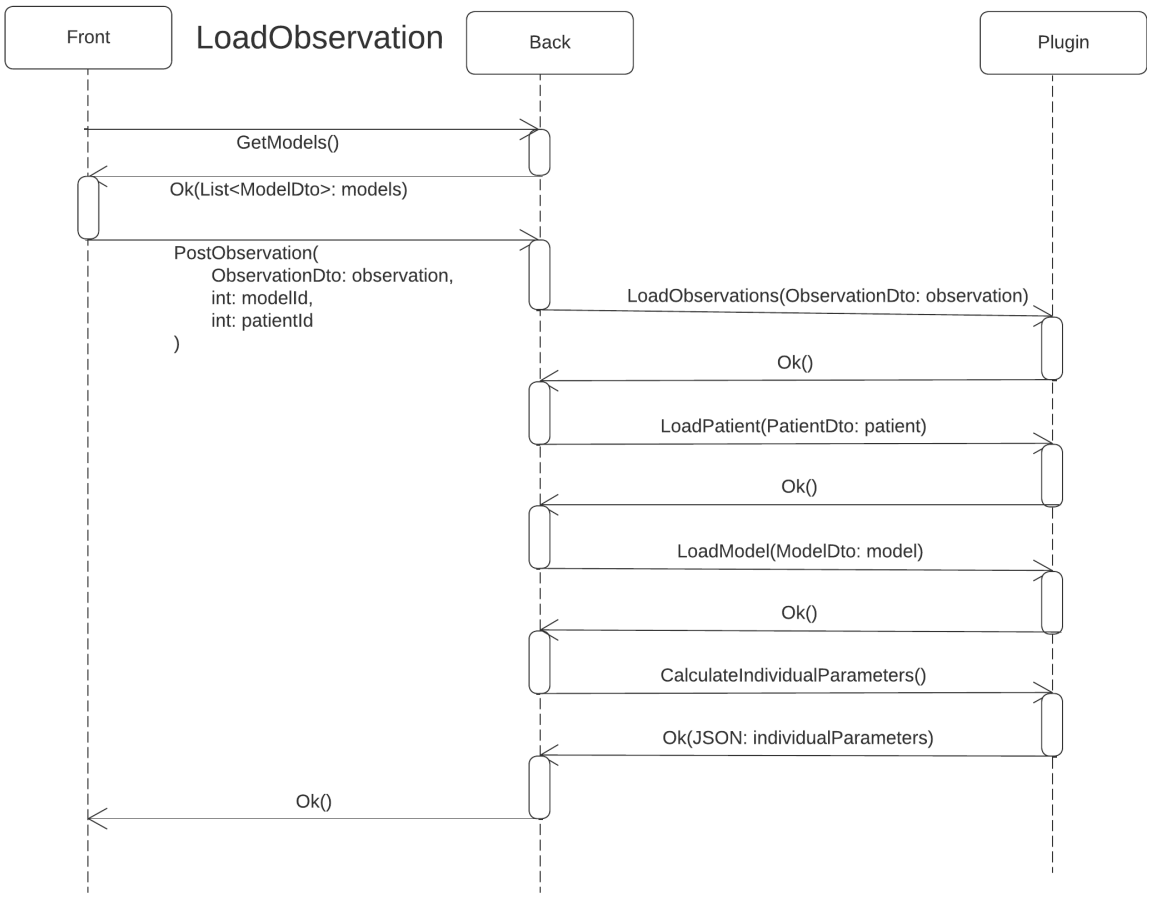


Figura 4.8: Diagrama de secuencia de cargar observaciones

4.5. Simulación objetivo

Como se mencionó en la sección 3.4.5 esta funcionalidad apunta a recomendar al usuario el mejor tratamiento posible para determinado objetivo y paciente.

Para abordar este requisito, se tomaron ciertas restricciones que reducen la cantidad de simulaciones que se deben realizar. Esto apunta a que sea computacionalmente viable su implementación, dado que cada simulación tiene un costo de cómputo del orden de segundos.

En primer lugar, se decidió trabajar con intervalos discretos, tanto de cantidad de dosis cómo de intervalo de administración. Dado que los fármacos se presentan en comprimidos de una masa específica, si un fármaco X tiene una unidad mínima de 10 miligramos, es razonable administrar 5 o 2.5 miligramos (al dividir la pastilla en 2 o 4 partes). Sin embargo, no es viable administrar 1 miligramo, ya que dividir el comprimido en fracciones tan pequeñas no es práctico.

Debido a lo anterior y que los modelos están asociados al fármaco, se agregó un campo en los modelos llamado "*min dose*", el cuál especifica cual es la dosis más pequeña posible. Luego, en base a esa dosis el sistema va incrementando de acuerdo con sus múltiplos hasta llegar a la dosis máxima que especificó el usuario. Por ejemplo, si el usuario ingresa como dosis inicial 2 miligramos y como dosis máxima 4 miligramos,

y si el modelo tiene configurado un “min dose” de 1 miligramo, el sistema probará con los valores 2, 3 y 4 miligramos.

4.5.1. Hipótesis

Para llevar a cabo esta funcionalidad se podría utilizar un “algoritmo de fuerza bruta”, que consiste en probar con todas las dosis e intervalos posibles en el rango especificado por el usuario (producto cartesiano), y quedarse con la simulación cuyo objetivo (AUC(x) o concentración mínima) se aproxima más al deseado.

Sin embargo, en algunos casos, esto conlleva un costo computacional elevado respecto al tiempo de ejecución, con lo cual se plantean las siguientes preguntas:

¿Qué pasa con el AUC(x) y concentración mínima si fijo el intervalo y varío la dosis?

¿Qué pasa con el AUC(x) y concentración mínima si fijo la dosis y varío el intervalo de administración?

En base a estas preguntas surge el que se denominó “algoritmo optimizado”. Se planteó analizar cómo se comportan los modelos al aumentar y/o disminuir los valores en uno de los ejes, dejando el otro fijo respecto al valor objetivo (AUC(x) o concentración mínima). Esto es, dejar fija la dosificación, pero ir cambiando el intervalo de administración, y análogamente, fijar el intervalo de administración, pero variar la dosificación.

4.5.2. Pruebas realizadas

En esta sección se describen las pruebas realizadas, sobre los principales fármacos de interés, para validar las hipótesis descritas en la Sección 4.5.1. Finalmente se describe el algoritmo propuesto y resultados obtenidos al utilizarlo.

Tacrolimus:
En la Figura 4.9 se presenta el resultado de fijar la dosis del fármaco Tacrolimus. Al variar el intervalo entre dosis se puede observar un comportamiento decreciente en el AUC(24) (AUC medido en un intervalo de 24 horas) al incrementar el intervalo entre dosis.

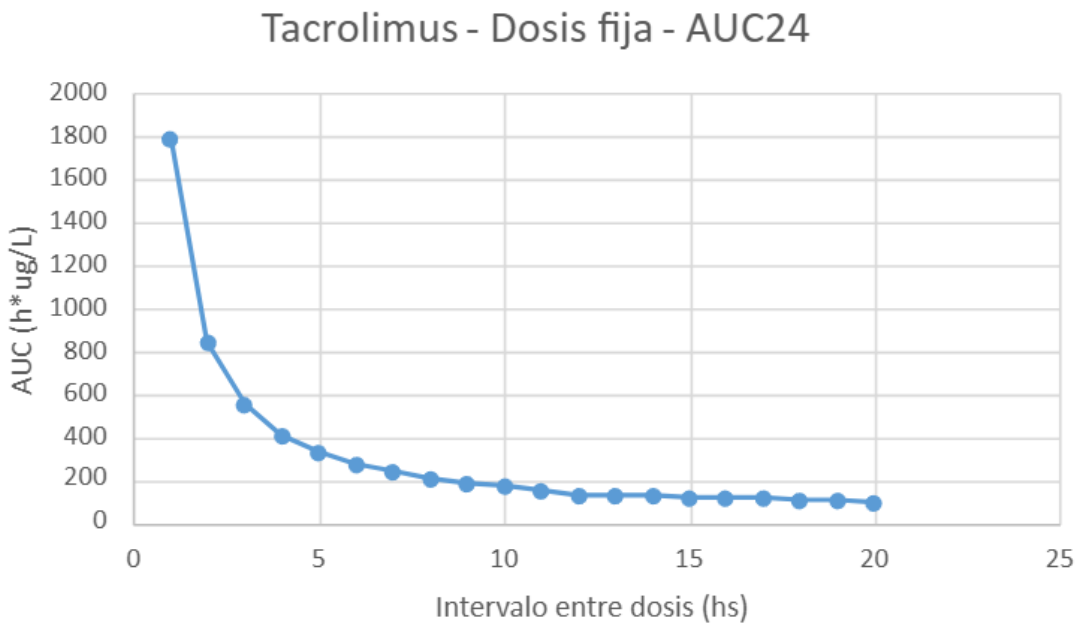


Figura 4.9: Dosis fija AUC(24) Tacrolimus

En la Figura 4.10 se presenta el resultado de fijar la concentración mínima de Tacrolimus. Al variar el intervalo entre dosis se puede observar un comportamiento decreciente en la concentración mínima al incrementar el intervalo entre dosis.

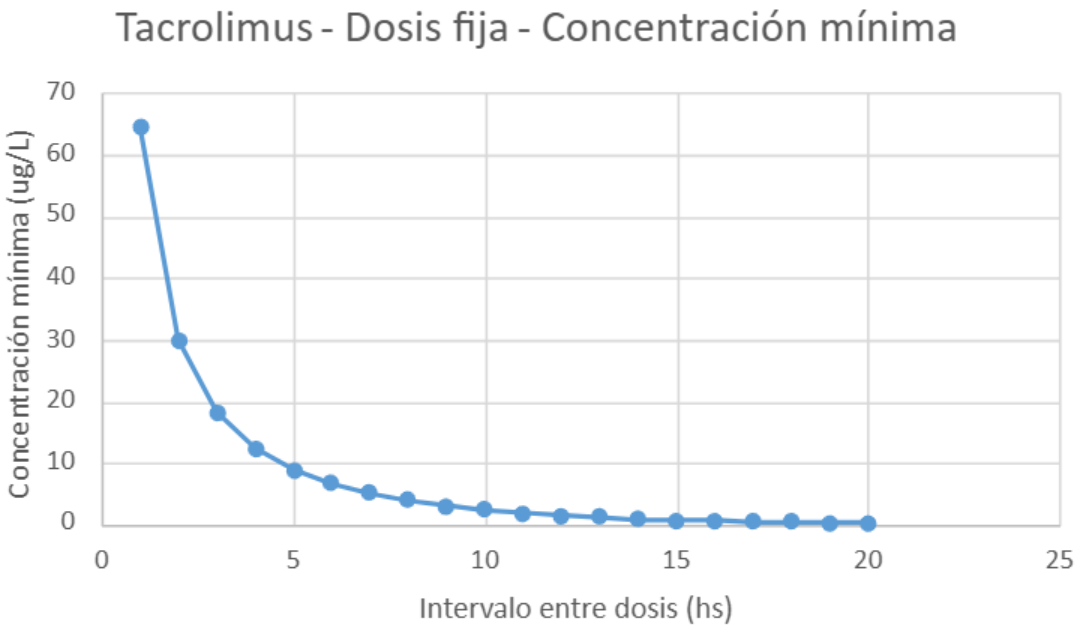


Figura 4.10: Dosis fija concentración mínima Tacrolimus

En la Figura 4.11 se presenta el resultado de fijar el intervalo de administración de Tacrolimus. Al incrementar la dosis se puede observar un comportamiento creciente en el AUC(24).

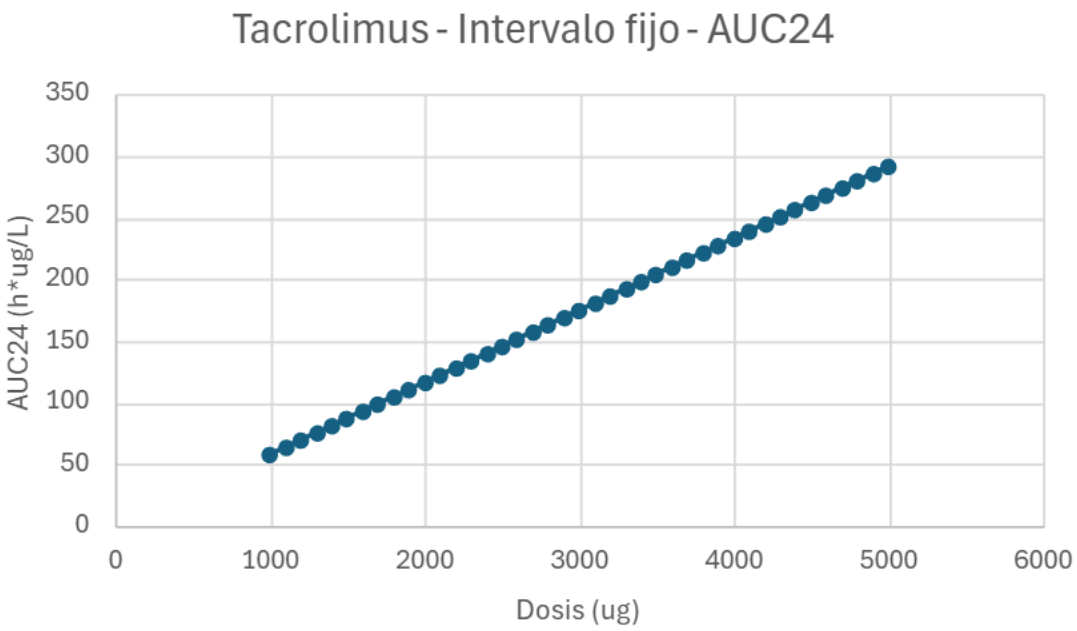


Figura 4.11: Intervalo de administración fijo AUC(24) Tacrolimus

En la Figura 4.12 se presenta el resultado de fijar el intervalo de administración de Tacrolimus. Al incrementar la dosis se puede observar un comportamiento creciente en la concentración mínima.

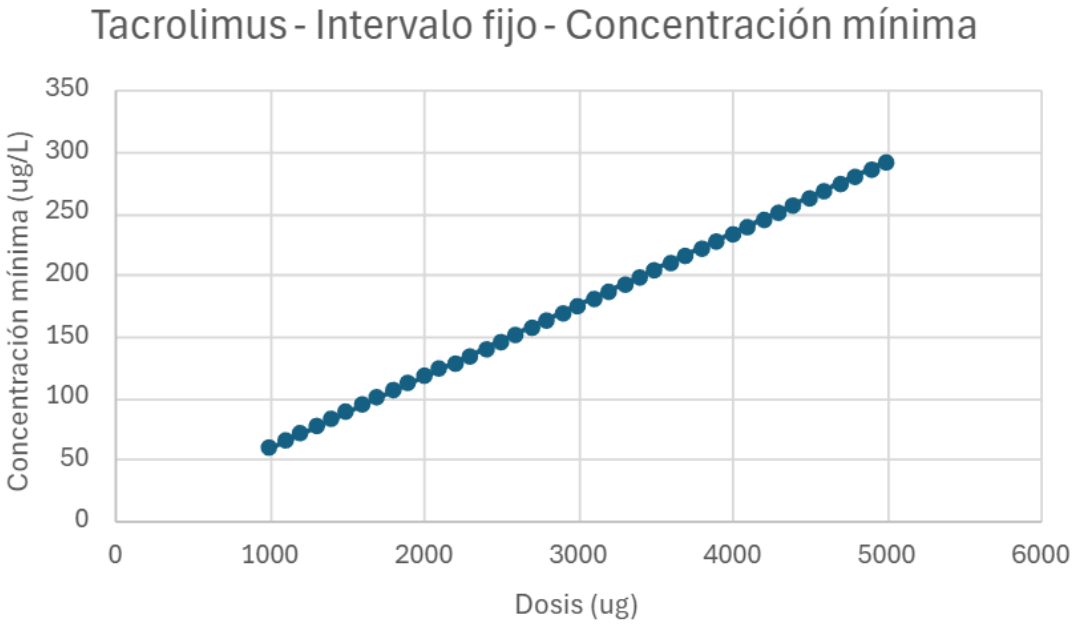


Figura 4.12: Intervalo de administración fijo concentración mínima Tacrolimus

Ciclosporina:

En la Figura 4.13 se presenta el resultado de fijar la dosis de Ciclosporina. Al incrementar el intervalo de administración se puede observar un comportamiento decreciente en el AUC(24).

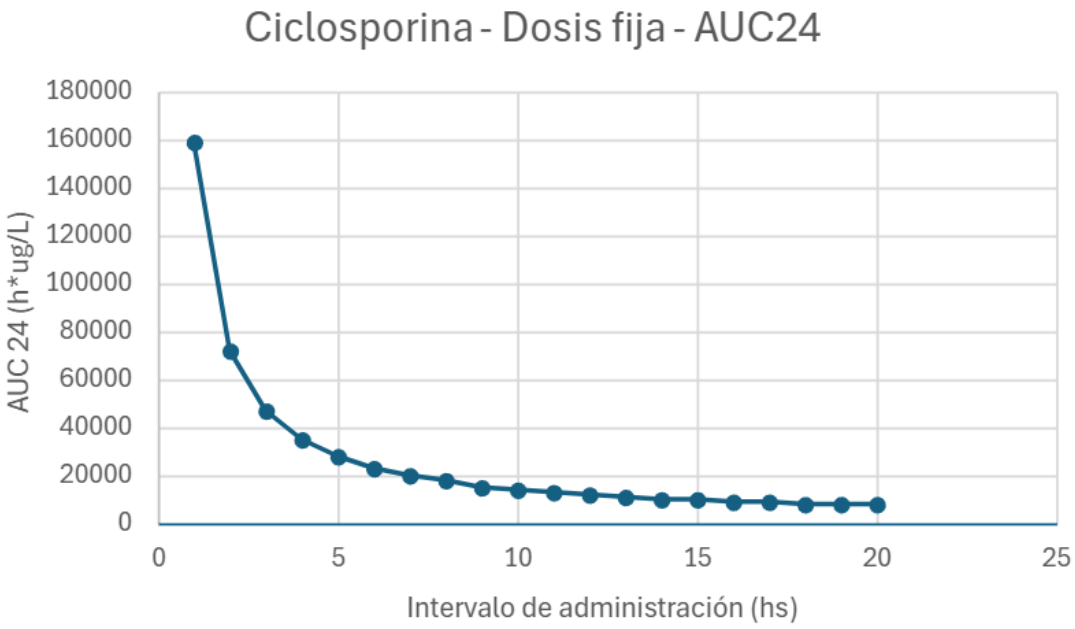


Figura 4.13: Dosis fija AUC(24) Ciclosporina

En la Figura 4.14 se presenta el resultado de fijar la dosis de Ciclosporina. Al incrementar el intervalo de administración se puede observar un comportamiento decreciente en la concentración mínima.

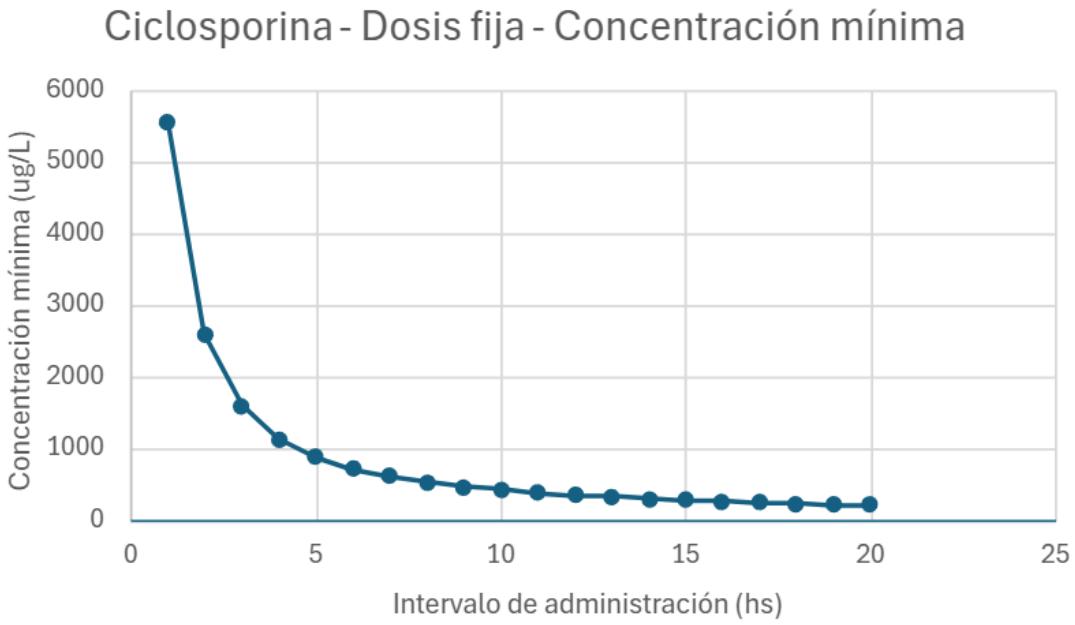


Figura 4.14: Dosis fija concentración mínima Ciclosporina

En la Figura 4.15 se presenta el resultado de fijar el intervalo de administración de Ciclosporina. Al incrementar la dosis se puede observar un comportamiento creciente en el AUC(24).

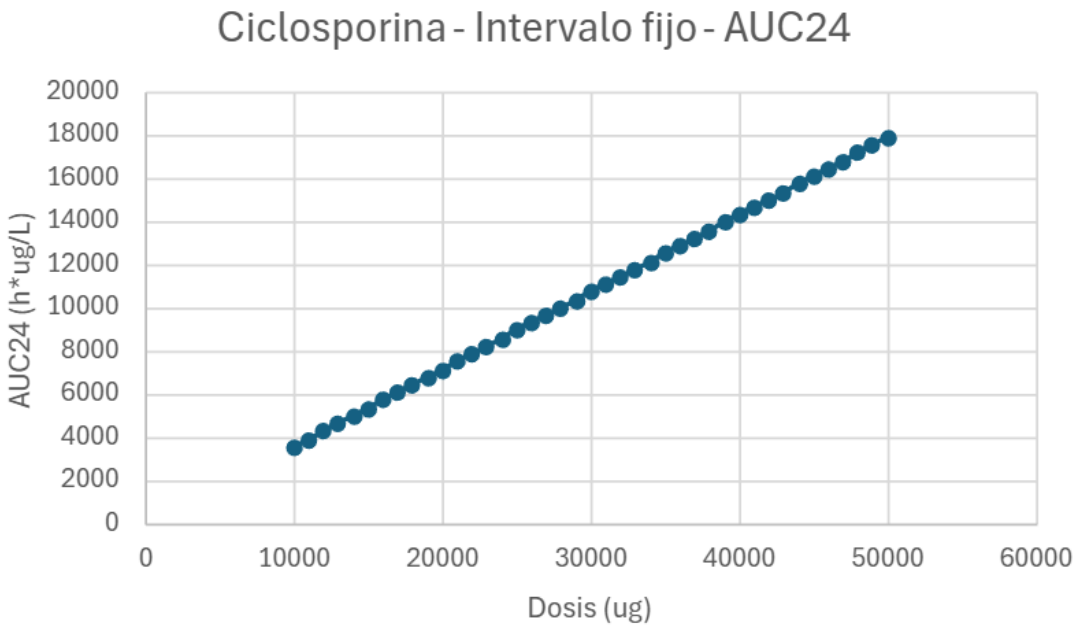


Figura 4.15: Intervalo de administración fijo AUC(24) Ciclosporina

En la Figura 4.16 se presenta el resultado de fijar el intervalo de administración de Ciclosporina. Al incrementar la dosis se puede observar un comportamiento creciente en la concentración mínima.

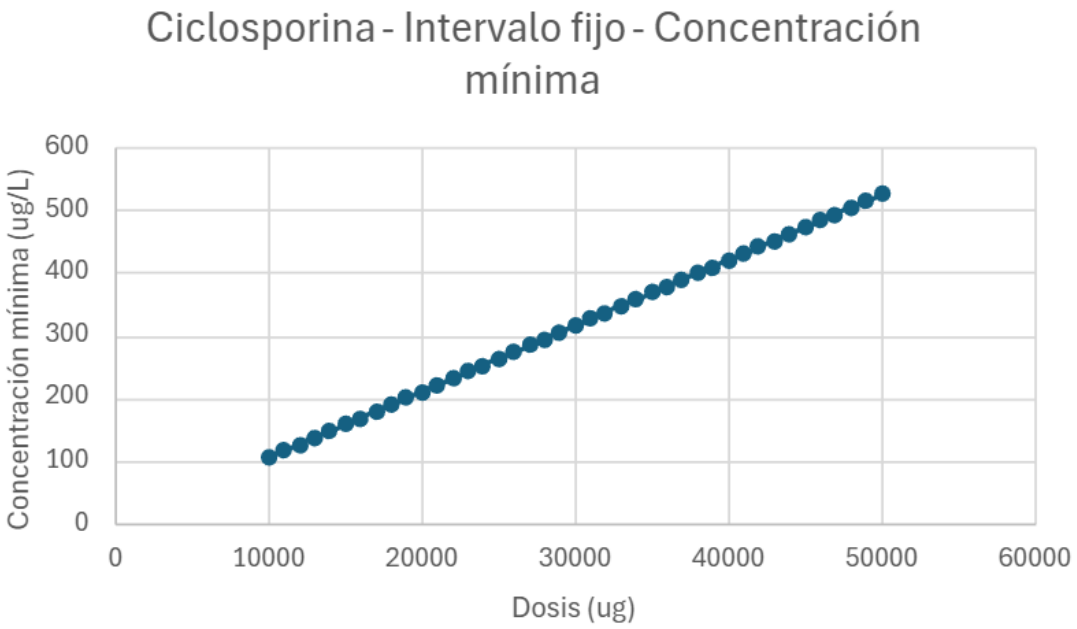


Figura 4.16: Intervalo de administración fijo concentración mínima Ciclosporina

4.5.3. Algoritmos

Como se mencionó previamente, el algoritmo de fuerza bruta consiste en realizar todas las simulaciones posibles e ir quedándose con aquella que más se acerca al valor

objetivo propuesto por el usuario. En este sentido, en el algoritmo de simulación por fuerza bruta, si se tienen n intervalos de administración posibles y m dosis posibles, realiza $m * n$ simulaciones.

Como se puede apreciar en las gráficas de la sección anterior, al aumentar el intervalo entre dosis, como es esperable, el AUC(X) y la concentración mínima se va reduciendo. Por otro lado, al aumentar la dosis, se puede ver que aumenta linealmente tanto el AUC(x) como la concentración mínima. De estas gráficas se desprende que la concentración mínima y el AUC(X) son decrecientes en el intervalo entre dosis, y crecientes en la dosis.

El algoritmo optimizado consiste en probar con todas las dosis posibles, pero para cada una de ellas realizar una búsqueda binaria en el intervalo de administración. Un ejemplo concreto es: si el usuario introdujo un valor de intervalo de administración entre una y seis horas, entonces como primer iteración el algoritmo probará con la mitad de este intervalo, es decir, tomará como intervalo de administración tres tomando en cuenta el redondeo hacia abajo y realizará una simulación con este intervalo de administración. Si el resultado de AUC(X) o Concentración mínima está por debajo del objetivo, gracias a las pruebas empíricas que se realizaron previamente, se sabe que se debe disminuir el intervalo de administración, por lo tanto, el algoritmo descartará valores mayores a tres (ya que por este lado solo conseguirá valores aún más pequeños), y probará con uno y dos.

Debido a que el algoritmo busca el valor que más se aproxime y no el valor exacto, cuando el algoritmo finaliza, se debe probar simular con los últimos dos valores, para elegir el que más se aproxime. Por lo tanto, si se tienen n intervalos de administración posibles y m dosis posibles, el algoritmo realiza $m * (log(n) + 1)$ simulaciones.

4.5.4. Resultados

En cuanto a los resultados, en la Figura 4.17 se presentan los tiempos de un algoritmo frente al otro:

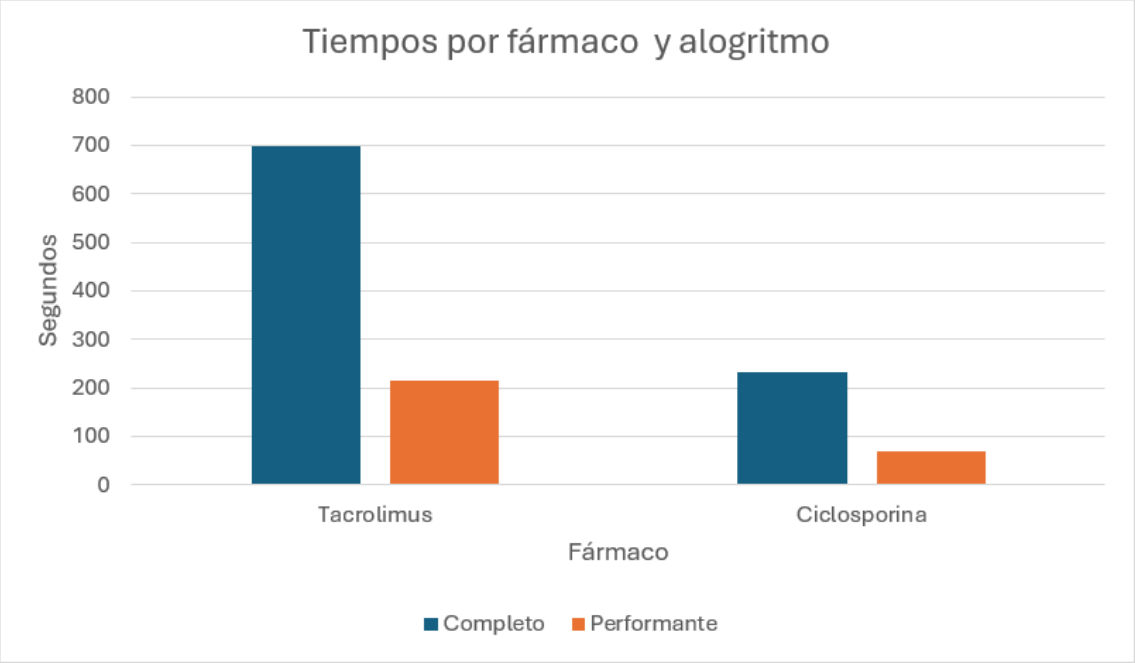


Figura 4.17: Comparación entre algoritmo optimizado vs completo

Para el caso de Tacrolimus se ingresó un intervalo entre dosis de 1 a 20 horas, dosis de 1 a 5 miligramos, con una dosis mínima de 0.1 miligramos y un AUC(24) objetivo de 1 hs*ug/L.

Para la Ciclosporina se ingresó un intervalo de dosis de 1 a 20 horas, dosis de 10 a 50 miligramos, con dosis mínima de 1 miligramo y un AUC(24) objetivo de 1 hs*ug/L.

En las pruebas realizadas se observa que se redujo aproximadamente el 70 % la cantidad de simulaciones realizadas, lo cual es consistente con el cálculo teórico que en el peor de los casos se reduciría solo el 50 % la cantidad de simulaciones necesarias. El algoritmo escala logarítmicamente en el eje de los intervalos y linealmente en el eje de las dosis.

La razón por la que se hace la búsqueda binaria sobre el intervalo de tiempo es porque se consideró que en la mayoría de los casos si se discretizan ambos ejes, el intervalo de tiempo era más probable que sea el que tenga más elementos, mientras que la dosis, al ser limitada por la masa de los comprimidos tendría menos elementos. Sin embargo, en las reuniones de validación surgió la idea de realizar la búsqueda binaria en la dosis, ya que en la práctica solo se utilizan intervalos de tiempos que sean múltiplos de cuatro, es decir nunca dan un comprimido cada tres horas, por ejemplo. Esto hace que los intervalos de tiempo a probar sean pocos, mas precisamente 4, 8, 12, 16, 20 o 24 horas. Debido a que en las pruebas realizadas los tiempos obtenidos fueron buenos, esta nueva idea se dejó como trabajo a futuro.

En la Figura 4.18 se describe el diagrama BPMN, que explica el proceso esperado para poder obtener el tratamiento óptimo.

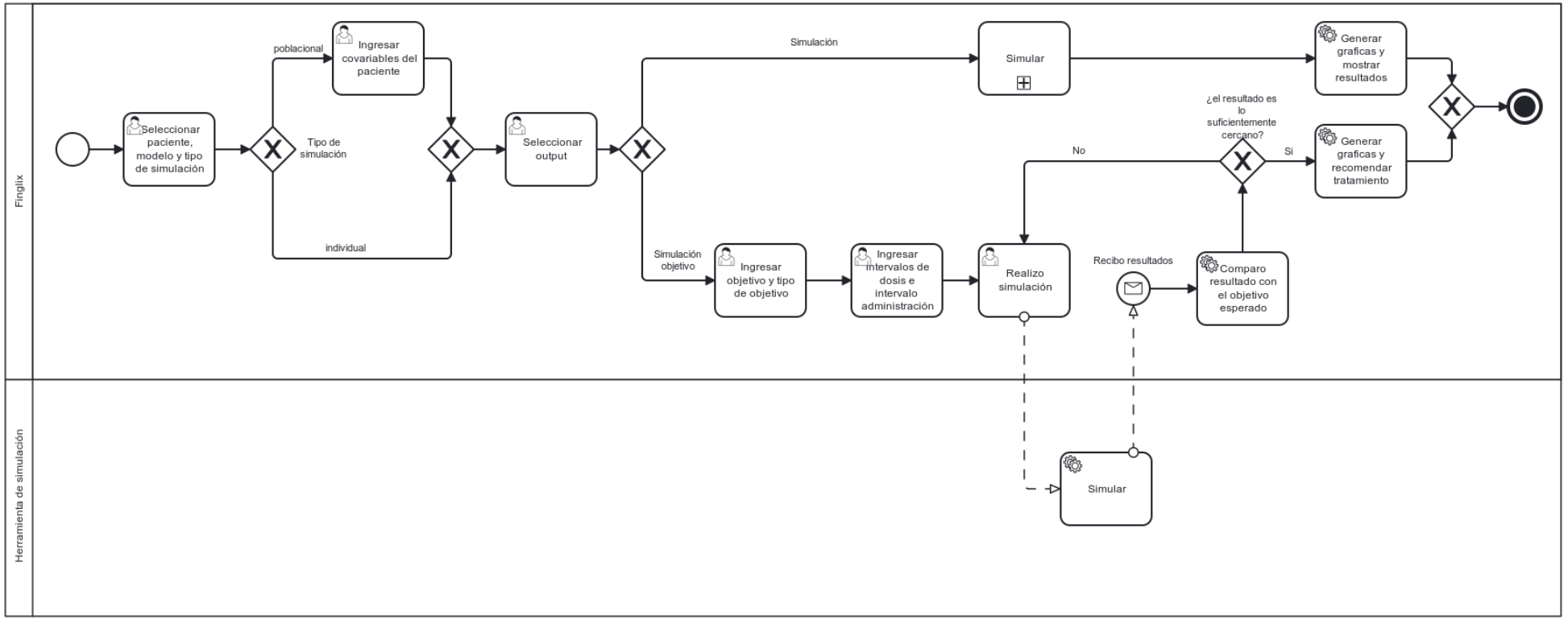


Figura 4.18: Diagrama BPMN - Proceso de simulación objetivo

Capítulo 5

Implementación

En este capítulo se introducen las tecnologías utilizadas en Finglix 1.0, luego se describe el proceso de implementación del sistema y las decisiones tomadas para cumplir con el diseño propuesto en el capítulo anterior.

5.1. Tecnologías del proyecto anterior

El diagrama 5.1 muestra las tecnologías utilizadas en Finglix 1.0, qué rol cumplían y cómo se comunicaban entre ellas.

Para implementar el *frontoffice* se utilizó React, que es un *framework* de Javascript. Como tecnología de *backend* se utilizó Django, un *framework* de Python. El *frontoffice* se comunica con el *backend* a través de una API REST. Además en el *backend*, se utiliza RabbitMQ[1] y Celery[2] para gestionar la cola de mensajes, permitiendo que Django envíe tareas asíncronas. Tanto el *worker* como la API del *backend* interactúan con el sistema de archivos de Linux y la base de datos PostgreSQL.

[1] `https://www.rabbitmq.com/`
[2] `https://docs.celeryq.dev/en/stable/`

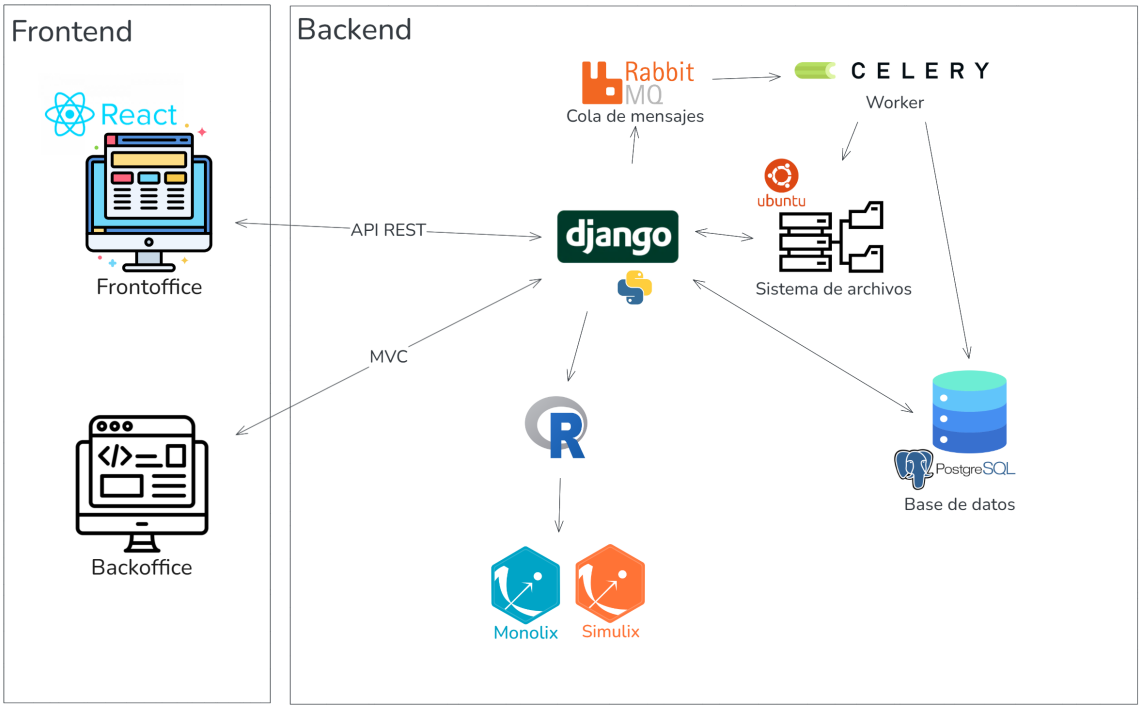


Figura 5.1: Tecnologías Finglix 1.0

5.2. Alternativas analizadas

Se analizaron dos posibles enfoques para la implementación de Finglix 2.0, reutilizar el proyecto anterior o crear un nuevo proyecto con otras tecnologías. Esto aplica solo para el *backend*, ya que rápidamente se observó que no había necesidad de rehacer el *frontend* debido a que las nuevas funcionalidades que se evaluaron no difieren significativamente de las existentes. Además, la interfaz de usuario ya había sido validada previamente por el equipo anterior. Por lo tanto, se consideró más beneficioso para el proyecto expandir las funcionalidades actuales del *frontend* y mejorar sus puntos fuertes basándose en retro alimentación recibida del proyecto anterior. La arquitectura del grupo anterior permite esta flexibilidad, ya que el componente *frontend* está desacoplado del lenguaje del *backend*, ya que se comunica a través de una API REST.

5.2.1. Opción de reutilización

Reutilizar el *backend*, y por ende las tecnologías de éste, presenta la ventaja de aprovechar muchas funcionalidades ya realizadas en Finglix 1.0. Sin embargo, el equipo no posee tanta experiencia en las tecnologías utilizadas, lo que genera un esfuerzo en aprenderlas y además requiere comprender el código de las funcionalidades existentes para poder hacer las modificaciones correspondientes y agregar las nuevas. Además, otra desventaja es que el sistema tiene un alto acoplamiento con MonolixSuite, lo que dificultaría la integración de nuevas herramientas y requeriría una reestructuración significativa del código para desacoplarlo.

5.2.2. Opción de un nuevo sistema

Ya que el sistema existente (Finglix 1.0) no es muy extenso y la calidad del código es buena, crear un nuevo sistema desde cero tiene sentido si las tecnologías son otras y presentan claras ventajas en comparación a las utilizadas en el sistema existente. Por esto el análisis de posibles nuevas tecnologías se basó básicamente en dos puntos: que tenga el mismo o mejor potencial que las tecnologías existentes para el sistema que se desea implementar, y la experiencia del equipo.

El equipo cuenta mayormente experiencia en las tecnologías del ecosistema *.NET*. Estas tecnologías ofrecen una serie de ventajas clave que pueden mejorar significativamente la arquitectura y la flexibilidad del sistema. El beneficio principal de reemplazar el *backend* con .NET radica en la utilización de un *framework* más estructurado y robusto.

Esto se debe a que .NET ofrece un conjunto sólido de herramientas y características que permiten un desarrollo más organizado y escalable, ya que es un lenguaje orientado a objetos el cual tiene soporte para control de acceso en objetos, tipos de datos estáticos, es un lenguaje compilado no interpretado entre otras características que lo convierten en una buena opción a contemplar. Lo que podría facilitar la incorporación de nuevas herramientas de simulación en el futuro. Al construir el sistema desde cero, se puede diseñar con una estructura que sea inherentemente flexible y adaptable, lo que facilitaría los cambios y ajustes necesarios para integrar futuros sistemas de simulación.

Además, esta transición podría eliminar el *middleware* de mensajería (Rabbitmq) y optar por un manejo más simple y nativo de las tareas de simulaciones en segundo plano ya que .NET tiene un manejo más simple de servicios sin la necesidad de usar herramientas extras, otra ventaja importante de este enfoque es que al construir el *backend* desde cero, se puede diseñar con el objetivo de facilitar su despliegue y flexibilidad frente a cambios. Un diseño cuidadoso podría permitir un proceso de implementación más fluido y eficiente, reduciendo el tiempo y los recursos necesarios para poner en marcha nuevas versiones o actualizaciones.

Como desventaja reconstruir el *backend* implica un retrabajo significativo, pero es importante destacar que se cuenta con el análisis del equipo anterior. Este conocimiento previo facilitaría la identificación de los puntos débiles y áreas de mejora en el proyecto actual. Al reestructurar el proyecto, se podrían fortalecer estos puntos, lo que resultaría en un sistema más robusto y resistente a las dificultades que se hayan enfrentado anteriormente.

5.2.3. Prototipo Python

Dado que la solución anterior ya usaba Python, no fue necesario probar la integración con R. Para probar la capacidad de Python de operar con un sistema de plugins, se desarrolló un prototipo con este fin. Este prototipo permitía instalar y desinstalar dinámicamente librerías de Python usando el módulo *subprocess*. Con la construcción exitosa de este prototipo fue posible verificar la capacidad del lenguaje para soportar un sistema de plugins. Este sistema busca permitir que los desarrolladores puedan

utilizar otras herramientas de simulación.

5.2.4. Prototipo C#

Se desarrollaron dos prototipos. El primero fue un programa de consola que ejecutaba código R mediante *RDotNet* para conectar C# con R, logrando ejecutar comandos básicos.

El segundo prototipo consistió en cargar y ejecutar módulos en tiempo de ejecución, creando una *DLL* que contenía una función simple. La exitosa construcción de este prototipo valida la capacidad del lenguaje para operar con plugins. Esto es posible a través de la carga de *DLLs*, en tiempo de ejecución.

5.2.5. Decisión

Debido a que el prototipo de Python demostró que la tecnología existente posee las capacidades necesarias para el sistema de plugins y además, fue posible generar un entorno de desarrollo para Finglix 1.0 funcional, se decidió reutilizar todas las tecnologías existentes, ya que el retrabajo de hacer el *backend* en otras tecnologías implicaría un consumo de tiempo mucho mayor, respecto a adaptarse a las tecnologías ya existentes pese a la inexperiencia en ellas.

5.3. Frontend

El *frontend* de Finglix 2.0 como se mencionó previamente, fue reutilizado. Se agregaron y modificaron los componentes necesarios para soportar las nuevas funcionalidades desarrolladas. Además, se realizaron mejoras sobre algunos de los componentes ya existentes, entre ellos los principales son: validaciones de valores ingresados, para evitar valores fuera de rango o tipos de datos no deseados, claridad en la información de las pantallas y mejoras gráficas.

5.3.1. Nuevas funcionalidades

Se implementaron dos nuevas funcionalidades de forma completa: simulación objetivo y visualización de precisión en las predicciones.

En la funcionalidad simulación objetivo el sistema muestra un nuevo formulario donde se deben ingresar los datos necesarios. En la Figura 5.2 se observa un ejemplo. Además, debido a que es una funcionalidad que tiene un tiempo de ejecución muy variado porque depende de la cantidad de simulaciones que requiera realizar el sistema, se le indica al usuario aproximadamente cuanto se estima que demorará en ejecutar.

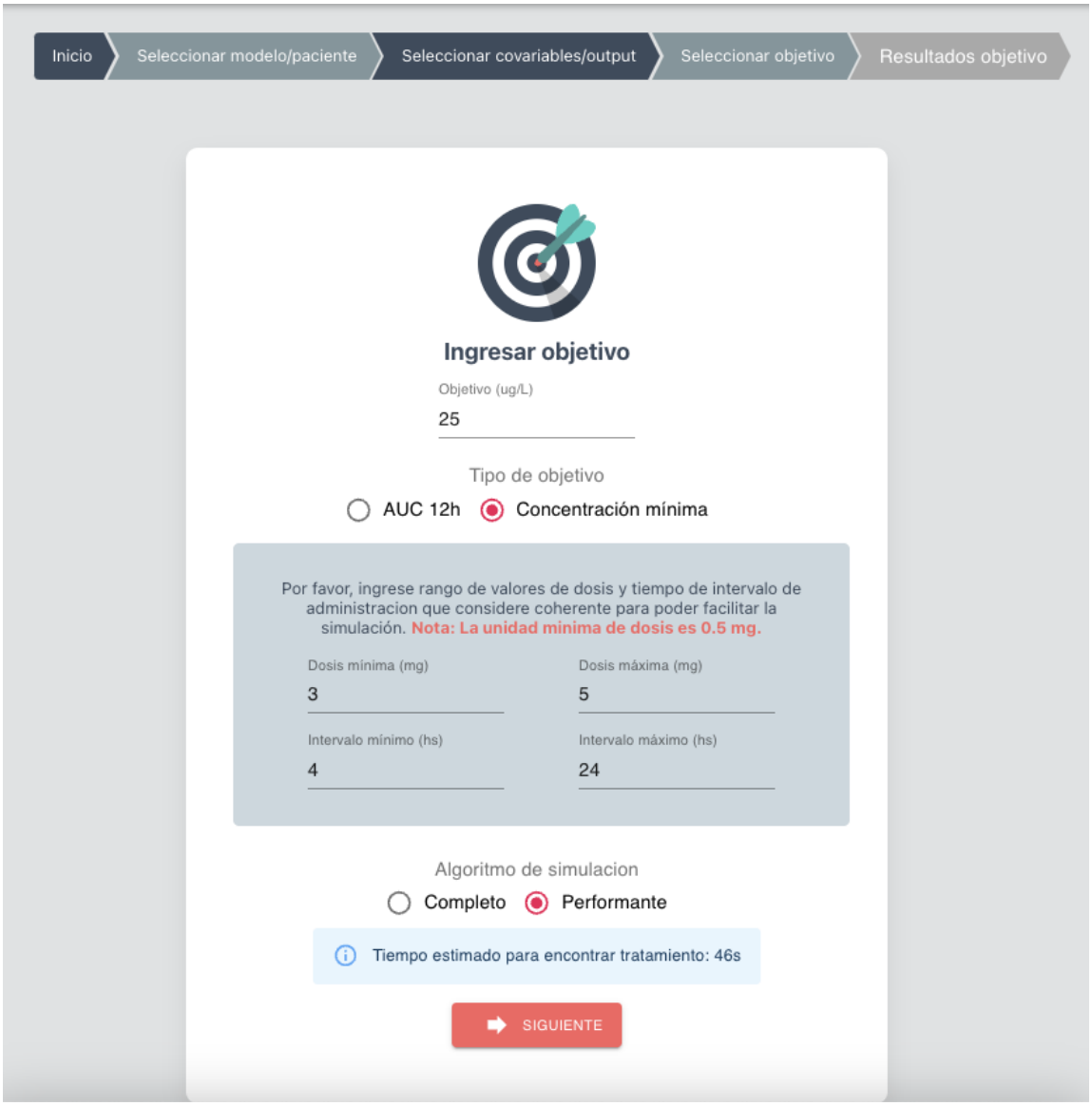


Figura 5.2: Simulación objetivo

Por otro lado, en la visualización de "precisión en las predicciones" se agregó un nuevo botón, el cual luego de cargada la observación, permite comparar las simulaciones respecto a la observación recién ingresada. En la Figura 5.3 se observa un ejemplo, los puntos rojos representan datos observados del paciente, la curva verde representa la concentración estimada en función del tiempo usando parámetros poblacionales y la curva azul representa la concentración estimada en función del tiempo usando los parámetros estimados a partir de las observaciones.

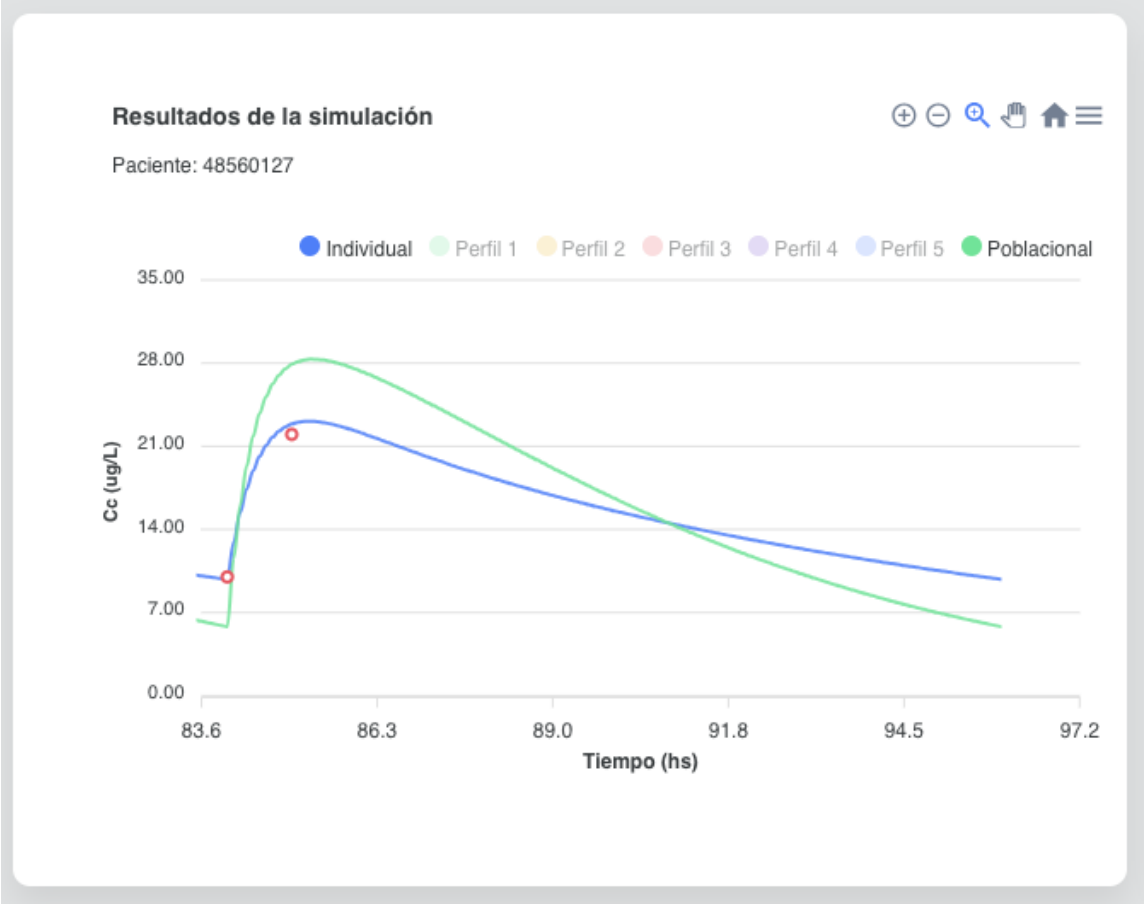


Figura 5.3: Comparación entre observaciones y simulaciones

5.3.2. Actualización gráfica de la plataforma.

Si bien se reutilizó el *frontend* y se siguió la misma metodología para los nuevos flujos de forma de mantener la misma experiencia de usuario, se agregaron varias modificaciones y validaciones para mejorar la usabilidad del sistema. Entre los cambios más importantes se encuentran:

- Paleta de colores: se utilizaron tonos de grises y verdes para el fondo. Para las acciones se utilizaron dos colores más fuertes (rojo #ff5b62 y turquesa #21d0c3).
- Imágenes: se cambiaron por imágenes en formato SVG[3]. Además, se editaron para seguir la paleta de colores del sitio.
- Se removieron las imágenes de los integrantes del grupo de proyecto de grado anterior de la pantalla de inicio, debido a que, si el objetivo era mostrar el equipo de desarrollo, agregar tres imágenes más no tendría mucho sentido. Como trabajo a futuro se podría crear una sección para mostrar esta información.
- Se agregaron validaciones a los campos en los formularios donde se ingresan variables del modelo o datos del paciente. Finglix 2.0 permite indicar para cada

[3]https://datatracker.ietf.org/doc/html/rfc7996

parámetro su rango de valores. En caso de no estar en el rango apropiado, se indica un error al usuario.

- En la sección de cargar datos de un paciente o variables del modelo, al igual que en el punto anterior, si la configuración del modelo lo indica, los campos también pueden ser de tipo lista de valores posibles, para seleccionar una opción entre una cantidad finita de opciones.
- Se agregaron más alertas y validaciones, por ejemplo, que un usuario no pueda realizar simulaciones individuales si todavía no se han ingresado observaciones.
- Cálculo de demora aproximada de en la búsqueda de la simulación objetivo.
- Barra de carga que ajusta su duración respecto a la demora estimada de cuando se quiere encontrar la simulación objetivo.
- Mostrar qué paciente, modelos y herramienta de simulación se esta utilizando mientras se espera a que termine la acción de simular y luego de simular.
- Se modificó o quitaron secciones innecesarias que hacían que las distintas páginas quedaran muy sobrecargadas. Este cambio logra que en una pantalla de tamaño estándar (catorce pulgadas o más) no es necesario realizar *scroll* en prácticamente ninguna sección de la aplicación. En las figuras 5.4 y 5.5 se aprecia la diferencia.

Figura 5.4: Finglix 1.0

Figura 5.5: Finglix 2.0

5.4. Backend

En esta sección, se detalla cómo se implementaron las principales funcionalidades en el *backend*.

5.4.1. Integración con múltiples plataformas

Como se mencionó en la sección 4.4, se definió un estándar inicial para los plugins, un contrato que debe cumplir, desde cómo se debe llamar cada archivo, qué funciones debe proveer, cómo se llamarán y en qué orden se invoca a cada función y qué se espera que realicen.

En la Figura 5.6 se encuentra la estructura de carpetas de los plugins, la cual sigue una organización específica para asegurar su correcto funcionamiento dentro del sistema. Al cargar un plugin, se crea una carpeta dedicada que contiene todos los archivos descomprimidos del archivo ZIP del plugin. Es obligatorio que dentro de esta carpeta exista un archivo llamado *main.py*, el cual incluye todas las funciones y características definidas según el estándar del sistema.

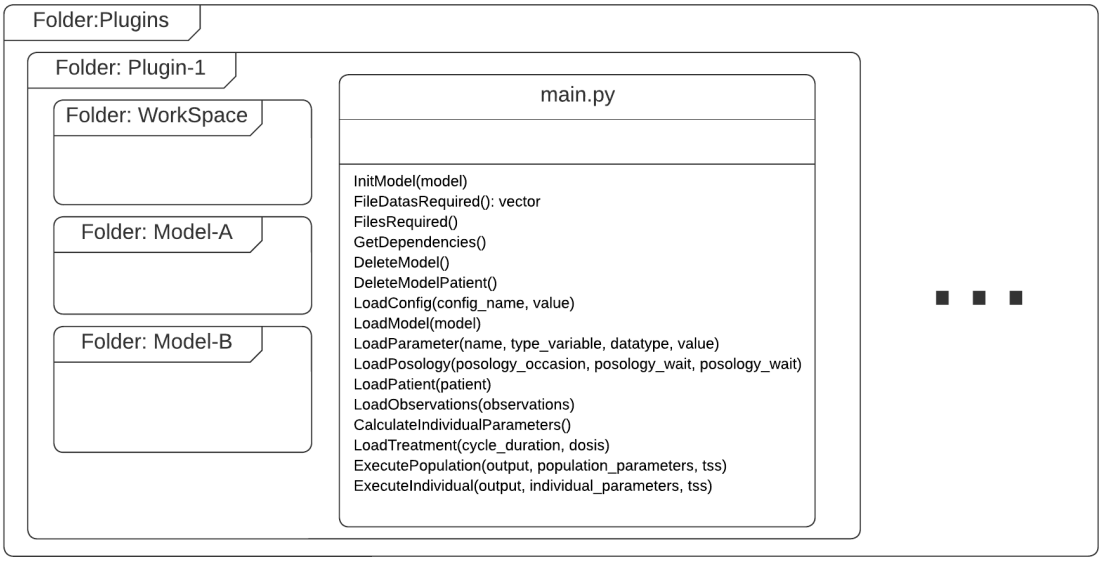


Figura 5.6: Estructura de carpetas

Dentro de la carpeta de cada plugin, cuando se carga un modelo asociado a ese plugin, se genera una nueva subcarpeta específica para dicho modelo. Siguiendo la misma jerarquía, al cargar observaciones de un paciente para un modelo, se crea una carpeta adicional dentro de la carpeta del modelo para almacenar las simulaciones correspondientes a ese paciente. Esta estructura jerárquica busca lograr una organización clara y sistemática de los archivos y carpetas, facilitando el acceso y la gestión de los plugins, modelos y observaciones de pacientes.

Para poner a prueba la interfaz definida, se creó un primer plugin de MonolixSuite reutilizando parte del código del grupo anterior (desacoplándolo de la aplicación), Esta primera versión fue útil para ajustar y apropiarse del conocimiento relevante al flujo de trabajo de las simulaciones. Además, permitió evaluar el primer diseño de arquitectura del sistema de plugins, habiendo repetido posteriormente el proceso con prototipos de Nlmixr2 y Mgrsolve. Con la generación de prototipos de plugins se logró ir ajustando el estándar de la interfaz de plugins de tal manera que se logre un nivel de abstracción que se aparta del funcionamiento particular de cada tecnología y herramienta.

5.4.2. Plugins desarrollados

Como continuación a lo explicado en la sección anterior, se realizaron tres plugins, para tres herramientas distintas, con distintos tipos de profundidad y alcance. Primero se desarrolló en su totalidad un plugin para MonolixSuite, manteniendo todas las funcionalidades provistas en Finglix 1.0 y otras que se detallaran en la siguiente sección . Luego, se desarrollaron dos plugins más para las herramientas Mrgsolve+Mapbayr y Nlmixr2, aunque por el alcance de este proyecto, no era posible lograr la misma profundidad y detalle que en MonolixSuite.

5.4.2.1. Plugin para MonolixSuite

Para realizar este plugin, se trató de extraer la mayor parte de la lógica de Finglix 1.0. Principalmente se aprovechó todo el manejo de archivos que requiere esta herramienta, tanto para leer los modelos, los conjuntos de datos, como la creación de archivos temporales necesarios para realizar las simulaciones. También, se pudo reutilizar en buena parte la funcionalidad de estimar parámetros. Los mayores cambios fueron las simulaciones, pero principalmente por el cambio que fue necesario realizar para el soporte de modelos que utilizan regresores, teniendo que crear una nueva rutina de simulación en R. Además, esta nueva rutina ahora requiere solo un archivo por modelo. Cómo se mencionó en la Sección 3.2.4, previamente se necesitaba dividir el modelo en tres archivos de una forma específica para soportar cada acción (estimar parámetros individuales, parámetros poblacionales y simular). También, se aprovechó para actualizar la versión de MonolixSuite a la versión R12023 lo que no requirió grandes cambios respecto a la anterior.

Se hizo foco en fortalecer la flexibilidad multi modelo de este plugin, para lograr este objetivo se trabajó en que sea lo suficiente flexible como para soportar distintos modelos y no únicamente la Ciclosporina. Concretamente, se probó con modelos de Tacrolimus (que tienen la particularidad que utilizan regresores) y también se probó con modelos de Vancomicina.

5.4.2.2. Plugin para Mrgsolve

Implementar este plugin presentaba los siguientes desafíos.

Flujo de trabajo: Tanto Mrgsolve como Mapbayr son paquetes de R, lo que significa que la rutina a seguir para realizar estimaciones y simulaciones son realizadas en memoria y no utilizan archivos temporales a diferencia de MonolixSuite.

Conocimiento: la experiencia del equipo del CIENFAR no era tanta como con MonolixSuite, implicando que no se tuvieran muchos modelos y principalmente modelos probados y utilizados. Sin embargo, se pudo acceder a algunos modelos de prueba del fármaco Tacrolimus para esta herramienta de simulación.

Estimaciones: Mrgsolve no realiza estimaciones de parámetros, por lo tanto, era necesario complementarlo con código auxiliar para realizar esta tarea. En este sentido, el equipo del CIENFAR sugirió utilizar el paquete Mapbayr, el cual es un paquete de R que permite realizar estimaciones bayesianas de máximo a posteriori.

El plugin desarrollado permite realizar simulaciones individuales y poblacionales, y además que se estimen parámetros. También se pudieron realizar simulaciones objetivo. En resumen, este plugin soporta las mismas funcionalidades que el plugin de MonolixSuite. Sin embargo, es importante remarcar que la profundidad y flexibilidad de este plugin no fue probada, ya que se probó con modelos del mismo fármaco, y muy similares entre sí. Tampoco fueron extensas las validaciones de los resultados obtenidos.

En las siguientes rutinas de R, que son ejecutadas por el plugin, se muestran dos

ejemplos muy básicos de cómo realizar simulaciones poblacionales y cómo estimar parámetros. Luego, en las figuras 5.7 y 5.8 se presentan algunos ejemplos de simulaciones.

```
# Simulacion poblacional

library(mrgsolve)
library(mapbayr)
library(shiny)

model <- # Modelo

my_model <- mcode("Example_model", model)

my_data <- # Covariables

mrgsolve = my_model %>%
    ev(my_data) %>%
    mrgsim(end=tss,delta=1)
output <- as.data.frame(mrgsolve)

# Estimacion de parámetros

library(mrgsolve)
library(mapbayr)
library(shiny)

model <- # Modelo
my_model <- mcode("Example_model", model)
data <- # Covariables

my_est <- mapbayest(my_model, data = data)
```

Figura 5.7: Simulación poblacional utilizando Mrgsolve

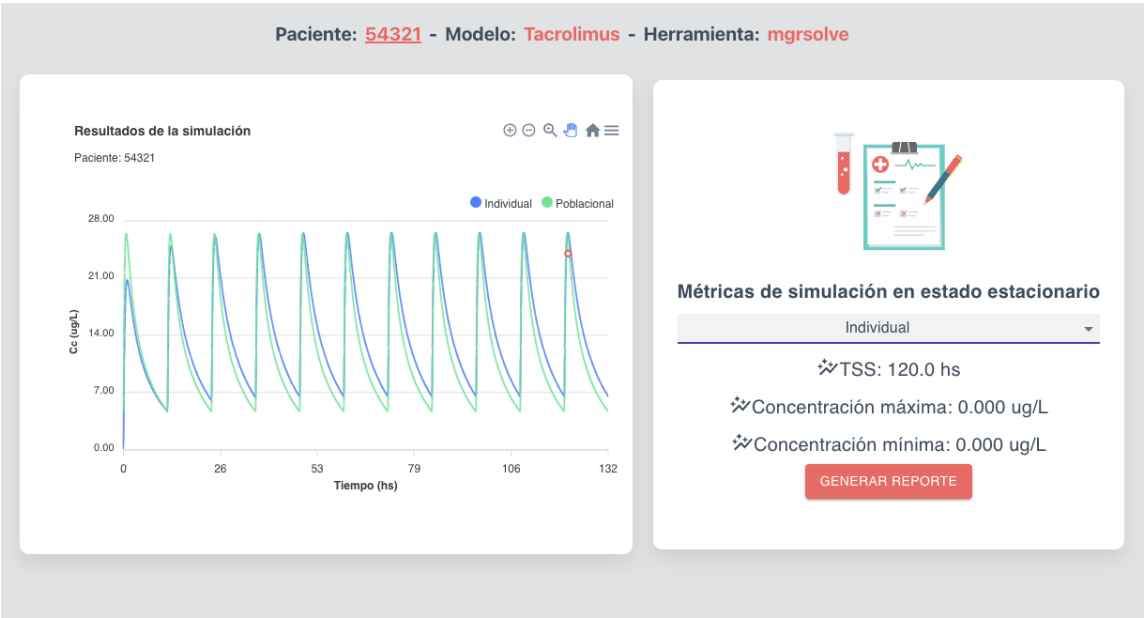


Figura 5.8: Cargar observación y simular utilizando Mrgsolve

5.4.2.3. Nlmixr2

Nlmixr es otra herramienta que tiene un flujo de trabajo similar a Mrgsolve, ya que también es un paquete de R. Esta herramienta es más compleja que Mrgsolve debido a que ofrece más funcionalidades, entre ellas estimar parámetros. El principal objetivo en la construcción de este plugin era lograr que al menos se puedan hacer simulaciones poblacionales para algún modelo de cierto fármaco. De la misma manera que los anteriores el modelo que se pudo obtener fue para el fármaco Tacrolimus.

La rutina utilizada en el plugin para simular fue la siguiente:

```
simulate <- function() {
    library(nlmixr2)
    library(rxode2)
    my_model <- function() {
        ini({
            # Parametros poblacionales
        })
        model({
            # Modelo
        })
    }

    my_data <- data.frame(
        # Datos del paciente
    )

    my_dose <- eventTable() %>%
        add.dosing(
            # Tratamiento
        )
    sim <- rxSolve(my_model, my_data, my_dose)
}
```

El reto más grande fue abarcar las distintas formas de representar los datos que permiten los modelos de esta herramienta. Por ejemplo, en esta herramienta es común que los parámetros poblacionales ya estén definidos en el modelo. Por lo tanto, cuando el administrador agrega un modelo, Finglix debe interpretarlos para: almacenarlos, utilizarlos en las simulaciones y desplegarlo en el *backoffice*. Estos parámetros poblacionales tienen las siguientes particularidades:

- Algunos parámetros pueden ser definidos como una aproximación (utilizando el operador ~) o como valores fijos (utilizando la función *fix()*). Se resolvió que los parámetros, covariables, datos del paciente, etc que represente ese tipo de dato deberían llamarse en Finglix *app*.NOMBRE y *fix*.NOMBRE respectivamente.
- Hay parámetros en forma de vectores. En estos casos, se realizan las transformaciones de las listas en Python a los arreglos en R
- Existen parámetros encapsulados en funciones matemáticas como *log()* o *exp()*. En estos casos, se necesita que el administrador use los valores ya calculados.

A continuación, se muestra un ejemplo de cómo son almacenados los parámetros poblacionales en Finglix y como son transformados para Nlmixr2.

```
# Finglix
{
    "tka": 2.49,
    "app.Q": 0.82,
    "cl": [0, 0.25],
    "fix.b": 1
}

# Nlmixr2
{
    tka <- 2.49;
    Q ~ 0.82;
    cl <- c(0, 0.25);
    b <- fix(1);
}
```

En la Figura 5.9 se muestra una simulación realizada en Nlmixr2 luego de haber transformado todos los datos provistos por Finglix en la estructura de datos esperada por Nlmixr2.

Figura 5.9: Simulación poblacional con Nlmixr2

5.4.3. Cargar modelos

Cargar modelos farmacocinéticos era una funcionalidad que ya estaba desarrollada, no era lo suficientemente flexible y general para permitir cargar múltiples tipos de modelos y sobretodo soportar modelos que sean de otras herramientas. Además, esta funcionalidad estaba muy ligada a modelos que se comporten de forma muy similar a la Ciclosporina y usen MonolixSuite. Tampoco era muy intuitivo para el farmacéutico cómo y qué debería cargar en cada campo solicitado. En la Figura 5.10 se observa un ejemplo de un modelo cargado en Finglix 1.0 para el fármaco Ciclosporina.

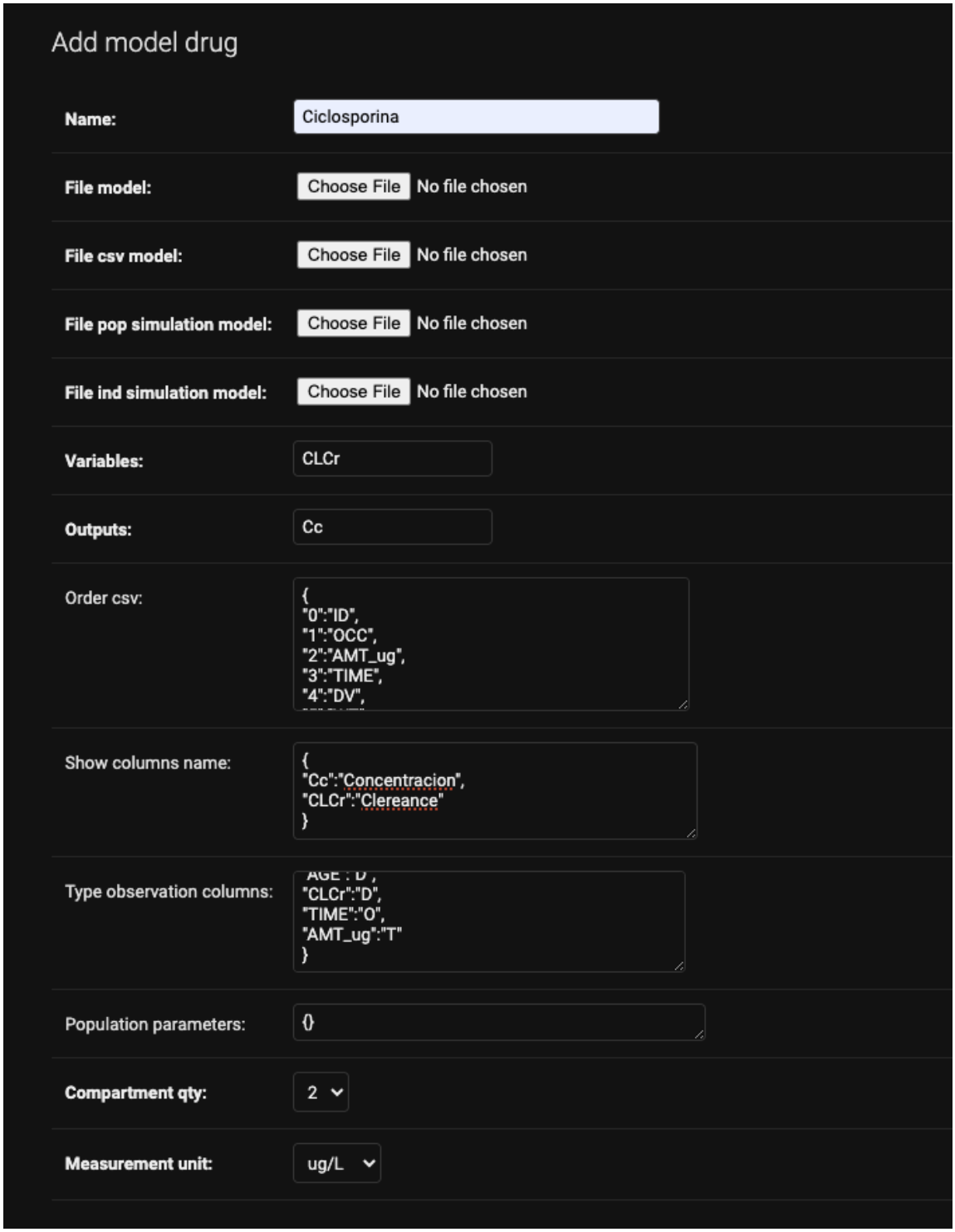


Figura 5.10: Cargar modelo en Finglix 1.0

Por este motivo, y que igualmente se tenía que rediseñar la estructura con la cual se representaba a los modelos para permitir agregar esa capa de abstracción que permite integrar otras herramientas para simular y estimar, fue que se modificó la carga de modelos.

Este proceso se hacía previamente utilizando el backoffice y por su facilidad a la hora de crear alta/baja/modificación de una entidad, y tomando en cuenta la nueva estructura de los modelos, se decidió que era la mejor manera seguir utilizándolo.

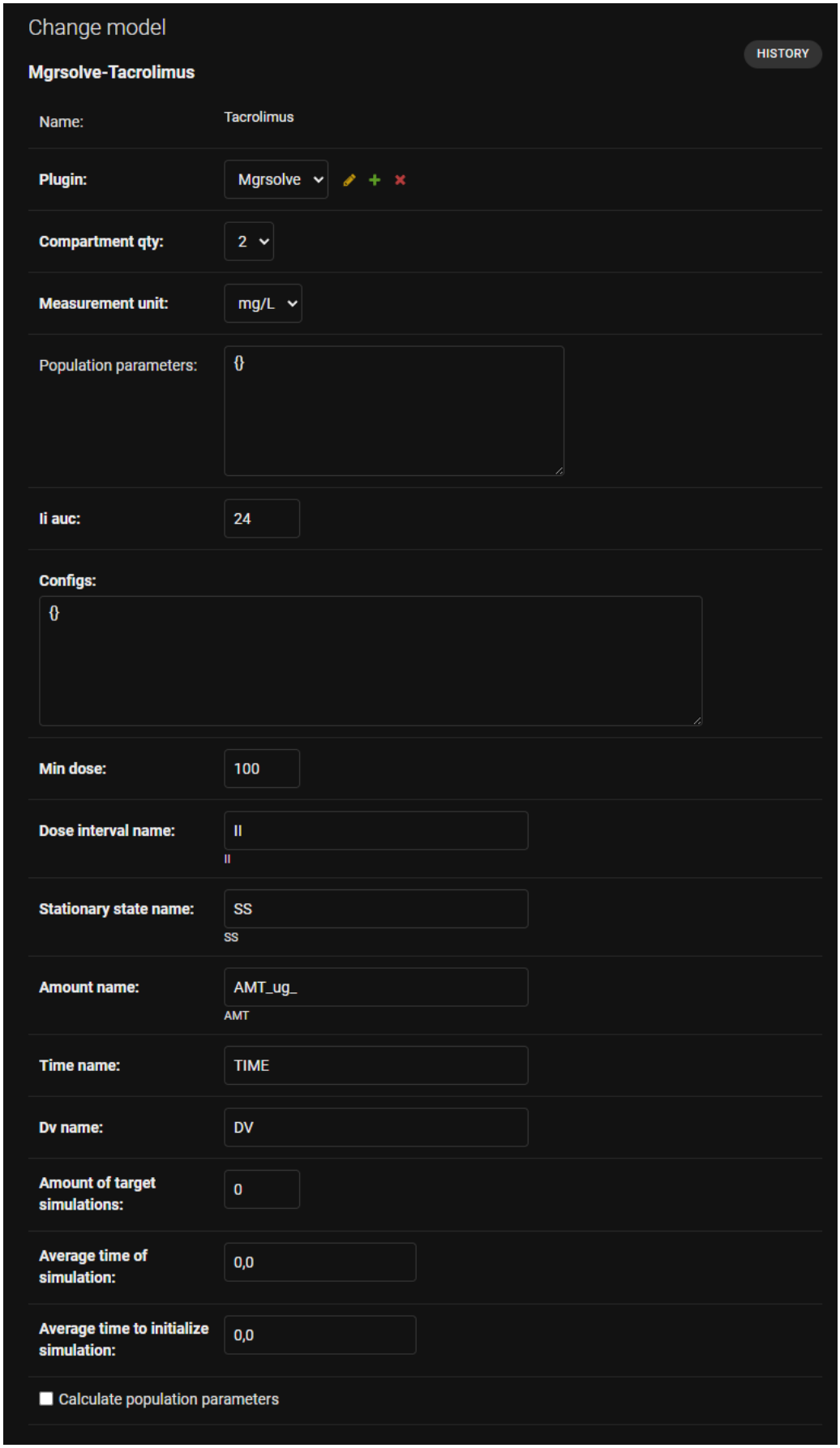


Figura 5.11: Cargar modelo Finglix 2.0

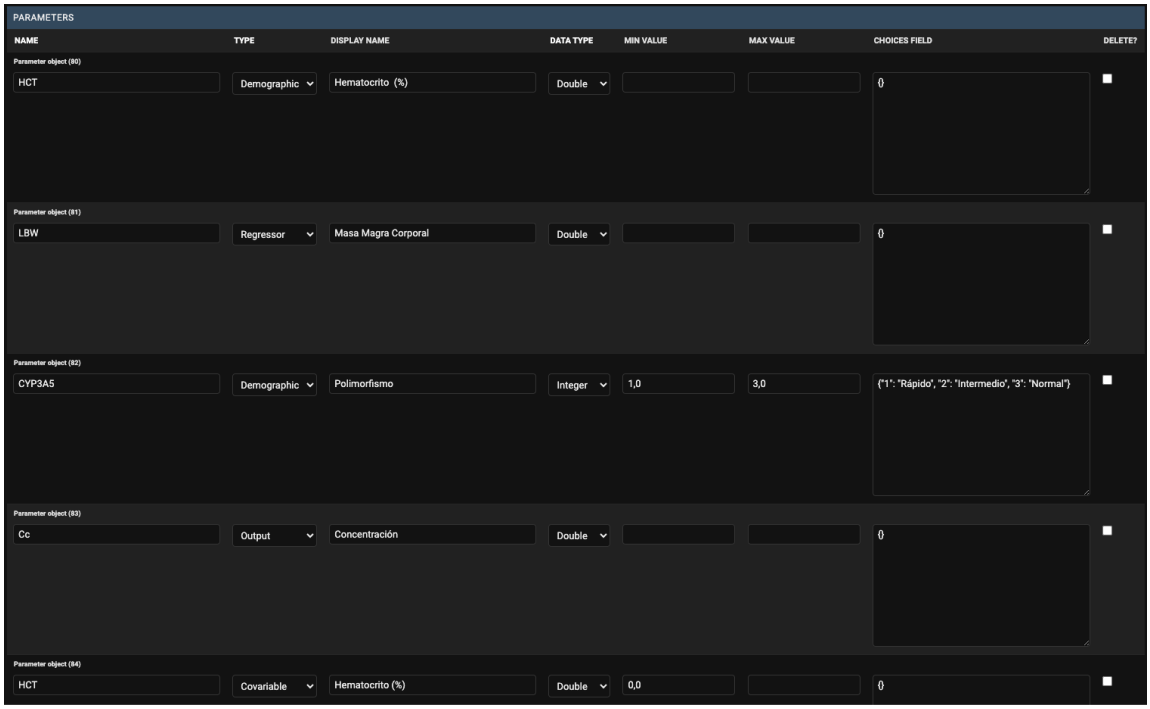


Figura 5.12: Cargar parámetros del modelo Finglix 2.0

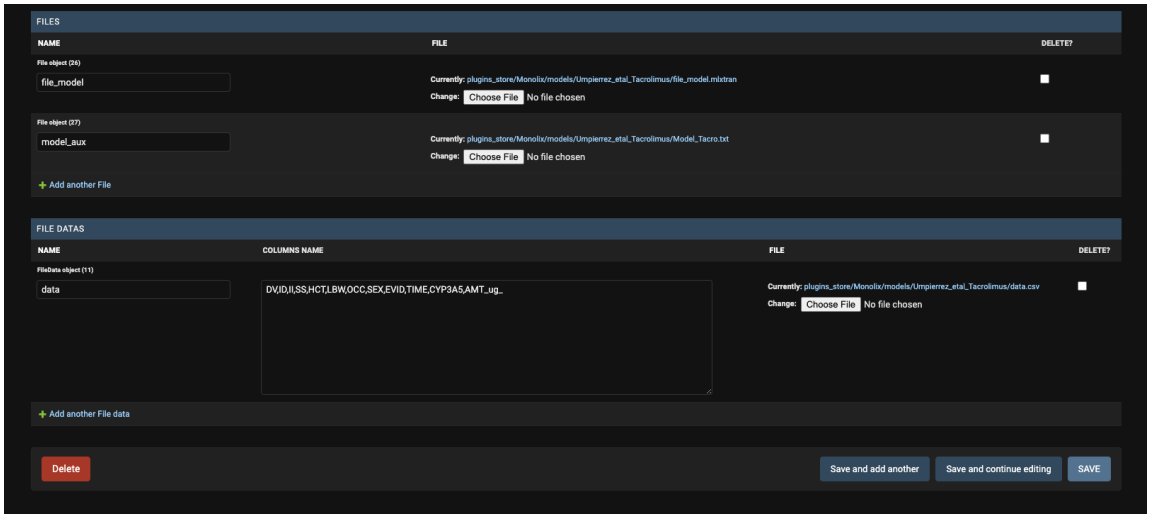


Figura 5.13: Cargar archivos del modelo Finglix 2.0

Como se puede apreciar en las Figuras 5.11, 5.12 y 5.13, en esta nueva versión hay varias diferencias respecto a la anterior. Probablemente lo más importante es la manera de ingresar todas las covariables, *outputs*, regresores, variables, etc. Antes se especificaban en campos que se debían ingresar o en formato JSON (ej: Type observation columns) o en formato texto separados por "," (ej: variables). En la Figura 5.12 se aprecia que ahora hay una sección entera dedicada a esto denominada *Parameters*, en donde en cada entrada se ingresa un parámetro indicando los siguientes campos:

- *Type* (*demogrpahic*, covariable, *regressor*, *output*, *treatment*): indica de qué tipo es el parámetro.
- *Display name* (opcional): indica el nombre que verá el usuario final.

- *Data type*: tipo de dato que está representando
- *Min value* (opcional): cota mínima que soporta
- *Max value* (opcional): cota máxima que soporta
- *Choices field* (opcional): es un JSON que permite asociar un valor a un nombre comprensible para el usuario, la clave es el valor que tomará el parámetro en el modelo y el valor es el nombre que se le asocia a dicho valor, y que el usuario podrá seleccionar.

Esto permite de una forma más sencilla e intuitiva ingresan múltiples "parámetros", y además con más información sobre sus naturalezas.

Además, se observó que ciertos parámetros eran comunes a todos los modelos, por lo tanto, para no ser repetitivos se asume que todos los modelos lo tendrán y no es necesario que lo agreguen en los parámetros. Sin embargo, se brinda la posibilidad al usuario de asociarles los nombres deseados, por ejemplo, el intervalo de administración por defecto se denomina *II*, pero si un modelo llama a esta variable intervalo, el usuario lo puede ingresar en el campo "*Dose interval name*". Los otros parámetros que son comunes a todos los modelos son: *Stationary state name* (SS), *Amount name* (AMT), *Time name* (TIME) y *Dv name* (DV).

Finalmente, en la Figura 5.13 se observa que los archivos de los modelos y conjuntos de datos también tienen su propia sección, esto da mayor flexibilidad en la cantidad de archivos que debe cargarse para cada modelo. Principalmente porque no necesariamente todas las herramientas van a utilizar la misma cantidad de archivos.

5.4.4. Soporte para regresores

Algunos modelos de MonolixSuite presentan una característica particular, poder usar regresores, que los modelos previamente usados en Finglix 1.0 no tenían. Por ejemplo, el primer modelo con regresores que se tomó como ejemplo es del fármaco Tacrolimus, que incluía un regresor denominado LBW que representaba la masa magra corporal.

El *script* de R que ejecutaba las simulaciones en Simulx fallaba si se quería utilizar regresores. Por esta razón fue necesario desarrollar un nuevo *script* en R de simulación.

En el *script* anterior se utilizaba la función llamada *Simulx* de la API de *Simulx*, donde recibe como parámetro todos los datos necesarios. La limitante de esta solución es que no permite ingresar regresores. Para poder simular mediante la API de Simulx con regresores se necesita utilizar otras funciones para definir cada tipo de dato ingresando en el modelo (por ejemplo, *defineTreatmentElement* para definir el tratamiento). Y al final, utilizar la función *runSimulation* para efectuar la simulación. Este método permite simular con regresores utilizando la función *defineRegressorElement*. Estos regresores se configuran al momento de cargar el modelo, en la sección parámetros se elige de tipo regresor.

5.5. Visualizar precisión en las predicciones

Una tarea muy importante para los farmacéuticos, es poder saber cómo se están comportando los modelos respecto a lo que realmente le sucede al paciente, con el fin de evaluar la adecuación de distintos modelos y determinar cuáles son mejores bajo determinadas circunstancias.

Por esta razón, Finglix 2.0 incorpora la funcionalidad de comparar mediante una gráfica, los valores observados (valores reales) con las simulaciones individuales y poblacionales. Las simulaciones individuales se calculan con los parámetros recién estimados luego de ingresar las observaciones, es decir con los parámetros ya ajustados. En la Figura 5.14 se muestra un ejemplo, en donde la curva verde representa la concentración estimada en función del tiempo usando parámetros poblacionales, la curva azul representa la concentración estimada en función del tiempo usando los parámetros estimados a partir de las observaciones y el punto rojo representa el dato observado en el paciente.

Figura 5.14: Comparación entre observaciones y simulaciones

5.6. Perfiles de simulación

Como se mencionó en la sección anterior, cuando se agregan observaciones de un paciente, el usuario puede visualizar la comparación entre las observaciones y las simulaciones poblacionales. Finglix 2.0 permite además que los plugins realicen más simulaciones, en caso de querer visualizar simulaciones con distintos parámetros. A estas otras simulaciones se las denomina perfiles.

Por este motivo, el contrato definido en la interfaz de los plugins, permite retornar el resultado de múltiples simulaciones, y estas serán mostradas por el *frontend* en un mismo gráfico para poder compararlas.

Por ejemplo, para el caso de la MonolixSuite, cuando se calculan parámetros, Monolix retorna los parámetros que considera más precisos, pero también, proporciona una cantidad configurable de "perfiles", los cuales son distintas combinaciones de parámetros que se acercan bastante a la predicción ideal. El plugin realizado para MonolixSuite permite que en la carga de modelo se indique cuántos perfiles extras se quieren simular. Esto se realiza agregando en el campo *config* de los modelos la siguiente pareja de clave-valor: "perfiles": *cantidad_de_perfiles*. En la Figura 5.15 se observa un ejemplo de simulaciones poblacionales, individuales y cinco perfiles.

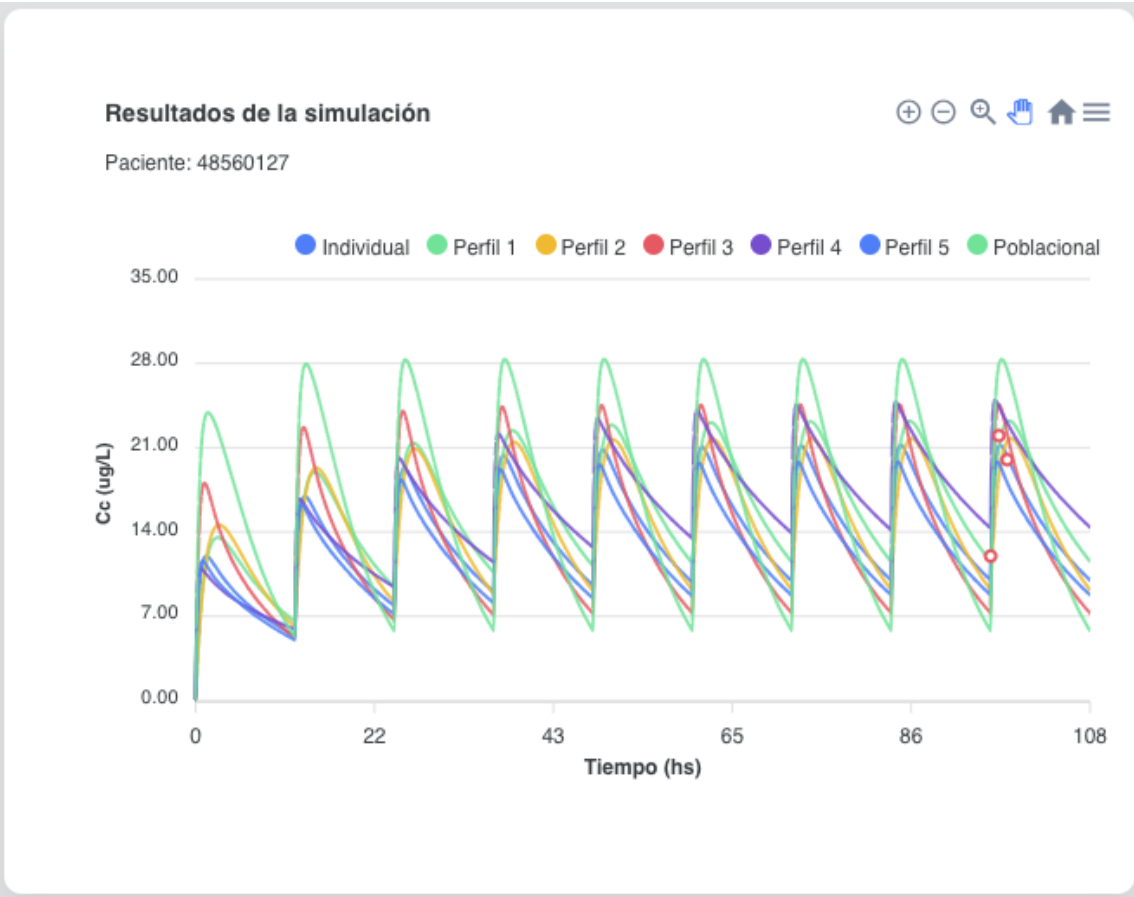


Figura 5.15: Simulaciones de perfiles

5.7. Despliegue en la nube

El despliegue se realizó en un *Virtual Private Server* provisto por Antel[4], utilizando la plataforma denominada Mi Nube[5].

[4] `https://www.antel.com.uy`
[5] `https://minubeantel.uy`

Durante el desarrollo, se configuraron *cron jobs* para iniciar los servicios automáticamente al reiniciar el servidor y scripts en bash encargados de inicializar estos procesos.

En la fase final del proyecto, se optó por un despliegue en producción utilizando Nginx como *proxy* inverso y servidor web, y uWSGI para el *backend*. Se mejoró el rendimiento de la plataforma configurando múltiples procesos de uWSGI, evitando el uso de hilos por incompatibilidad con la extensión Rpy2 de Python.

Detalles técnicos adicionales y scripts de despliegue se encuentran en el Apéndice D.1.

Capítulo 6

Pruebas y validaciones

En este capítulo, se describe el proceso de pruebas y validaciones llevado a cabo en la plataforma. Se detallan las pruebas realizadas, los resultados obtenidos y las acciones tomadas para abordar los aspectos identificados en base a los resultados.

6.1. Pruebas del sistema

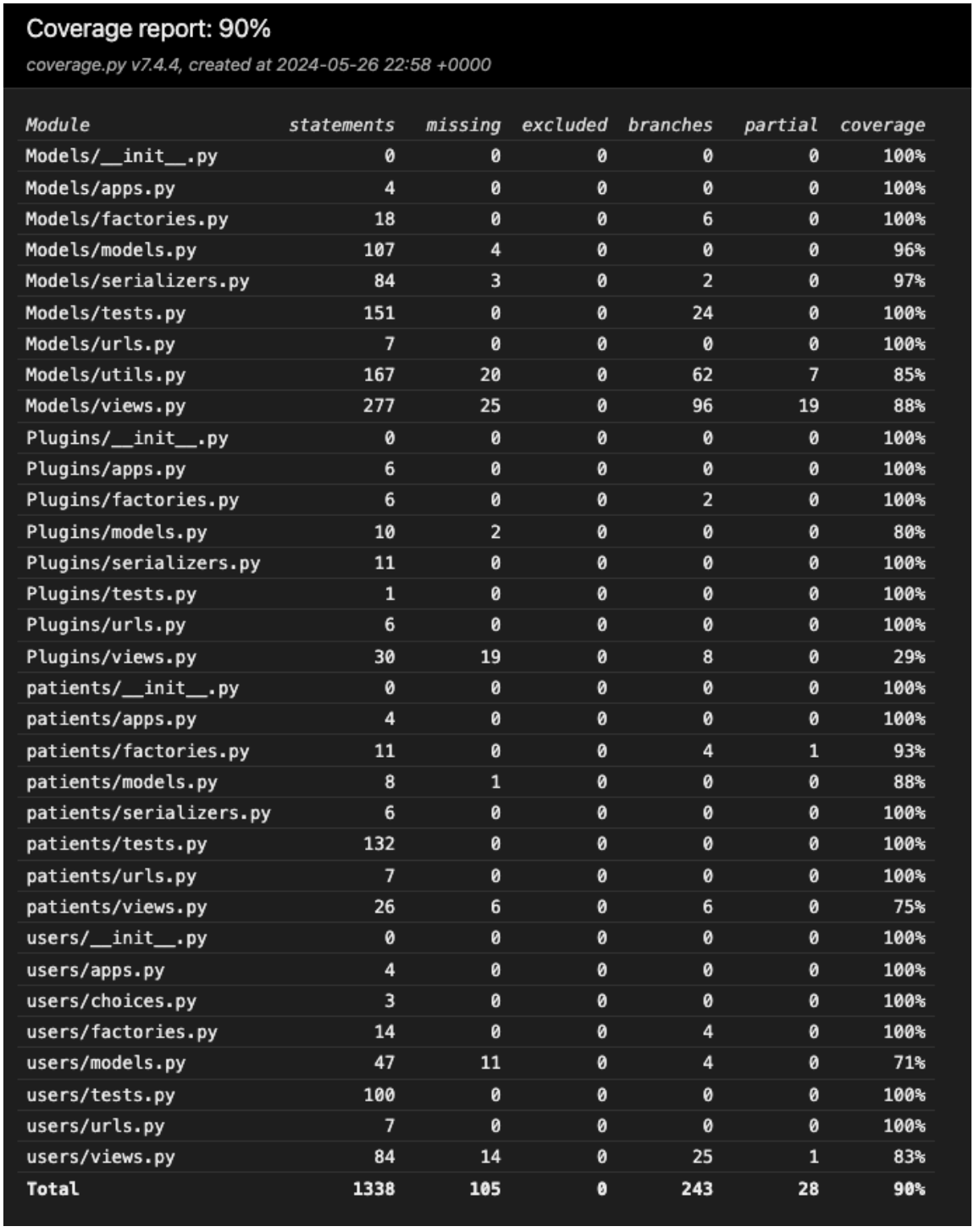
En esta sección se describen las principales pruebas que se realizaron en la plataforma tanto de forma automática como manual. En particular, se realizaron pruebas unitarias y análisis de código estático.

6.1.1. Pruebas unitarias

Se realizaron pruebas unitarias en el *backend*. Se utilizó la librería *pytest*. También se utilizó *factory-boy* para la generación de datos de prueba.

Se hicieron tests para la mayoría de las operaciones del *backend*, enviando datos a las operaciones y utilizando *mocking* para que los plugins devuelvan los datos solicitados por la operación y finalmente se verificó que la respuesta obtenida sea la esperada. También se probaron casos en los que se genera un error porque se ingresaron datos erróneos.

Se utilizó la librería *py-coverage*, la cual hace un análisis de qué partes del código son cubiertas por el test. Como se observa en la Figura 6.1 se logró un 90 % de cubrimiento sobre los archivos que interesaba *testear*.



Coverage report: 90%

coverage.py v7.4.4, created at 2024-05-26 22:58 +0000

Module	statements	missing	excluded	branches	partial	coverage
Models/__init__.py	0	0	0	0	0	100%
Models/apps.py	4	0	0	0	0	100%
Models/factories.py	18	0	0	6	0	100%
Models/models.py	107	4	0	0	0	96%
Models/serializers.py	84	3	0	2	0	97%
Models/tests.py	151	0	0	24	0	100%
Models/urls.py	7	0	0	0	0	100%
Models/utils.py	167	20	0	62	7	85%
Models/views.py	277	25	0	96	19	88%
Plugins/__init__.py	0	0	0	0	0	100%
Plugins/apps.py	6	0	0	0	0	100%
Plugins/factories.py	6	0	0	2	0	100%
Plugins/models.py	10	2	0	0	0	80%
Plugins/serializers.py	11	0	0	0	0	100%
Plugins/tests.py	1	0	0	0	0	100%
Plugins/urls.py	6	0	0	0	0	100%
Plugins/views.py	30	19	0	8	0	29%
patients/__init__.py	0	0	0	0	0	100%
patients/apps.py	4	0	0	0	0	100%
patients/factories.py	11	0	0	4	1	93%
patients/models.py	8	1	0	0	0	88%
patients/serializers.py	6	0	0	0	0	100%
patients/tests.py	132	0	0	0	0	100%
patients/urls.py	7	0	0	0	0	100%
patients/views.py	26	6	0	6	0	75%
users/__init__.py	0	0	0	0	0	100%
users/apps.py	4	0	0	0	0	100%
users/choices.py	3	0	0	0	0	100%
users/factories.py	14	0	0	4	0	100%
users/models.py	47	11	0	4	0	71%
users/tests.py	100	0	0	0	0	100%
users/urls.py	7	0	0	0	0	100%
users/views.py	84	14	0	25	1	83%
Total	**1338**	**105**	**0**	**243**	**28**	**90%**

Figura 6.1: Reporte del cubrimiento de código

6.2. Validación con usuarios

Para validar que el producto desarrollado satisface las necesidades de los potenciales usuarios, y cumple con sus expectativas, se llevó a cabo un proceso de validación presencial. Este proceso incluyó varias etapas clave que involucraron directamente a los usuarios finales, médicos, farmacéuticos y administradores, cada uno con acceso a funcionalidades específicas del sistema.

En primer lugar, se organizó una sesión de presentación del sistema, donde se mostraron en detalle las principales funcionalidades disponibles. Durante esta demostración, los usuarios pudieron ver cómo se realiza el alta de pacientes, cómo se llevan a cabo simulaciones poblacionales e individuales, y cómo se cargan observaciones y modelos de los fármacos. Esta presentación fue esencial para proporcionar una visión global del sistema y de sus funcionalidades.

Durante la presentación, se generó un ambiente interactivo en el que los usuarios pudieron hacer preguntas y expresar sus dudas de forma dinámica. Esto permitió aclarar conceptos y funcionalidades de inmediato, generando una buena comprensión del sistema. Además, se recopiló feedback valioso que ayudó a identificar áreas de mejora y ajustes necesarios en el sistema.

Tras la presentación, se concedió a los usuarios acceso al producto desplegado en la nube de Antel por una semana. Esto les permitió explorar y utilizar las funcionalidades del sistema de manera independiente, y poseer mucho más tiempo que si se realizaban las pruebas durante la presentación. Para apoyar su uso, se incluyeron tutoriales interactivos que guiaban a los usuarios a través de las principales funcionalidades[1][2][3]. Estos tutoriales fueron diseñados para ser intuitivos y proporcionar una asistencia práctica, facilitando el uso del sistema. Se utilizó la herramienta Supademo[4] la cual permite "crear una demo de producto de manera interactiva muy rápidamente", en la Figura 6.2 se muestra un ejemplo.

[1]Simulación: `https://app.supademo.com/demo/clwjpgxal01darnckpbjoaxic`
[2]Simulación objetivo: `https://app.supademo.com/demo/clwjrzv9401z2rnckcg4pejkv`
[3]Cargar observaciones: `https://app.supademo.com/demo/clwjx2twz02iqrnckiykte8ih`
[4]Supademo: `https://app.supademo.com/`

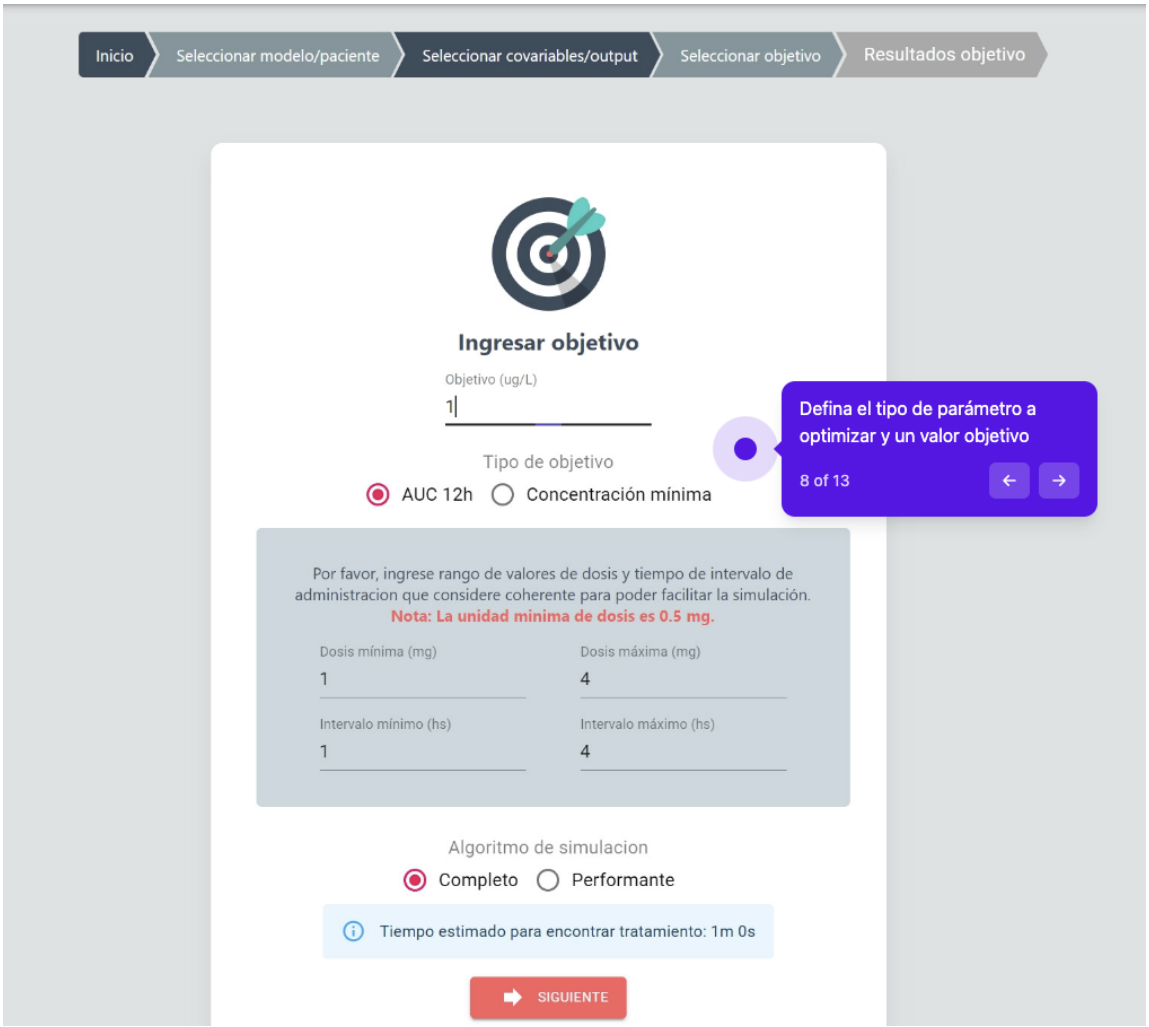


Figura 6.2: Supademo simulación objetivo

Para obtener una evaluación más detallada y estructurada, se enviaron encuestas a los usuarios después de la sesión de presentación. Como se puede apreciar en la Figura 6.3 las encuestas fueron diseñadas mediante *google forms* y tenían como objetivo capturar sus experiencias, percepciones y sugerencias sobre el sistema.

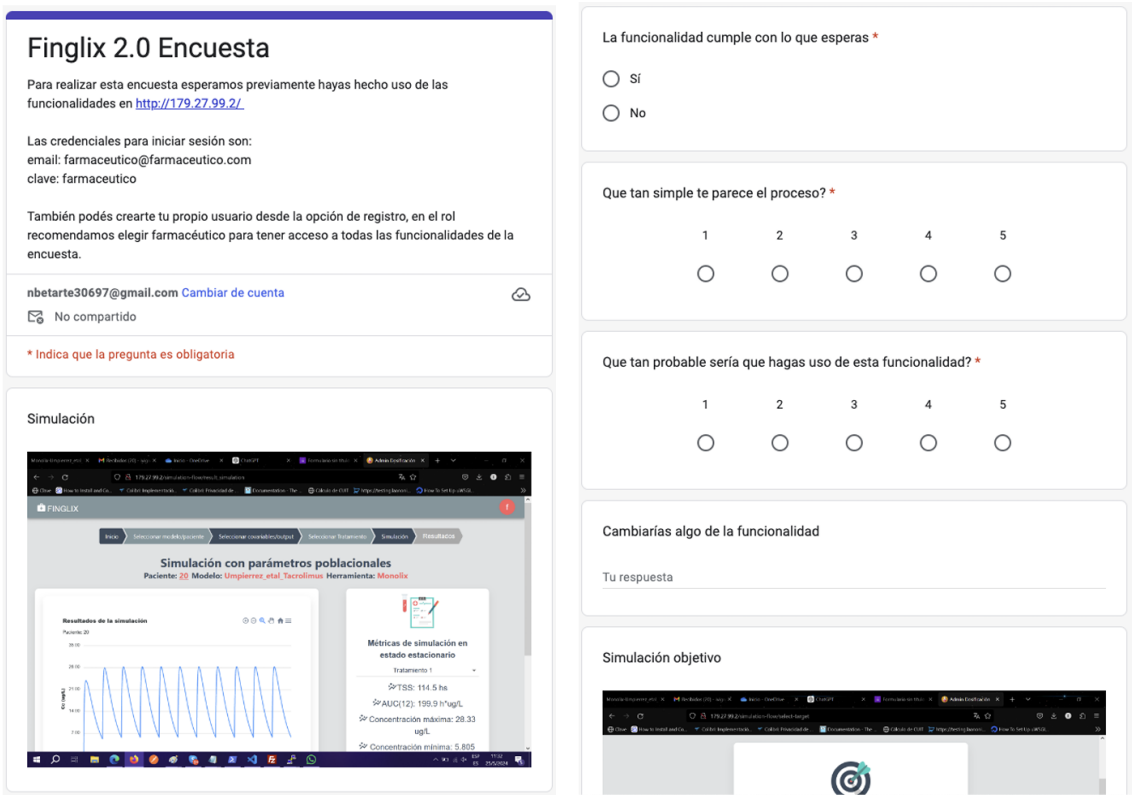


Figura 6.3: Encuesta google form

6.2.1. Detalles de la encuesta

En esta sección se detalla las preguntas formuladas en la encuesta realizada. Las encuestas se organizaron por funcionalidad abordando inicialmente las funcionalidades centrales del sistema y finalizando con preguntas más generales y abiertas del sistema.

Simulación

1. ¿La funcionalidad cumple con lo que esperas? **Sí** / **No**
2. ¿Qué tan simple te parece el proceso? (del uno al cinco)
3. ¿Qué tan probable sería que hagas uso de esta funcionalidad? (del uno al cinco)
4. ¿Cambiarías algo de la funcionalidad? (respuesta abierta)

Simulación objetivo

1. ¿La funcionalidad cumple con lo que esperas? **Sí** / **No**
2. ¿Qué tan simple te pareció el proceso? (del uno al cinco)
3. ¿Qué tan probable sería que utilices esta funcionalidad? (del uno al cinco)
4. ¿Cambiarías algo en la funcionalidad? (respuesta abierta)

Registro de Pacientes

1. ¿Qué tan simple te pareció el proceso? (del uno al cinco)

Cargar Observaciones

1. ¿La funcionalidad cumple con lo que esperas? **Sí** / **No**
2. ¿Qué tan simple te pareció el proceso? (del uno al cinco)
3. ¿Qué tan claro te pareció la forma de mostrar los datos generados a partir de la carga de observaciones? (del 1 al 5)
4. ¿Cambiarías algo en la funcionalidad? (respuesta abierta)

Cargar un Modelo - Rol Administrador

1. ¿La funcionalidad cumple con lo que esperas? **Sí** / **No**
2. ¿Qué tan simple te resulta el proceso? (del uno al cinco)
3. Respecto al proceso anterior de carga de modelos, ¿te pareció más simple? (en caso de no conocer el proceso anterior no es necesario contestar) (del uno al cinco)

Cargar un Plugin (Permite utilizar otras herramientas distintas a las de Lixsoft)

1. ¿Qué nivel de utilidad le ves a futuro a esta funcionalidad? (del uno al cinco)

Preguntas Generales
Estas preguntas son a nivel general y no de una funcionalidad específica.

1. ¿Qué tan sencillo te resulta Finglix? (del uno al cinco)
2. ¿Qué cambiarías de Finglix? (respuesta abierta)
3. ¿Crees que usarías Finglix en el futuro? **Sí** / **No**

6.3. Conclusiones de las pruebas realizadas

Durante la sesión presencial, los participantes realizaron preguntas interesantes y se respondieron sus dudas, surgiendo varios comentarios que podrían incorporarse en el sistema en el futuro.

Algunas de las mejoras propuestas incluyen:

- Permitir que los médicos registren pacientes en el sistema.
- Restringir a los médicos para que solo vean los modelos marcados como listos para usar, evitando el uso de modelos aún en desarrollo y no testeados.
- Mostrar diferentes modelos para cada fármaco y sub-población (por ejemplo, modelos para adultos y pediátricos).

- Unidades en los datos a ingresar de un paciente, ya ajustable modificando el nombre mostrado del parámetro.
- Definir valores por defecto a los parámetros del modelo: posibilitando que el sistema asigne valores cuando no se tiene el dato del paciente, definiendo criterios de para los valores por defecto para el caso de variables continuas (cómo peso corporal) y categóricas (cómo polimorfismos genéticos).
- Usar intervalos de administración fijos (4, 6, 8, 12 y 24 horas), especialmente útil para optimizar tratamientos en estado estacionario.
- Simulación objetivo: permitir al usuario seleccionar entre varias opciones, definir si la optimización es para estado estacionario o para una dosis específica, e incorporar el concepto de vincular modelos a fármacos.
- Alertar al usuario cuando falta un dato y ofrecer opciones (como valor por defecto, media, o el más común).
- Marcar la concentraciones mínima y máxima en la gráfica.
- Permitir ingresar observaciones por fármaco, en vez de por modelo. Es decir, que la observación sea transversal a todos los modelos del fármaco.
- Proveer al médico el "mejor modelo" establecido sin posibilidad de selección, optimizando así el uso del sistema.

Algunas de estas funcionalidades ya están presentes en el sistema y se explicó cómo utilizarlas. Las sugerencias no soportadas actualmente se agregarán a la lista de limitaciones y las mejoras viables en trabajos futuros del sistema. En el caso del registro de pacientes por parte de médicos y el control de acceso de los modelos para los médicos son mejoras que se incorporaron posteriormente a la sesión con el equipo de farmacéuticos.

6.4. Resultados de las encuestas

Los resultados de las encuestas sobre Finglix 2.0 indican una recepción generalmente positiva de sus funcionalidades, con varios aspectos destacados para cada función principal evaluada.

6.4.1. Simulaciones

En cuanto a las simulaciones, la mayoría de los usuarios considera que la funcionalidad cumple con sus expectativas. La simplicidad del proceso fue bien evaluada, aunque hubo algunas sugerencias para mejorar. La probabilidad de uso futuro también fue alta, con la mayoría de los participantes indicando que es muy probable que utilicen esta funcionalidad. Sin embargo, algunos usuarios sugirieron mejoras, como la modificación de la forma en que se agregan las observaciones y la unificación de las unidades y la codificación en los reportes. Sobre la unificación de las unidades cabe destacar que los nombres de las variables solicitadas son configurables al momento de cargar el modelo.

6.4.2. Simulación objetivo

La funcionalidad de encontrar la simulación objetivo recibió comentarios positivos en su mayoría. La simplicidad del proceso fue evaluada como adecuada, aunque algunos participantes indicaron la necesidad de ajustes menores. La utilidad futura de esta funcionalidad fue valorada de manera positiva, indicando que es una herramienta útil para los usuarios.

6.4.3. Registro de Pacientes

El registro de pacientes fue bien recibido, cumpliendo con las expectativas de los usuarios en términos generales, lo que sugiere una aceptación general en el flujo de trabajo.

6.4.4. Cargar Observaciones

En lo que respecta a cargar observaciones, dos de los seis encuestados indicaron que la funcionalidad es como lo esperaban, además, la mayoría consideraron el proceso relativamente simple pero las opiniones son divididas. Algunos comentarios sugirieron la necesidad de mejoras en la interfaz y la posibilidad de agregar funciones adicionales para facilitar el uso.

6.4.5. Carga de Modelos

La funcionalidad de cargar un modelo recibió comentarios positivos en cuanto a su cumplimiento de expectativas. Los usuarios evaluaron la simplicidad del proceso de manera favorable.

6.4.6. Cargar un Plugin

La carga de plugins fue evaluada como una funcionalidad útil y que cumple con las expectativas de los usuarios.

6.4.7. Sistema en General

En general, los usuarios encontraron que Finglix es una herramienta útil y que cumple con sus expectativas. La simplicidad de uso fue valorada de manera mixta, con algunos usuarios indicando que ciertas áreas podrían beneficiarse de una mayor claridad y mejor documentación. Las mejoras sugeridas incluyen modificaciones en las descripciones y una mayor coherencia en la interfaz. Todos los participantes indicaron que es probable que utilicen Finglix en el futuro, lo que sugiere una aceptación positiva general del sistema.

Capítulo 7

Etapas y gestión del proyecto

En este capítulo se describen las distintas etapas que tuvo el proyecto y se mencionan los principales desafíos que surgieron en cada una. Además, se especifica cómo fue la comunicación de todo el equipo y cuál fue la metodología de trabajo. Finalmente, se detallan las herramientas más relevantes que se utilizaron.

7.1. Etapas del proyecto

El proyecto comenzó con una fase de familiarización con el dominio y los términos específicos, seguido de un esfuerzo por lograr ejecutar el proyecto del equipo anterior. A medida que se avanzó, se fue profundizando en el funcionamiento del código y ampliando las competencias del equipo en las tecnologías utilizadas. Finalmente se continuo con el proceso desde la configuración inicial hasta la implementación, verificación y validación del sistema con los farmacéuticos. En la Figura 7.1 se encuentra el diagrama de Gantt con el detalle de las etapas del proyecto anteriormente mencionadas.

Id	Etapa	Tarea	Abril-4/2023	Mayo-5/2023	Junio-6/2023	Julio-7/2023	Agosto-8/2023	Septiembre-9/2023	Octubre-10/2023	Noviembre-11/2023	Diciembre-12/2023	Enero-1/2024	Febrero-2/2024	Marzo-3/2024	Abril-4/2024	Mayo-5/2024	Junio-6/2024
A	Inicio del proyecto y análisis preliminar																
B	Análisis, relevamiento y enfoque del																
C.1	Construcción del producto	Prototipos															
C.2		Entorno de desarollo															
C.3		Soporte para regresores															
C.4		Sistema de plugins															
C.5		Observaciones vs simulaciones, Perfiles															
C.6		Simulación objetivo															
C.7		Cambios menores															
D.1	Verificación y Validación del producto	Generacion de plugins Nlmixir2 y Mgrsolve															
D.2		Cargar modelos Ciclosporina, Tacrolimus y Vancomicina															
D.3		Pruebas unitarias															
D.4		Testing exploratorio															
D.5		Pruebas con usuarios															
E	Cierre del proyecto, documentación y																

Figura 7.1: Diagrama Gantt - Etapas del proyecto

7.1.1. Familiarización con el dominio

Al iniciar el proyecto, se invirtió tiempo en leer y comprender la literatura relevante para familiarizarse con los conceptos del dominio. Esto incluyó la revisión de la documentación generada por el equipo anterior y el apoyo del equipo de CIENFAR, quienes respondieron las dudas que fueron surgiendo y explicaron los conceptos clave para facilitar la comprensión del dominio.

Es importante destacar que ninguno de los integrantes del equipo tenía conocimientos previos en farmacología, un campo que presenta cierta complejidad debido a su vocabulario especializado. Este desafío no solo dificultó la comprensión del funcionamiento del sistema, sino también la comunicación efectiva con el equipo de farmacología. A diferencia de otros dominios, donde quizás se podría tener alguna experiencia previa, la farmacología representó un reto significativo para el equipo.

Para superar esta barrera, se adoptaron varias estrategias. En primer lugar, se mantuvieron múltiples reuniones con el equipo de CIENFAR, las cuales resultaron útiles para aclarar dudas, entender los términos técnicos y asegurar de que las expectativas y objetivos de ambas partes estuvieran alineados.

Con el tiempo, se fue mejorando la comprensión del dominio, lo que no solo facilitó una comunicación más fluida con el equipo de farmacéuticos, sino que también permitió un avance más rápido en el proyecto. Esta mayor comprensión se reflejó en un ritmo de trabajo más ágil y en la mejora de la calidad de los resultados obtenidos.

7.1.2. Configuración del proyecto existente

Tener una versión funcional del proyecto anterior requirió algunas configuraciones específicas no documentadas y se necesitó entender la implementación para lograr saber cómo cargar modelos. Sin embargo, el principal desafío residió en no poseer ejemplos funcionales de modelos ya manipulados para ser cargados en el sistema. Estos archivos se perdieron por la herramienta de comunicación utilizada por el proyecto anterior.

El equipo de farmacéuticos poseía una versión antigua de estos modelos, en concreto de la Ciclosporina, que no funcionaba correctamente al ser cargada al sistema siguiendo el manual de usuario, por lo tanto, fue necesario contactarse directamente con los integrantes del equipo anterior para solicitarles una versión del modelo de la Ciclosporina funcional con la última versión de Finglix hasta ese momento. El equipo anterior pudo facilitar el último modelo de la Ciclosporina funcional y finalmente, con esta versión, se pudo efectuar una simulación correctamente. Este proceso requirió entender todo el funcionamiento, tanto del código, MonolixSuite como de los archivos del modelo de la Ciclosporina para lograr su funcionamiento.

7.1.3. Aprendizaje de tecnologías

Dos de los miembros del equipo no tenían experiencia previa en las tecnologías utilizadas en el proyecto. Por lo tanto, se dedicó tiempo a aprender y familiarizarse con

estas herramientas mientras se configuraba el proyecto. También en algunas ocasiones se utilizó la técnica de programación a pares entre los menos familiarizados con las tecnologías.

7.1.4. Carga de modelos, estimaciones y simulaciones

Se comenzó por generar un entorno de desarrollo funcional, iniciando con la instalación de las tecnologías necesarias. Una vez que se logró ejecutar Finglix localmente, el siguiente paso fue ejecutar las distintas funcionalidades de la plataforma para verificar su correcto funcionamiento en el entorno de desarrollo.

Las funcionalidades que involucraban la utilización de MonolixSuite fueron las más desafiantes de que funcionen correctamente. Estas son: cargar un modelo, simular y estimar parámetros. Para poder ejecutar dichas funcionalidades, lo primero que se hizo fue cargar un modelo para el fármaco Ciclosporina. La carga de este modelo requería ingresar cuatro archivos: tres relacionados con el modelo y uno con el conjunto de datos. No obstante, no se pudo acceder a las versiones finales y funcionales de estos archivos que utilizó el equipo anterior. Solo obtuvimos versiones intermedias que necesitaban algunos ajustes para funcionar, lo que requirió un análisis más profundo de cómo estaban implementadas estas funcionalidades.

Por otra parte, también surgieron inconvenientes al querer ejecutar rutinas de R, ya que se se generaba un error que causaba que el *backend* finalizara su ejecución. El problema residía en la versión de la librería de *python rpy2*, que cumple la función de ejecutar código R embebido en Python. Fue necesario utilizar una versión más nueva: 3.5.1.

Además, se necesitaba una licencia para MonolixSuite, ya que es un *software* de pago, cada uno de los integrantes pidió al equipo de Lixoft la obtención de una licencia académica. Dos de los integrantes obtuvieron respuesta y licencias estudiantiles con un año de vigencia.

Una vez que el entorno estuvo completamente operativo, el siguiente paso fue profundizar en el proceso de carga de modelos. Un modelo es funcional si permite realizar simulaciones. Sin embargo, no es posible realizar simulaciones si no se estiman previamente los parámetros poblacionales o individuales. Por esta razón, el primer objetivo fue lograr la estimación de parámetros poblacionales, seguida de la prueba de simulaciones poblacional. Posteriormente, también se lograron estimar los parámetros individuales, lo que desbloqueó la funcionalidad de simulación individual y la capacidad de cargar observaciones.

7.1.5. Publicación del sistema y recolección de requisitos

Con una base de conocimiento establecida y una comprensión más amplia del dominio, se publicó en el VPS proporcionado por Antel el sistema, con el fin que el equipo del CIENFAR tenga acceso. Este despliegue fue uno de los desafíos marcados por el equipo previo ya que requería utilizar una instalación anterior de MonolixSuite. Además se observó que el despliegue realizado no era el adecuado para un entorno

de producción. Todo esto lleva a que el despliegue adecuado y funcional en Antel represente un desafío, el cual se profundizó en la Sección 5.7.

Una vez estabilizado el sistema, se llevaron a cabo numerosas reuniones para comprender las necesidades y los requisitos presentados por los usuarios. Uno de los primeros requisitos fue el soporte de regresores, lo cual llevó a trabajar con un modelo de Tacrolimus proporcionado por los farmacéuticos. También se evaluaron las prioridades, el grado de incertidumbre, y el impacto de cada requisito.

7.1.6. Reutilización o un nuevo desarrollo

Se evaluó la decisión de si era mejor construir un sistema desde cero o reutilizar el existente. Se optó por la reutilización, lo que implicó rehacer gran parte del *backend* debido a su alto acoplamiento con la Ciclosporina y una única herramienta de simulación. Aunque el proyecto anterior fue útil como base, se adaptó el sistema para que fuera más flexible a otras herramientas de simulación y plugins. Además, el diseño de la interacción con el usuario se replicó para mantener la coherencia en la experiencia general.

7.1.7. Implementación de plugins

El desarrollo de esta capa de abstracción fue detallado en la Sección 5.4.1. Como complemento, se realizaron tres prototipos de plugins, para MonolixSuite, Nlmixr2 y Mrgsolve. Luego, debido al código ya implementado, y por resultar MonolixSuite la herramienta más utilizada por el equipo del CIENFAR actualmente, se decidió desarrollar en mayor profundidad el plugin para esta herramienta. Otra razón de esta decisión, es que no se tenía el conocimiento y la experiencia sobre las otras herramientas por parte del CIENFAR, lo cual dificultaba obtener múltiples modelos y validar que los resultados fueran correctos.

En cuanto a implementación esta etapa es la que requirió más tiempo, por los cambios que implicó a nivel de arquitectura, y por ende del código. Por un lado, fue necesario desacoplar el código de Finglix del modelo de la Ciclosporina. Esto se debió a que si bien en Finglix 1.0 se intentó hacer una solución para cargar otros modelos, la realidad es que había ciertas partes que eran muy específicas para el modelo de la Ciclosporina en concreto. Por ejemplo, estaba *hardcodeado* en el código la inserción de *"dummy data"*, especifico de la Ciclosporina, en los CSV usados por MonolixSuite para poder estimar parámetros individuales. Otro ejemplo fueron los *scripts* de R utilizados para estimar y simular que, por cómo estaban implementados, no era posible cargar modelos con regresores.

Una vez lograda la compatibilidad con otros modelos, fue necesario desacoplar a Finglix de la dependencia estrecha con MonolixSuite. Fue necesario invertir tiempo considerable en comprender el flujo de trabajo para las simulaciones y la carga de observaciones, para tener una imagen clara en alto nivel de estos procesos. El objetivo fue lograr un nivel de abstracción que permitiera al sistema operar con otros modelos y herramientas. Esto implicó experimentar con diversas herramientas y modelos para diseñar una solución no acoplada a una única herramienta.

Una vez construida esta capa de abstracción y con el plugin de MonolixSuite funcional, se procedió a realizar plugins para Mrgsolve y Nlmxir2. En particular la instalación de Nlmixr2 implicó numerosos intentos. Esto se debió a que si bien la página oficial destaca un único comando introducido desde R para instalarse, en la práctica tiene muchas dependencias que hacen que la instalación no sea tan sencilla.

El listado de dependencias que es necesario instalar se encuentra en el Apéndice B. Para obtener cada una de las dependencias se hizo a través de ensayo y error, es decir, se intentaba la instalación, cuando fallaba, se veía cuál dependencia no estaba y se cortaba el bucle en el que entraba la instalación. Luego se procedía a instalar dicha dependencia y nuevamente se repitió el proceso hasta que se pudo obtener Nlmixr2 funcional en R.

También, ya avanzada la implementación, se presenció virtualmente una charla dada por el cotutor de esta tesis, Manuel Ibarra, en el marco del *Webinar* sobre *Model-Informed Precision Dosing*. Su charla fue sobre *"International Perspectives on Model-Informed Precision Dosing: From the Data to the Patients"* [52]. Cabe destacar que en esta charla se mostraron capturas de Finglix y se comentó en lo que se estaba trabajando en este proyecto.

7.1.8. Simulación objetivo

La implementación de la funcionalidad de simulación objetivo requirió una etapa de investigación y experimentación. Dado el alto costo de realizar muchas simulaciones y las limitaciones para paralelizar en R, se desarrolló un algoritmo que optimizara el proceso de encontrar el tratamiento óptimo de manera más rápida.

7.1.9. Mejoras y documentación

Se implementaron mejoras en la carga de observaciones y se documentó gran parte del proceso a medida que se avanzaba. Esto incluyó decisiones tomadas, lecciones aprendidas, diagramas de arquitectura, modelo de dominio y estándares de plugins. Al finalizar la implementación, se realizaron verificaciones y correcciones menores del sistema.

7.1.10. Verificación y validación del sistema

Durante el desarrollo, se fue probando continuamente lo implementado para verificar su correcto funcionamiento. Dada la dificultad del dominio, se mantuvieron reuniones frecuentes con el equipo de CIENFAR para ajustar errores y aclarar dudas, como unidades de medida o nombres de tipos de variables en los modelos. Para probar el sistema, se aprovechó la automatización del despliegue en Antel, lo que permitió que el equipo del CIENFAR probara las funcionalidades en desarrollo. Además, se realizaron pruebas unitarias para verificar que las unidades individuales del código funcionaran correctamente.

Finalmente, se realizó una reunión con los farmacéuticos para validar el producto. Esto permitió validar que el sistema cumpliera con sus necesidades y expectativas. La validación incluyó:

- Revisión de requisitos: confirmando que los requisitos fueran satisfechos.
- Pruebas de usuario: se presentó ejemplos del funcionamiento del sistema y se dio acceso a los farmacéuticos para que utilizaran el sistema y proporcionaron *feedback*.
- Ajustes finales: realización de ajustes basados en el *feedback* recibido para mejorar la usabilidad y funcionalidad del sistema.

El proyecto pasó por varias etapas críticas desde la familiarización con el dominio hasta la validación del sistema con los usuarios finales. Cada fase aportó conocimientos valiosos y mejoras que permitieron desarrollar un sistema funcional.

7.2. Metodología

La metodología de trabajo empleada en el proyecto se basó en la distribución de roles según las fortalezas individuales de cada miembro del equipo. Uno de los integrantes, por su dominio de las tecnologías utilizadas, fue el principal responsable en ajustar la arquitectura del proyecto y resolver problemas inherentes a dichas tecnologías, también se encargó de la coordinación de reuniones con la tutora y con los interesados en el proyecto.

Por otro lado, otro miembro del equipo se dedicó principalmente a la creación de entornos de despliegue, es decir la instalación de la aplicación y armado de ambientes, configuración, creación de máquinas virtuales e instalación y despliegue en Antel. Las principales tareas del tercer integrante fueron: analizar, planear y diseñar el proyecto e investigar la comunicación con las herramientas de simulación.

Aunque cada uno tuvo un enfoque principal, todos participaron en mayor o menor medida en todas las etapas del proyecto. Además, se utilizó la programación de a pares, principalmente en la fase de investigación, para desarrollar los plugins y en el diseño de la funcionalidad simulación objetivo.

7.3. Comunicación

Para la comunicación se utilizó principalmente la herramienta Mattermost, es una plataforma de mensajería y colaboración de código abierto, la cual facilitó la comunicación entre todas las partes. Se creó un servidor de Mattermost llamado Finglix, en el cual participó el equipo de desarrollo, los tutores y el equipo del CIENFAR. En total ocho personas. Además, se crearon múltiples canales para poder segmentar las conversaciones por temas. A diferencia de otras plataformas similares de mensajería como Slack, Mattermost permite en su versión gratuita persistir todos los mensajes indefinidamente, una funcionalidad valiosa para futuros equipos que deseen entender

detalladamente algunas decisiones.

Para la comunicación interna del grupo de desarrollo se utilizó principalmente Discord. Se creó un servidor con dos canales: *Bugs* y *General*. En *Bugs* se documentaban errores, problemas o posibles mejoras, y en *General* para el resto de información necesaria. Aquí se hacían las reuniones más frecuentes, en las cuales generalmente al principio se discutía algún problema a resolver y luego se dividían las tareas de cada integrante o se trabajaba en conjunto.

Además, se utilizó la mensajería por correo electrónico para situaciones puntuales, principalmente para confirmar u organizar determinadas reuniones.

Finalmente, se utilizó WhatsApp para aquellas conversaciones más informales que no agregan valor persistir en Mattermost. Se creó un grupo entre los tres integrantes de esta tesis, donde principalmente se organizaron reuniones y se usó para comentar avances o problemas específicos.

7.4. Herramientas utilizadas

En las siguientes subsecciones se describen las diversas herramientas y tecnologías empleadas durante el desarrollo del proyecto. Se abordarán herramientas utilizadas para la virtualización, la gestión de repositorios, la elección de entornos de desarrollo, editores de código, la creación de diagramas y la automatización de tareas.

7.4.1. Virtualbox

Al inicio del proyecto se pensó en crear una máquina virtual Linux en donde se instalaría Finglix, para que pudiera ser utilizado por el equipo del CIENFAR.

También se pensó en esta máquina virtual como entorno de desarrollo, ya que Finglix 1.0 está pensado para ser ejecutada en Linux o Mac OS e introducir Windows como entorno de desarrollo podría generar que el resultado final no fuera el adecuado. Por ejemplo, Celery no existe oficialmente para Windows, esto implica que si se trabaja en dicho sistema operativo se iba a tener una variabilidad extra en el entorno de desarrollo en comparación con el entorno de despliegue (Linux).

Se creó entonces una máquina virtual con Kubuntu la cual automáticamente iniciaba el navegador web al iniciar la máquina con el *frontend* de Finglix (y por detrás se iniciaba también Celery y el *backend*).

Sin embargo, en la práctica no resultó tan útil, ya que el cambio arquitectónico significó un cambio en la manera de ejecutar la aplicación (instalación de dependencias, plugins, base de datos) lo cual por cada cambio se debía entregar una VM nueva, las cuales además tenían un tamaño en torno a 8GB y se debía explicar cómo iniciar la VM al equipo de CIENFAR [53].

Finalmente, la exitosa instalación del proyecto en la nube Antel hizo que el uso de la máquina virtual fuera innecesario, ya que en este caso se hacían las actualizaciones

en Antel y quienes probaban la aplicación solo debían acceder a una URL desde el navegador web.

7.4.2. Manejo de repositorios

Se utilizó el gestor de repositorios GitLab que permite que el equipo colabore en el código del sistema. En el desarrollo del proyecto Finglix 1.0, se utilizaron dos repositorios separados, *backend* y *frontend*. Sin embargo, esto genera un desafío importante relacionado con la gestión de múltiples ramas y la necesidad de garantizar la sincronización de los cambios entre el *backend* y el *frontend*. Por ejemplo, cuando se modificaba una dirección de un *endpoint* en el *backend*, era esencial que el *frontend* estuviera al tanto de dicha modificación.

Para abordar esta problemática, se consideró la opción de crear ramas similares tanto en el *frontend* como en el *backend*. No obstante, se llegó a la conclusión de que una solución más ordenada sería unificar ambos repositorios en un solo repositorio con dos proyectos independientes: uno destinado al *frontend* y otro al *backend*. Estos dos proyectos debían ser capaces de ejecutarse y funcionar de manera coordinada.

Con el fin de establecer un proceso claro para la incorporación de cambios, se definió un protocolo. En este protocolo, se destaca la presencia de una rama principal, denominada "main", que debe mantenerse siempre en un estado funcional. Cuando se planifica la implementación de un nuevo cambio, se inicia la creación de una nueva rama a partir de "main". En esta rama, se procede a llevar a cabo la implementación del cambio propuesto, luego de implementados los cambios, se deben traer los cambios de la rama principal a la rama que contiene la implementación para posteriormente fusionarla con la rama principal. Una vez que se ha verificado su correcto funcionamiento, se procede a generar un "*pull request*". Este *pull request* deberá ser revisado y evaluado por los demás miembros del equipo antes de su incorporación al proyecto principal. Este enfoque de desarrollo estructurado y colaborativo permite mantener un alto nivel de calidad y cohesión en el proyecto Finglix [54].

7.4.3. Windows Subsystem for Linux

Finglix 1.0 fue pensado para ejecutarse en Linux, en particular Celery no ofrece soporte oficial para Windows. Dado lo anterior, se creó un Subsistema de Windows para Linux (WSL) que permite a los desarrolladores instalar una distribución de Linux (como Ubuntu, OpenSUSE, Kali, Debian, Arch Linux, etc.) y usar aplicaciones, utilidades y herramientas de línea de comandos de Bash directamente en Windows, sin modificar, sin la sobrecarga de una máquina virtual tradicional o una configuración de arranque dual [55].

Además, permite exportar imágenes de WSL. Por lo tanto, se utilizó WSL con una imagen de Ubuntu 22.04.3, instalándose todas las dependencias necesarias, incluyendo MonolixSuite, el cual como se mencionó en la sección 5.7 requiere de una interfaz gráfica. Esto no fue problema, ya que WSL a partir de su versión 2, posee compatibilidad con aplicaciones con GUI (X11 y Wayland). Finalmente, se corroboró el

correcto funcionamiento y luego se exportó la imagen para que sea descargable por el resto de los miembros del equipo. Esta fue almacenada en el repositorio de tesis alojado en la nube de Microsoft (OneDrive) [56].

7.4.4. Visual Studio Code

Como IDE (Entorno de desarrollo integrado) se utilizó Visual Studio Code, que es un IDE muy popular y que tiene buena integración con las tecnologías utilizadas. [57].

7.4.5. Lucidchart

Se utilizó para crear diagramas, ya que facilita la elaboración de diagramas de flujo, diagramas de arquitectura y otros gráficos necesarios para visualizar el diseño y la estructura del proyecto de forma colaborativa [58].

7.4.6. Onedrive

Onedrive fue utilizado para almacenar todos los archivos del sistema. Esta herramienta ofrece un almacenamiento en la nube seguro y accesible para todos los miembros del equipo, asegurando que los documentos y archivos del proyecto estén centralizados y respaldados. Gracias al uso del correo académico, se utilizó el plan para estudiantes que incluye principalmente cien GB de almacenamiento y las herramientas de office 365 que fueron utilizadas para toda la documentación intermedia [59].

7.4.7. Bash

Los *scripts* en *bash* permiten automatizar tareas y procesos ya que proporcionan una forma de encadenar comandos en un único archivo. Estos *scripts* pueden incluir estructuras de control, variables, funciones y otros elementos de programación [60].

En el contexto del proyecto, se utilizaron *scripts* para el despliegue en el VPS de Antel. Durante la etapa de desarrollo se programaron *scripts* en *bash* encargados de correr el *frontend* y *backend*. En la etapa final se programó un *script* con una secuencia de comandos encargada de automatizar el despliegue en su versión productiva, a grandes rasgos, bajar los cambios de *git* y generar una versión compilada del proyecto *backend* y *frontend* y publicarla.

Capítulo 8

Conclusiones y trabajos a futuro

Este capítulo presenta las conclusiones relacionadas con el proyecto en su conjunto, los resultados obtenidos, las dificultades encontradas, potenciales mejoras y trabajo a futuro.

8.1. Conclusiones generales

El objetivo de permitir la integración de Finglix con otras herramientas de simulación y estimación, más allá de MonolixSuite, se cumplió con éxito. Se rediseñó el *backend* incorporando un sistema de plugins. Además, se desarrolló el plugin para MonolixSuite ampliando el soporte para diversos modelos, lo que permitió operar con modelos que incluyen regresores. Este plugin se probó con modelos para fármacos como Ciclosporina, Tacrolimus y Vancomicina. Además, se realizaron prototipos de plugins con Mrgsolve y Nlmixr, validando la capacidad de la nueva arquitectura para operar con herramientas distintas a MonolixSuite.

Se simplificó el proceso de carga de modelos, haciéndolo más flexible agregando opciones adicionales como renombrado de variables, establecimiento de máximos y mínimos para parámetros, y soporte para listas de posibles valores.

Se desarrollaron las siguientes funcionalidades nuevas: "simulación objetivo", "comparar observaciones y simulaciones" y "simular perfiles". Además, se realizaron mejoras sobre las siguientes funcionalidades existentes: "simulaciones poblacionales", "simulaciones individuales", "cargar observaciones", "cargar modelos", "cargar plugins".

La colaboración con el equipo del CIENFAR fue esencial para el avance del proyecto, adquiriendo un buen conocimiento del dominio y obteniendo *feedback* continuo. Además el sistema se publicó en la nube para facilitar el acceso y la colaboración con este equipo. Se aplicaron mejoras de procesos en el trascurso del desarrollo, como la fusión de repositorios de *backend* y *frontend*, la automatización del despliegue en la nube y la definición de un entorno de desarrollo uniforme utilizando WSL. Estas mejoras en los procesos facilitaron la incorporación de funcionalidades y agilizaron el desarrollo del proyecto.

Finalmente, el proyecto se gestionó durante aproximadamente catorce meses, logrando cumplir con los objetivos propuestos. Se mejoró la plataforma Finglix en términos

de integración, soporte de modelos, funcionalidades avanzadas y usabilidad, dejando una base sólida para futuros desarrollos.

8.2. Limitaciones

En esta sección se mencionan las limitaciones del sistema, derivadas de las decisiones tomadas durante su construcción. Estas limitaciones pueden implicar esfuerzos de implementación y requerir cambios más profundos en el sistema.

- **Soporte para nuevas herramientas de simulación**: La integración de nuevas herramientas de simulación requiere la implementación de plugins adicionales. Algunos de estos plugins pueden necesitar la instalación manual de software adicional en el servidor. La instalación de este software no se encuentra automatizada.
- **Compatibilidad con dosis variables**: Actualmente el sistema al momento de cargar un tratamiento solo soporta un intervalo de administración del fármaco fijo (por ejemplo, cada ocho horas) por lo que no es posible cargar tratamientos en los que los intervalos entre dosis no sean siempre iguales.
- **Dosis Continua**: El sistema no ofrece soporte para la administración de dosis continua.
- **Dificultades en el testeo de plugins y modelos**: A la hora de desarrollar un nuevo plugin, o modificar uno existente, probar su correcto funcionamiento puede ser complejo. Esto se debe a que el sistema no cuenta con un mecanismo para mostrar errores de ejecución del plugin. También se ve afectado el proceso de testeo de modelos ya que el usuario no tiene acceso a la salida generada por R, donde muchas veces esta salida indica los posibles errores en el modelo y observaciones cargadas del paciente. En ambos casos, para visualizar los errores se debe ejecutar el *backend* desde una terminal.
- **Soporte para parámetros calculados**: En el contexto de las simulaciones, existen parámetros que requieren cálculos a partir de otros parámetros que carga el usuario. A modo de ejemplo, calcular el IMC a partir de la altura y el peso. El sistema actual no soporta la inclusión de parámetros en los modelos que se calculen a partir de otros parámetros.

8.3. Trabajo futuro

En base a las limitaciones, a la retroalimentación recibida en las instancias de validación detallada en la Sección 6.3 y los requisitos que quedaron fuera del alcance, se identificaron las siguientes líneas de trabajo a futuro.

- **Manejo de parámetros en Modelos**
  - Actualmente, el usuario debe ingresar todos los valores requeridos. Se plantea entonces como trabajo a futuro permitir que se carguen valores por defecto para las variables de los modelos.

- Relacionado al punto anterior, se podría definir funciones para asignar valores por defecto a las variables de los modelos, por ejemplo, dependiendo de cada caso tomar máximo, mínimo, promedio o media basándose en los datos (CSV en el caso de MonolixSuite) que el farmacéutico carga al momento de cargar el modelo en el sistema.
- Permitir a los plugins agregar nuevos tipos de parámetros. Dado que para poder cargar modelos que soporten regresores en MonolixSuite, se agregó un nuevo tipo de parámetro denominado "regresor", puede resultar conveniente tener la flexibilidad de que otras herramientas puedan especificar nuevos tipos también. En este sentido, los plugins podrían tener la capacidad de especificar estos parámetros.

- **Simulación de Tratamientos**
  - Actualmente, el sistema recomienda un tratamiento con una dosis e intervalo de administración fijo. Resultaría interesante que el tratamiento recomendado pueda incluir dosis e intervalo de administración variados.
  - Actualmente, el sistema prueba con todos los valores del intervalo discretizando en horas. Se propone acotar la definición de posibles intervalos de dosis para la optimización de dosis (cuatro, seis, ocho, doce y 24 horas).

- **Resultados de simulación**
  - En la simulación objetivo mostrar no solo el mejor tratamiento, sino otros que también pueden ser buenas opciones. Además, se podría permitir al usuario definir cuántos de estos tratamientos posibles mostrar.

- **Gestión de fármacos y modelos**
  - Implementar la funcionalidad para cargar fármacos en el sistema y poder asociar los modelos a ellos.
  - Relacionado al punto anterior se podría permitir la entrada de observaciones por fármaco, no solo por modelo. Actualmente el sistema requiere que las observaciones se carguen por modelo.

- **Interfaz de usuario**
  - Marcar los valores máximo y mínimo en las gráficas generadas por las simulaciones.

- **Gestión de Errores**
  - Implementar un log de gestión de errores para los plugins utilizados en la aplicación. Este log debería escribir la salida de la consola del *backend*. Esto resulta útil para depurar el correcto funcionamiento de los modelos nuevos que se carguen al sistema. Por ejemplo, permitiría ver la salida de la ejecución del código de R, utilizado en los plugins.

- **Integración con Historia Clínica**

- Desarrollar la integración con la HCEN (Historia Clínica Electrónica Nacional). Lo que facilitaría el intercambio de información y la interoperabilidad con otros sistemas de salud utilizados.

- **Creación de nuevos plugins**

  - Crear nuevos plugins para otras herramientas de simulación y estimación no consideradas en este proyecto.

- **Mejora de plugins existentes**

  - Mejorar la calidad, robustez y alcance de los plugins desarrollados durante este proyecto (MonolixSuite, Nlmixr2, Mrgsolve).

- **Cargar modelos de nuevos fármacos**

  - Actualmente, solo se han probados distintos modelos de tres fármacos (Ciclosporina, Vancomicina y Tacrolimus). Por lo tanto es de interés probar nuevos fármacos.

- **Realizar mas instancias de validación**

  - Las instancias de validación fueron muy valiosas y aportaron varias ideas de mejoras y nuevas funcionalidades. Además, de ayudar a mejorar las funcionalidades ya desarrolladas. También, puede ser interesante tener instancias con otros actores. Por ejemplo, con doctores de distintas áreas.

Bibliografía

[1] David W Uster, Sophie L Stocker, Jane E Carland, Jonathan Brett, Deborah JE Marriott, Richard O Day, and Sebastian G Wicha. A model averaging/selection approach improves the predictive performance of model-informed precision dosing: vancomycin as a case study. *Clinical Pharmacology & Therapeutics*, 109 (1):175–183, 2021.

[2] Ibarra Manuel. Grupo interdisciplinario para el desarrollo de la dosificación de precisión, 2024. URL `https://www.ei.udelar.edu.uy/grupos-financiados/sigla-acronimo/grupo-interdisciplinario-para-desarrollo-dosificacion-precision`.

[3] Ibarra Manuel y Lorier Marianela y Trocóniz Iñaki. *Pharmacometrics: Population Approach*, pages 946–958. Springer International Publishing, Cham, 2022. ISBN 978-3-030-84860-6. doi: 10.1007/978-3-030-84860-6_172. URL `https://doi.org/10.1007/978-3-030-84860-6_172`.

[4] Ibarra Manuel y Lorier Marianela y Trocóniz Iñaki. Pharmacometrics in precision dosing. *Department of Pharmaceutical Sciences, Faculty of Chemistry, Universidad de la República, Montevideo, Uruguay*, 2021. URL `https://doi.org/10.1007/978-3-030-51519-5_172-1`.

[5] Edison Cid Cárcamo. *Introducción a la farmacocinetica.* 1982.

[6] El viaje de un fármaco en nuestro cuerpo: transporte y fases de actividad, 2024. URL `https://tinyurl.com/4ratsxz9`.

[7] Agustina Drocco y Tomás Facal y Agustín Pirotto. *Plataforma para dosificación de precisión informada por modelos.* PhD thesis, Facultad de Ingeniería Universidad de la República Montevideo, Uruguay, 2022.

[8] Definición de posología, 2024. URL `https://www.cun.es/diccionario-medico/terminos/posologia`.

[9] Definición de posología rae, 2024. URL `https://dle.rae.es/posolog%C3%ADa`.

[10] Milo Gibaldi and Donald Perrier. *Farmacocinética.* Reverté, 2022.

[11] Presentación realizada en el curso de "farmacología básica" dentro de la licenciatura de médico cirujano del Área académica de medicina, 2011. URL `https://uaeh.edu.mx/docencia/P_Presentaciones/icsa/asignatura/SANCHEZ_GUTIERREZ_%20Farmacocinetica.pdf`.

[12] Q.F. Leslie Escobar. Monitorización terapéutica de fármacos y farmacogenética. 07 2024. URL https://tinyurl.com/2n5vuf6c.

[13] Laurence L. Brunton Jhon S. Lazo Keith L.Parker. *Goodman y Gilman: las bases farmacológicas de la terapéutica*. Undécima edición edition. ISBN 0-07-142280-3.

[14] J. F. Borel, Z. L. Kis, and T. Beveridge. *The History of the Discovery and Development of Cyclosporine (Sandimmune®)*. Birkhäuser Boston, Boston, MA, 1995. ISBN 978-1-4615-9846-6. doi: 10.1007/978-1-4615-9846-6_2. URL https://doi.org/10.1007/978-1-4615-9846-6_2.

[15] Tacrolimus, 2024. URL https://medlineplus.gov/spanish/druginfo/meds/a601117-es.html.

[16] Michisane Hashimoto Michihisa Nishiyama Toshio Goto* Masakuni Okuhara Masanobu Kohsaka Hatsuo Aoki Toru Kino, Hiroshi Hatanaka and Hiroshi Imanaka. Fk-506, a novel immunosuppressant isolated from a streptomyces. XL(9): 1249–1255, 1987. doi: https://doi.org/10.7164/antibiotics.40.1249.

[17] Vancomicina, 2024. URL https://medlineplus.gov/spanish/druginfo/meds/a601167-es.html.

[18] Vancomicina, 2015. URL https://www.aeped.es/comite-medicamentos/pediamecum/vancomicina.

[19] Lixsoft, 2024. URL https://lixoft.com/.

[20] Monolix, 2024. URL https://lixoft.com/products/monolix/.

[21] Monolix documentation, 2024. URL https://tinyurl.com/32mj6jbm.

[22] Simulx, 2024. URL https://lixoft.com/products/simulx/.

[23] Lixoftconnectors monolix, 2024. URL https://monolix.lixoft.com/monolix-api/lixoftconnectors_installation/.

[24] Lixoftconnectors simullx, 2024. URL https://simulx.lixoft.com/simulx-api/lixoftconnectors_installation/.

[25] Using regression variables, 2024. URL https://monolix.lixoft.com/demo-projects/regressor/.

[26] Simulate from pkpd qsp models in r, 2024. URL https://mrgsolve.org/.

[27] nlmixr2, 2024. URL https://nlmixr2.org/index.html.

[28] Faqs about front office vs back office, 2024. URL https://sonatafy.com/about-front-office-vs-back-office-2/.

[29] ¿qué es un plugin?, 2024. URL https://www.tooltyp.com/definiciones-de-marketing/que-es-un-plugin/que-es-un-plugin/.

[30] Marta Mariño. ¿qué es un plugin y para qué sirve?, 2023. URL https://www.tooltyp.com/definiciones-de-marketing/que-es-un-plugin/que-es-un-plugin/.

[31] Plugin: Qué es, significado, definición y tipos, 2024. URL https://aulacm.com/que-es/plugin-significado-definicion/.

[32] Qué es una dll, 2024. URL https://learn.microsoft.com/es-es/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library.

[33] ¿qué es un servidor privado virtual (vps)?, 2024. URL https://aws.amazon.com/es/what-is/vps/.

[34] En uruguay ahora todos tenemos una historia clínica digital - k2bhealth, 2024. URL https://k2bhealth.com/blog/post/en-uruguay-ahora-todos-tenemos-una-historia-clinica-digital/.

[35] Historia clínica electrónica nacional | agesic (www.gub.uy), 2024. URL https://www.gub.uy/agencia-gobierno-electronico-sociedad-informacion-conocimiento/node/312.

[36] Sobre el centro - salud.uy - centrodeconocimiento.agesic.gub.uy, 2024. URL https://centrodeconocimiento.agesic.gub.uy/web/salud.uy/#:~:text=El%20programa%20Salud.uy%20es,el%20sector%20de%20la%20salud.

[37] Avances y desafíos de la historia clínica digital en la jornada salud.uy, 2020. URL https://tinyurl.com/4hsuvtpm.

[38] Python, 2024. URL https://www.python.org/about.

[39] Python tutorial, 2024. URL https://www.w3schools.com/python/.

[40] Postgresql, 2024. URL https://www.postgresql.org/about/.

[41] ¿qué es sql (lenguaje de consulta estructurada)?, 2024. URL ttps://aws.amazon.com/es/what-is/sql/.

[42] Introduction to r, 2024. URL https://www.r-project.org/about.html.

[43] React, 2024. URL https://es.react.dev/.

[44] Javascript, 2024. URL https://developer.mozilla.org/es/docs/Web/JavaScript.

[45] Material ui - overview, 2024. URL https://mui.com/material-ui/getting-started.

[46] Material design - overview, 2024. URL https://m3.material.io/.

[47] Supademo, 2024. URL https://supademo.com.

[48] Laura Morvan Felicien Le Louedec, Kyle T Baron. Map-bayesian estimation of pk parameters. Technical report, 2023. URL https://github.com/FelicienLL/mapbayr.

[49] Nonlinear mixed-effects model development and simulation using nlmixr and related r open-source packages, 2024. URL https://doi.org/10.1002/psp4.12445.

[50] nlmixr2 2.1.0+ new estimation methods, 2024. URL https://blog.nlmixr2.org/blog/2024-02-07-new-estimation-methods/.

[51] Bertrand Clarke Tri M. Le. Model averaging is asymptotically better than model selection for prediction. *Journal of Machine Learning Research 23*, 2022.

[52] International perspectives on model-informed precision dosing: From the data to the patients, 2024. URL https://tinyurl.com/yc26bkx6.

[53] Welcome to virtualbox.org!, 2024. URL https://www.virtualbox.org/.

[54] Gitlab, 2024. URL https://about.gitlab.com/.

[55] Instalación de linux en windows con wsl, 2024. URL https://learn.microsoft.com/es-es/windows/wsl/install.

[56] Run linux gui apps on the windows subsystem for linux, 2024. URL https://learn.microsoft.com/en-us/windows/wsl/tutorials/gui-apps.

[57] Code editing. redefined, 2024. URL https://code.visualstudio.com/.

[58] Lucidchart, 2024. URL https://www.lucidchart.com/.

[59] Onedrive, 2024. URL https://www.microsoft.com/es-es/microsoft-365/onedrive/online-cloud-storage.

[60] ¿qué es bash script? características y principales usos (unir.net), 2024. URL https://unirfp.unir.net/revista/ingenieria-y-tecnologia/bash-script/.

[61] uwsgi — documentación de flask (3.0.x) (palletsprojects.com), 2024. URL https://flask.palletsprojects.com/es/main/deploying/uwsgi/.

[62] The uwsgi project, 2024. URL https://uwsgi-docs.readthedocs.io/en/latest/.

[63] ¿qué es nginx y cómo funciona? nginx explicado para principiantes (kinsta.com), 2024. URL https://kinsta.com/es/base-de-conocimiento/que-es-nginx/.

[64] Openbsd manual page server, 2024. URL https://man.openbsd.org/OpenBSD-current/man1/tmux.1.

[65] Tilix, 2024. URL https://github.com/gnunn1/tilix.

[66] Npm, 2024. URL https://docs.npmjs.com/about-npm.

[67] How to use django with apache and mod wsgi, 2024. URL https://docs.djangoproject.com/en/5.0/howto/deployment/wsgi/modwsgi/.

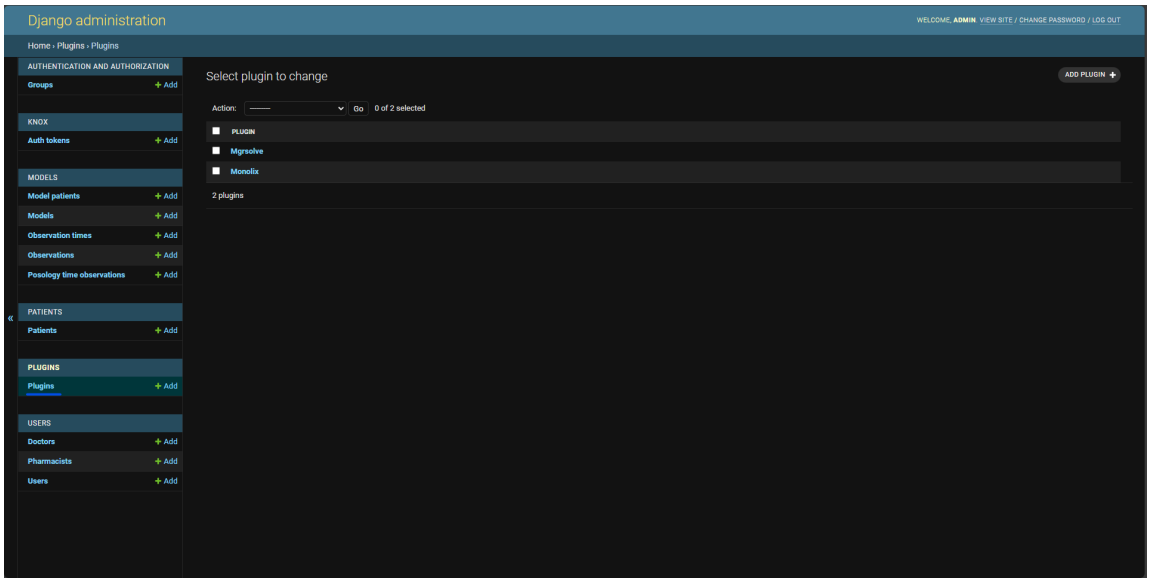
[68] Apache virtual host documentation, 2024. URL https://httpd.apache.org/docs/2.4/vhosts/.

Apéndice A

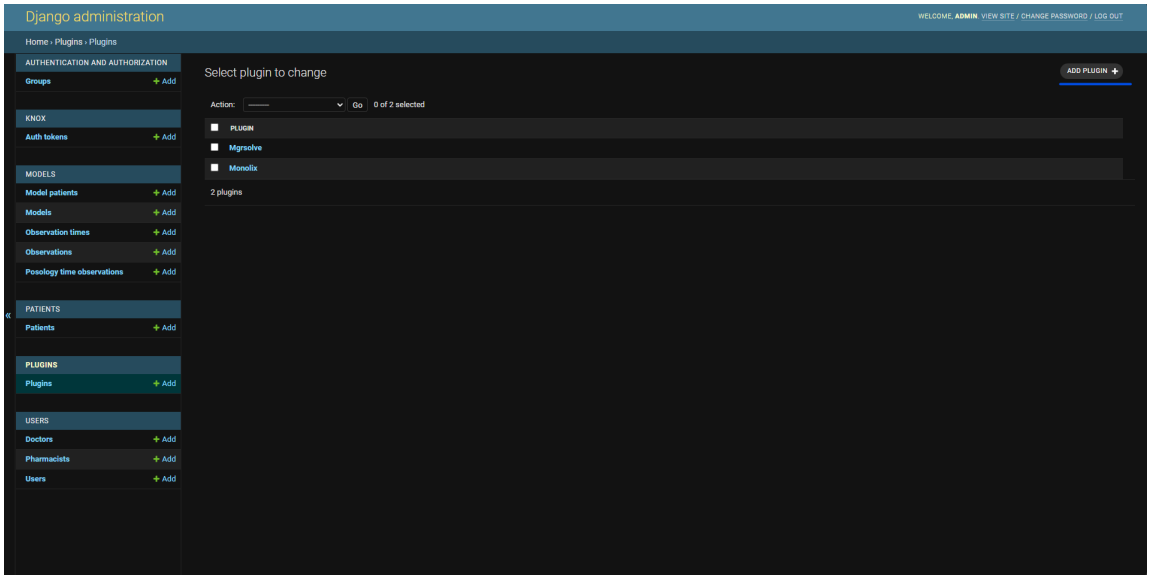
Manual de usuario

A.1. Plugins

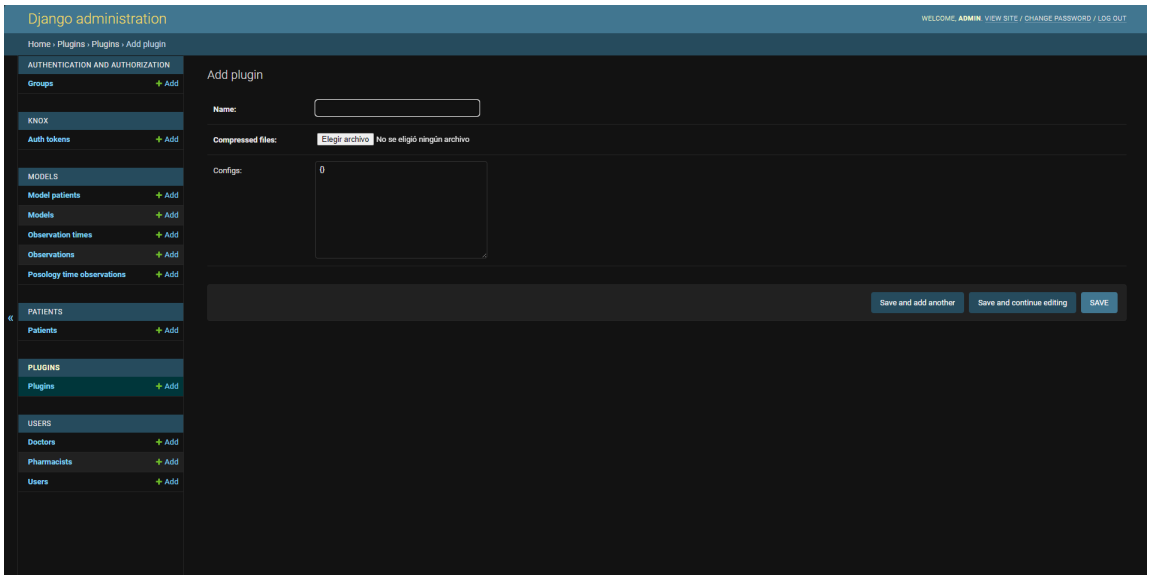
Ingrese a la sección de plugins



Seleccione la opción "+ ADD PLUGIN"

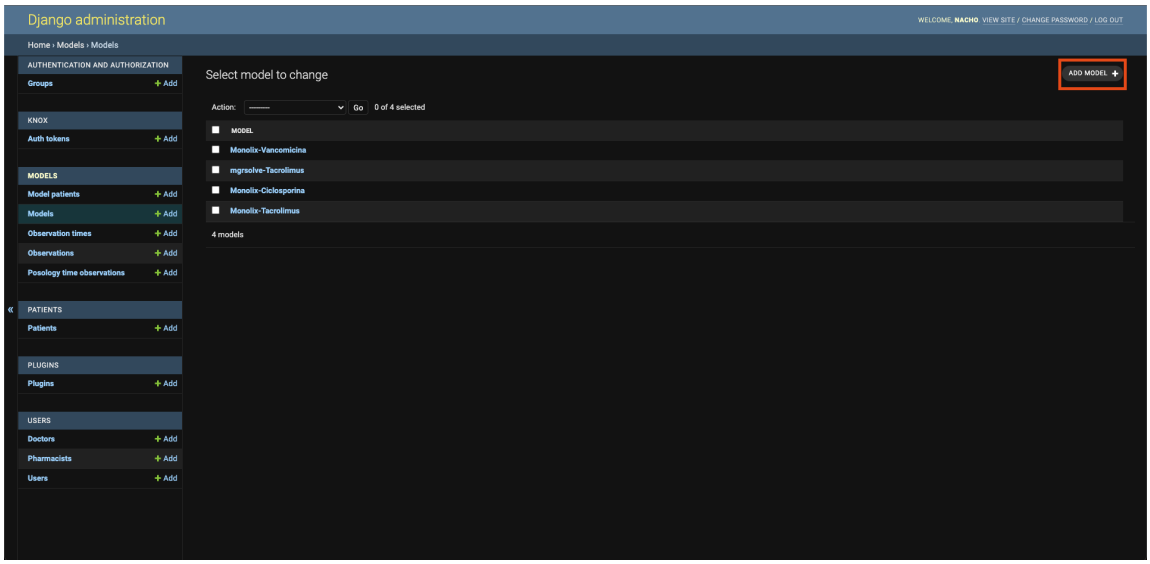


Ingrese el nombre del plugin, un zip con los archivos del plugin y las configuraciones correspondientes del plugin, finalmente presione “SAVE”



A.2. Modelos

Para cargar un modelo en Finglix ir a [FINGLIX_URL]:8000/admin/Models/model/ y hacer clic en “ADD MODEL”.



Aparecerá un formulario para ingresar ciertos datos de ese modelo.

- **Name**: nombre
- **Plugin**: herramienta utilizada para estimar y simular
- **Compartment qty**: cantidad de compartimentos
- **Measurement unit**: unidad de medida

- **Population parameters (Lectura)**: indica los parámetros poblacionales calculados, se calculan la primera vez que se crea el modelo en Finglix, o cuando se guarde el modelo con la opción Calculate population parameters activada
- **II_AUC**: se utiliza para la funcionalidad de simular objetivo, indica sobre que intervalo se mide el AUC. Por ejemplo, el valor doce indica que el valor objetivo que se ingrese corresponderá al AUC en doce horas luego la dosis
- **Config (opcional)**: es un campo que da flexibilidad a las distintas plataformas integradas para proveerles ciertos datos específicos. Consultar en el apartado de cada plataforma qué opciones ofrece.
- **Min dose**: La dosis mínima disponible para suministrar el fármaco, la unidad corresponde a la seleccionada en measurement unit.
- **Dose interval name**: nombre que se utilizó en el modelo para indicar el intervalo entre dosis
- **Stationary state name**: nombre que se utilizó en el modelo para indicar la variable estado estacionario
- **Amount name**: nombre que se utilizó en el modelo para indicar la variable cantidad de dosis
- **Time name**: nombre que se utilizó en el modelo para indicar la variable tiempo
- **Dv name**: nombre que se utilizó en el modelo para indicar la variable distribución
- **Calculate population parameters**: indica si se quiere estimar o reestimar los parámetros poblacionales al guardar

A.2.1. Parámetros

En esta sección podemos cargar todos las variables y output utilizadas en el modelo. No es necesario especificar las ingresadas previamente (tiempo, cantidad de dosis, intervalo, etc).

En un principio aparecerán para cargar hasta tres parámetros, pero presionando el botón **Add another Parameter** se podrán cargar todas los parámetros deseados. Por cada variable deseada se deberán llenar los siguientes campos:

- **Name**: el nombre utilizado en el modelo
- **Type**: el tipo de la variable.

	Descripción	Simulación	Observación
Covariable	Covariables	Si	No
Covariable	Covariables	Si	No
Observation	Covariables	No	Si

Demographic	Variables demograficas	No	Si
Treatment	Tratamiento	No	Si
Regressor	Regresores	Si	Si
Output	Nombre de la salida esperada	Si	Si

- Display name: nombre que se desea que vean los usuarios al ingresar el dato
- Data type: el tipo de dato de esa variable
- Min value (opcional): el valor mínimo que será aceptado por el usuario
- Max value (opcional): el valor máximo que será aceptado por el usuario
- Choices field (opcional): en el caso de que se quieran corresponder ciertos valores a denominaciones más comprensibles por el usuario, también se puede ingresar un diccionario con la llave como el valor que tomará en el modelo y valor el nombre que verá el usuario. Ejemplo: {"1": "Rápido", "2": "Intermedio", "3": "Normal"} IMPORTANTE: tanto las claves como los valores deben estar entre comillas dobles, aunque sean números, estos luego serán interpretados según el tipo de dato seleccionado en el campo Data type.

A.2.2. Files

Aquí se ingresan los archivos necesarios para ejecutar el modelo, a excepción de los archivos de datos (CSV) que serán agregados en la siguiente sección.

Cada herramienta integrada a Finglix tiene su propio estándar sobre que archivos y que nombres deben tener.

A.2.3. File datas

Aquí se ingresan los conjuntos de datos necesarios.

Cada plugin en Finglix tiene su propio estándar sobre que archivos y que nombres deben tener.

Luego si el plugin lo necesita, se deberán cargar las columnas que tenga el archivo de datos. Estos deberán estar separados por "," y sin ningún espacio entre ellos. Ejemplo: DV,ID,II,SS,HCT,LBW,OCC,SEX,EVID,TIME,CYP3A5,AMT_ug_

Finalmente, presionar el botón guardar. No hacer nada hasta que seamos redireccionados al django admin nuevamente.

A.2.4. Lixsoft

Algunas consideraciones para ingresar modelos usando las herramientas de Lixoft son: Antes de ingresar el modelo en Finglix es recomendable ingresarlo a Monolix,

junto con el CSV con los datos poblacionales para corroborar que este puede correrlo y estimar parámetros.

Luego, recomendamos también crear un CSV simulando los datos de una observación de un solo paciente. Esto es para determinar si es necesario cargar datos dummy al CSV del paciente para que este funcione correctamente en Monolix. Si es necesario cargar datos adicionales para que funcione, esto se puede configurar en el campo config, agregando la palabra dummy como key y una lista de filas a cargar (no se debe ingresar las columnas del CSV pero sí se debe respetar el orden y no dejar ninguna columna vacía). A continuación, un ejemplo:

"dummy": ["1,101,1,1,2,2,0,1,1,1,1,1",
"1,101,1,1,2,2,0,1,1,1,1,1",
"1,102,1,1,3,3,1,1,1,1,2,1",
"1,102,1,1,3,3,1,1,1,1,2,1",
"1,103,1,1,4,4,2,1,1,1,3,1",
"1,103,1,1,4,4,2,1,1,1,3,1"]

Ejemplo para modelo del fármaco tacrolimus utilizado por Lixoft:
Name: Tacrolimus
Plugin: Lixoft
Compartment qty: 2
Measurement unit: ug/L
Population parameter: -
Ii auc: 12
Configs: "dummy": [
"1,101,1,1,2,2,0,1,1,1,1,1",
"1,101,1,1,2,2,0,1,1,1,1,1",
"1,102,1,1,3,3,1,1,1,1,2,1",
"1,102,1,1,3,3,1,1,1,1,2,1",
"1,103,1,1,4,4,2,1,1,1,3,1",
"1,103,1,1,4,4,2,1,1,1,3,1"]
Min dose: 500
Dose Interval name: II
Stationary state name: SS
Amount name: AMT_ug_
Time name: TIME
Dv name: DV
Calculate population parameters: True

Parameters

Name	Type	Display name	Data type	Min value	Max value	Choices
HCT	Demographic	Hct	Double			
LBW	Regressor		Double			

CYP3A5	Demographic	Polimorfismo	Integer	1	3	{“1”: “Rápido”, “2”: “Intermedio”, “3”: “Normal”}
HCT	Covariable	Hct	Double			
CYP3A5	Covariable	Polimorfismo	Integer	1	3	{“1”: “Rápido”, “2”: “Intermedio”, “3”: “Normal”}
Cc	Output	Contentración	Double			

Files

Name	File
file_model	Ingresar archivo .mlx-tran
model_aux	Ingresar archivo .txt

File datas

Name	Clumns name	File
data	DV, ID, II, SS, HCT, LBW, OCC, SEX, EVID, TIME, CYP3A5, AMT_ug_	Ingresar archivo .csv

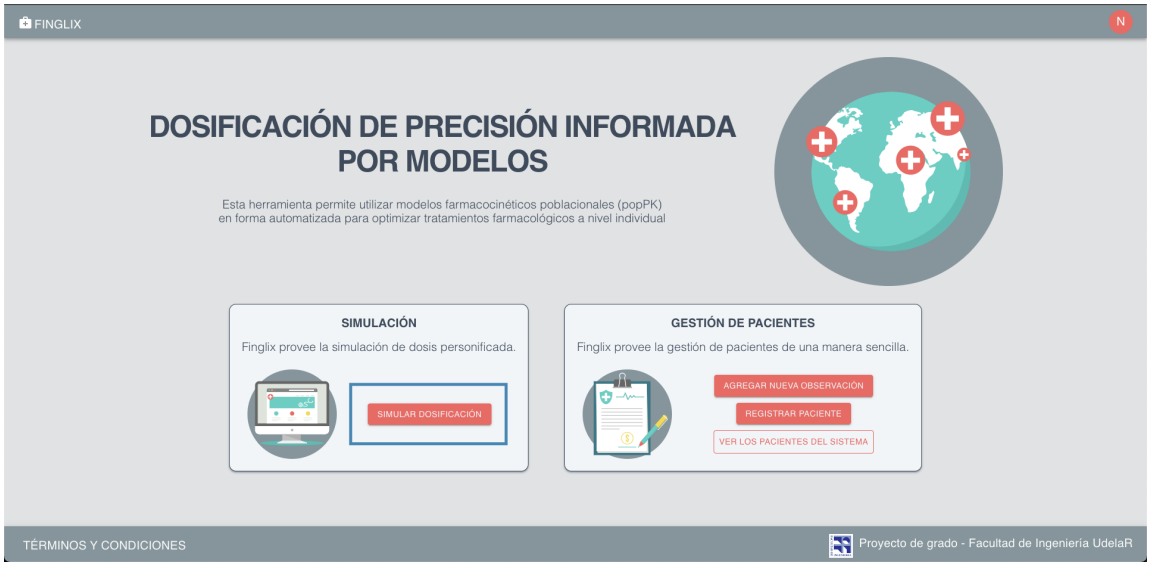
A.3. Simulación

Una simulación es una estimación de cómo se espera que se comporte una variable en un modelo a partir de una serie de parámetros y una serie de valores provistos por el usuario. En el contexto de Finglix a partir de los modelos cargados en el sistema, es posible ejecutar simulaciones tales que a partir de parámetros de ajuste sean poblacionales o individuales (parámetros calculados internamente en el sistema no es responsabilidad del usuario cargarlos), covariables, regresores y un posible output. Posteriormente para cada tratamiento que consiste en una dosis y un intervalo entre dosis definido, se ejecuta la simulación generando como resultado la estimación la variación de la salida seleccionada en función del tiempo y una serie de métricas como son el AUC, el AUC (X) para un intervalo definido en el modelo, máxima y mínima concentración entre otros.

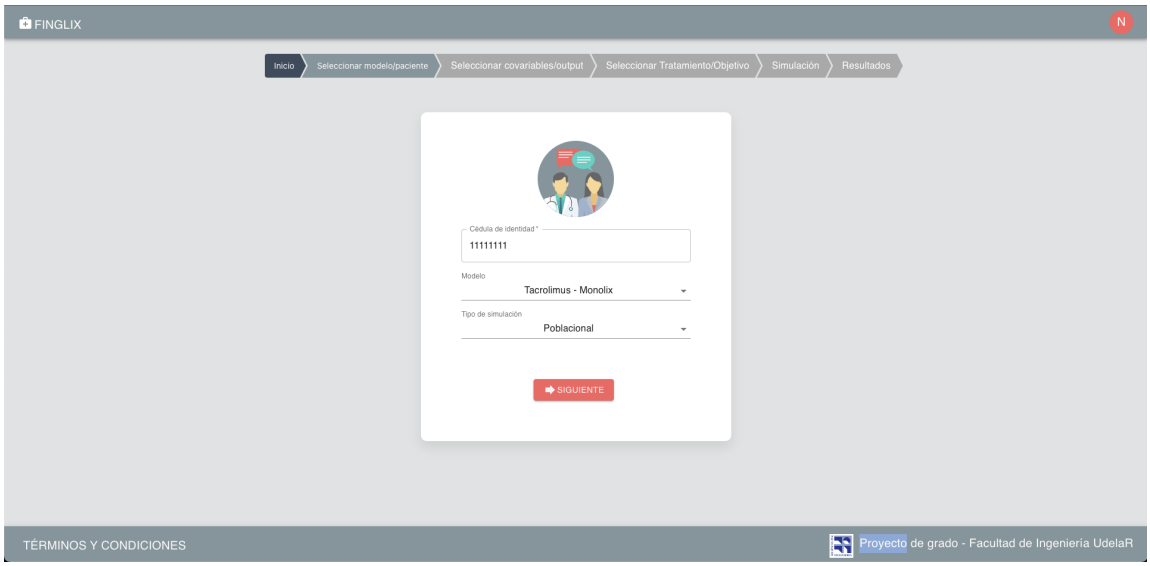
A.3.1. Poblacional

Para realizar una simulación poblacional como requisito debe de estar registrado en el sistema como médico o farmacéutico.

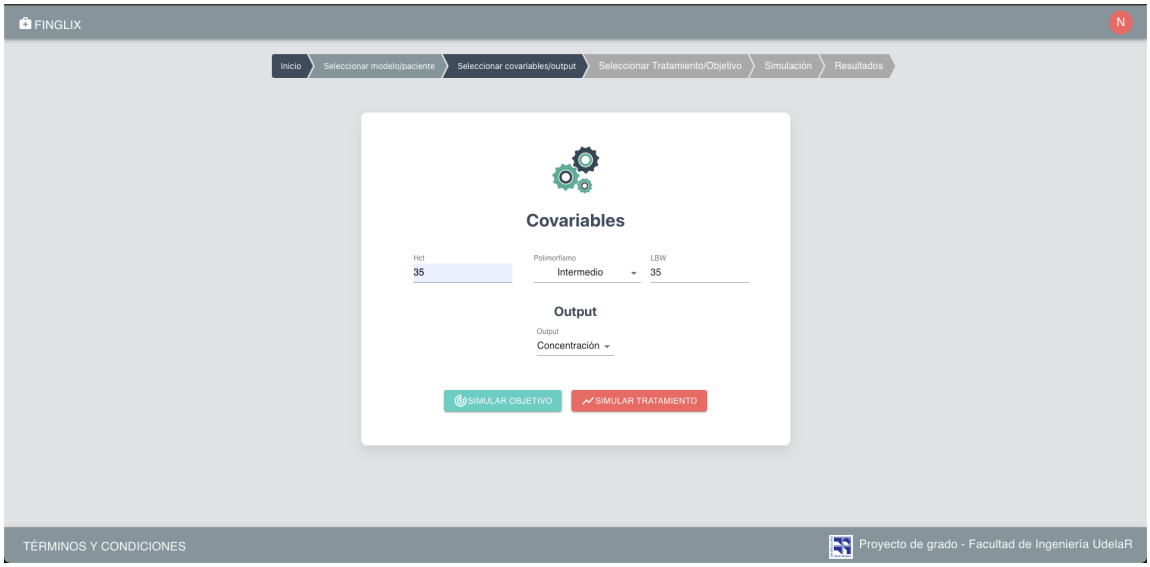
Situado en la pantalla de inicio de la página principal del sistema seleccione “SIMULAR DOSIFICACIÓN”.



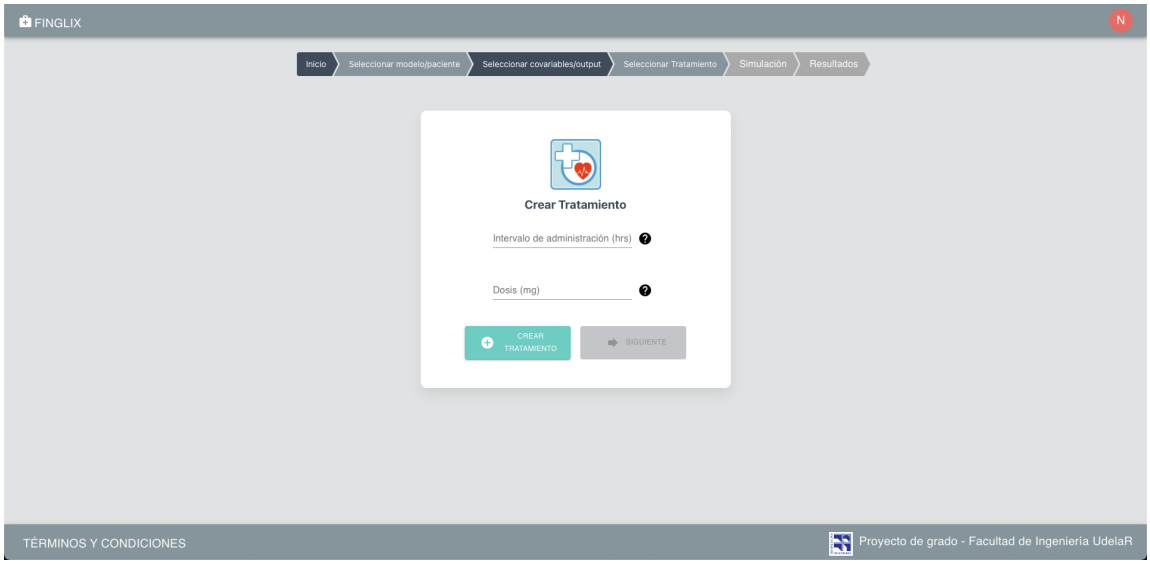
Ingrese la cédula del paciente para el cual desea simular, seleccione un modelo, cada modelo indica su nombre y el nombre del plugin con el que se va a ejecutar la simulación, seleccioné la opción "Poblacional" y finalmente seleccioné la opción "SIGUIENTE".



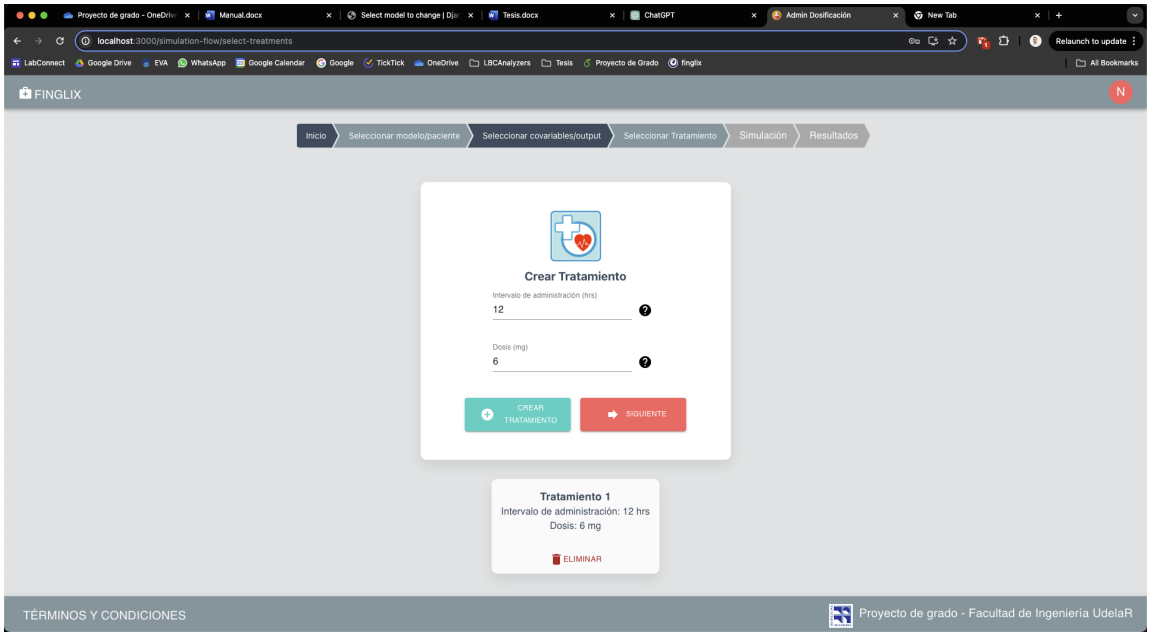
Se desplegará un formulario en el cual se solicitan las covariables y valores para regresores que serán utilizados en la simulación, ingrese los valores con los que desee realizar la simulación, luego seleccione un output, este valor será el que se graficará en la simulación y finalmente seleccione la opción "SIMULAR TRATAMIENTO".



Ingrese el intervalo y dosis que desea simular y luego presione en “CREAR TRATAMIENTO”, puede repetir este proceso tantas veces como desee para realizar una simulación por cada tratamiento.



Presione el botón “SIGUIENTE”.



Se desplegará un resumen con los datos ingresados, finalmente presione "SIMULAR" para realizar la simulación.

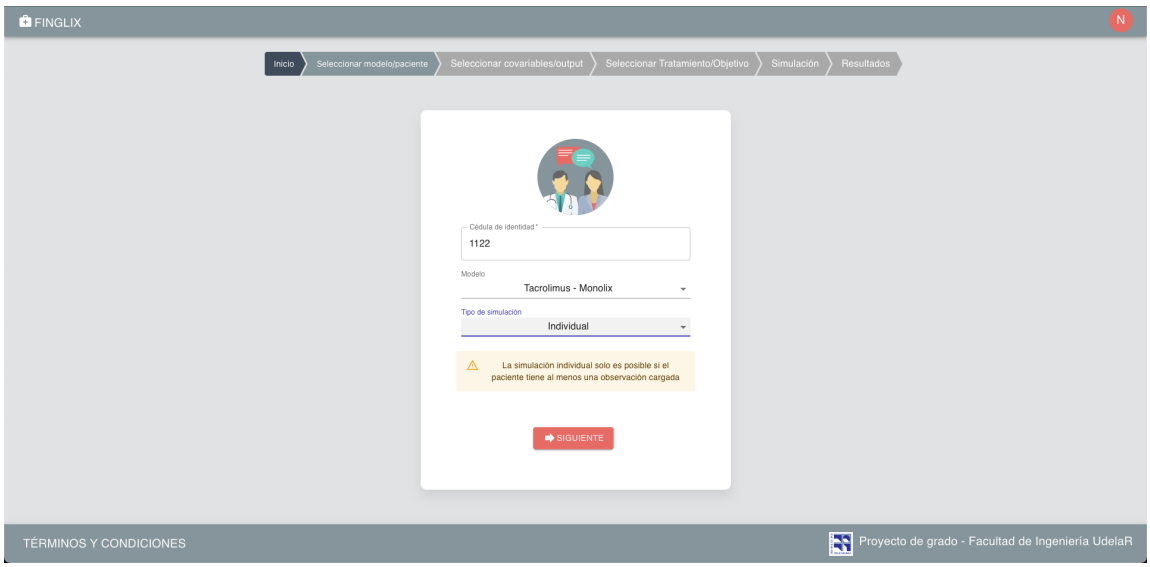
Se mostrará un gráfico del output seleccionado en función del tiempo en la sección izquierda y en la sección derecha se encontrará con las métricas correspondientes a la simulación.

A.3.2. Individual

Para realizar una simulación individual debe de estar registrado en el sistema e ingresar la cédula de identidad de un usuario que previamente se le haya ingresado observaciones para el par modelo plugin que utilizará para realizar la simulación.

Situado en la pantalla de inicio de la página principal del sistema seleccione “SIMULAR DOSIFICACIÓN”.

Ingrese la cédula de identidad del paciente, selección el par modelo plugin, seleccione el tipo de simulación individual y finalmente presiona el botón “SIGUIENTE”



En el siguiente paso a diferencia de la simulación poblacional no se solicitan las covariables del paciente ya que fueron registradas cuando se cargo las observaciones, seleccione una salida y presione “SIMULAR TRATAMIENTO”.

Ingrese un tratamiento definiendo intervalo de dosis y dosis, finalmente presione "CREAR TRATAMIENTO", puede repetir el proceso para cargar todos los tratamientos que desee simular para el paciente.

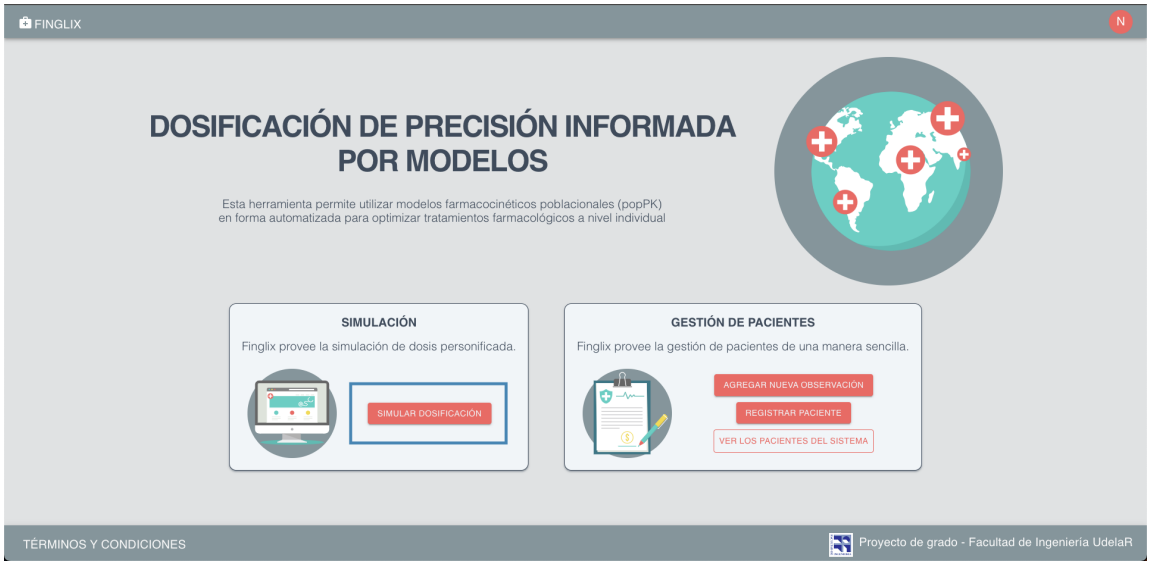
Presione el botón "SIGUIENTE", se le mostrara un resumen de los datos que se utilizaran para la simulación, luego presione "SIMULAR".

Se mostrará un gráfico del output seleccionado en función del tiempo en la sección izquierda y en la sección derecha se encontrará con las métricas correspondientes a la simulación.

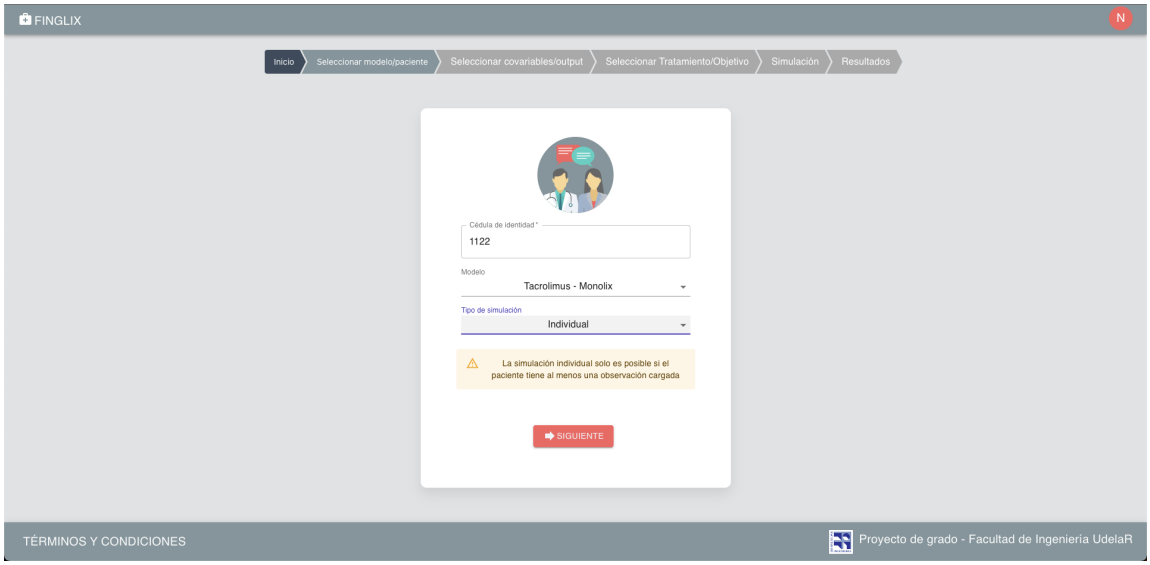
A.3.3. Optimización de dosis

Al utilizar la optimización de dosis en vez de definir un tratamiento y estimar el comportamiento esperado de la salida, se define un objetivo y varios rangos para los tratamientos, y el sistema buscará el tratamiento que más se ajuste al objetivo definido.

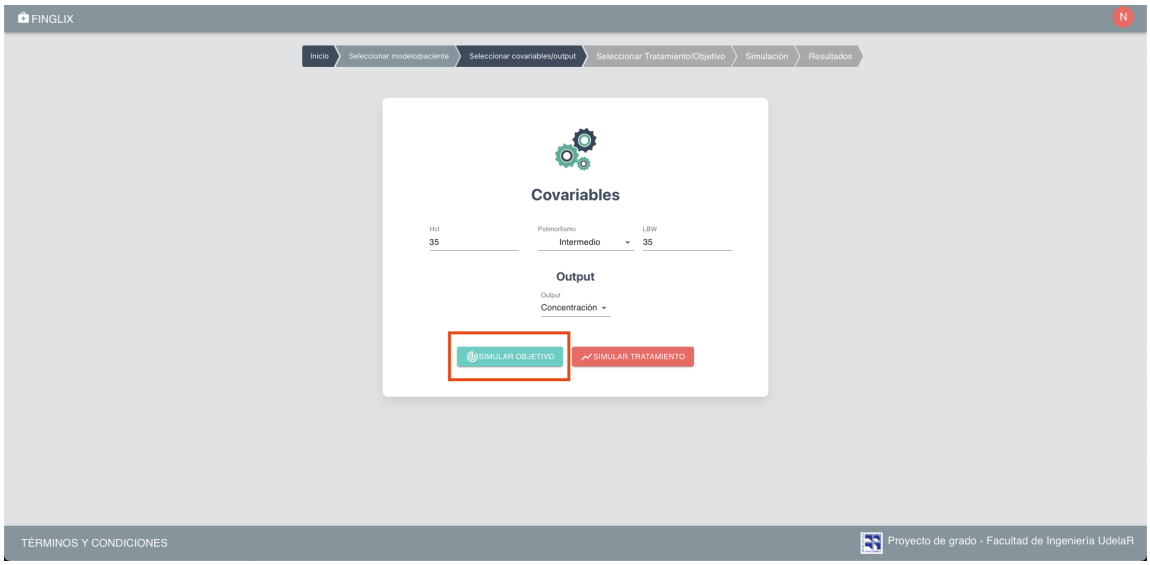
Situado en la pantalla de inicio de la página principal del sistema seleccione "SIMULAR DOSIFICACIÓN".



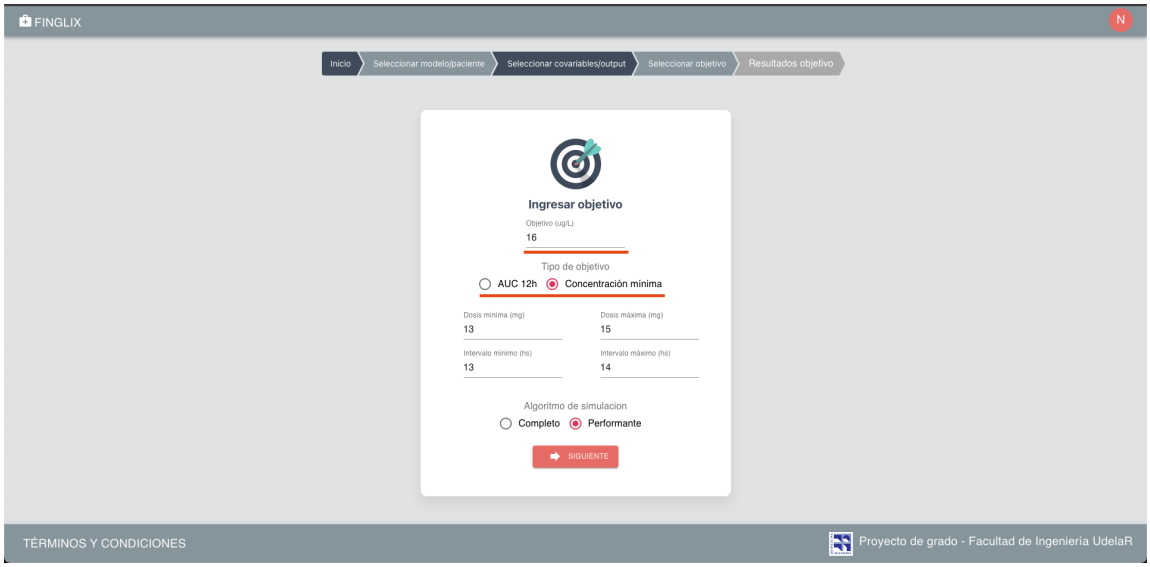
Ingrese la cédula de identidad del paciente, seleccione el par modelo plugin, seleccione el tipo de simulación individual o poblacional y finalmente presiona el botón "SIGUIENTE"



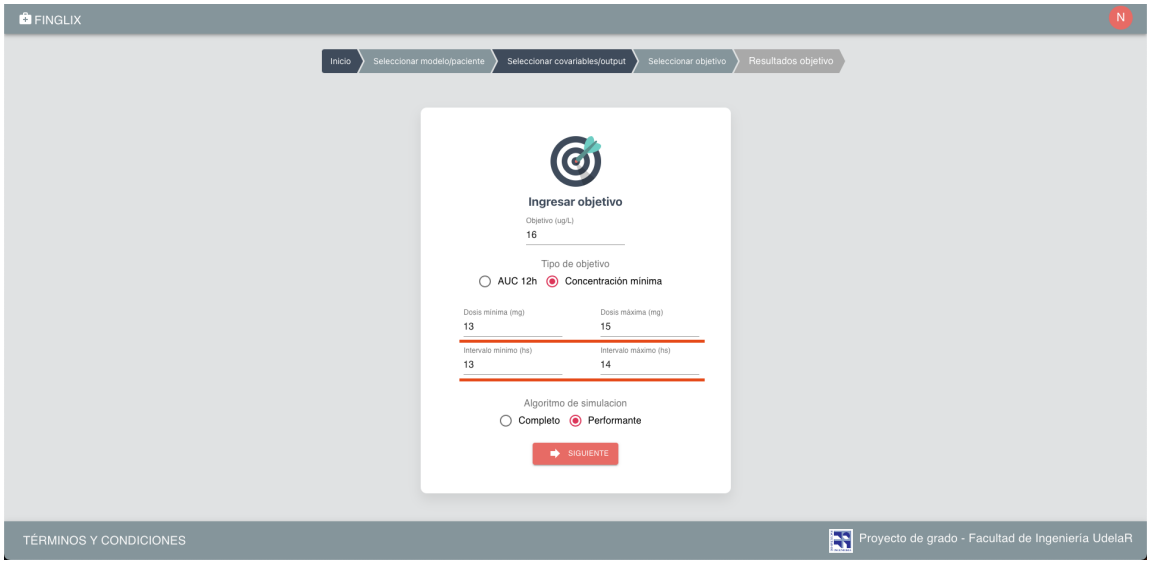
En caso de ser simulación poblacional ingrese las covariables correspondientes, seleccione una salida y finalmente presione el botón “SIMULAR OBJETIVO”.



Ingrese el objetivo deseado, existen dos tipos de objetivos a optimizar, el primero es AUC (X) donde X es el valor definido para el modelo utilizado o el segundo tipo de objetivo es por concentración mínima.



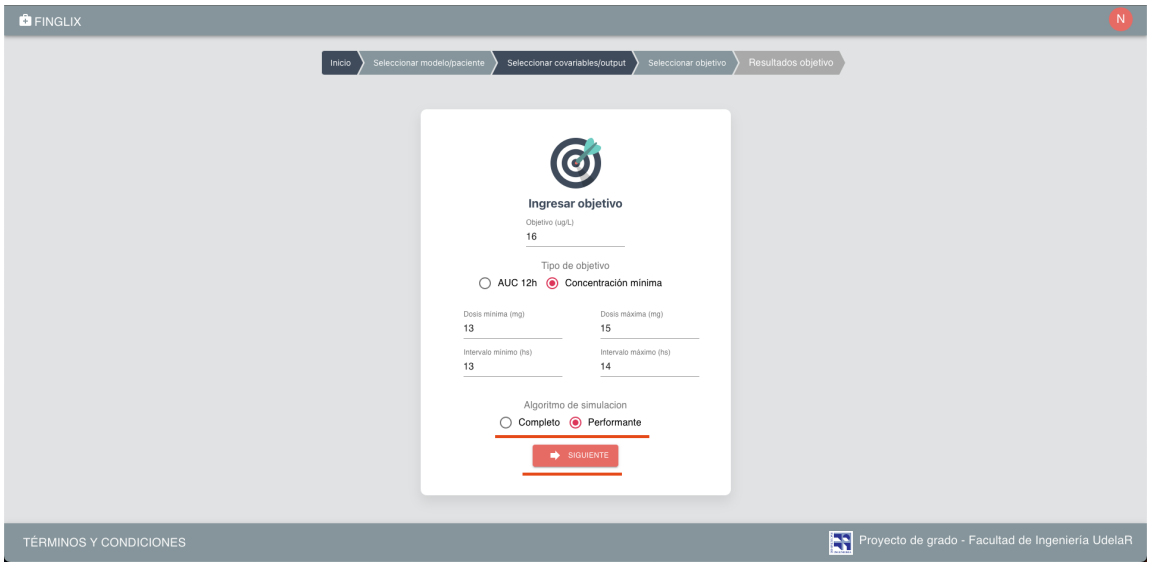
Ingrese los rangos definidos para los tratamientos, es importante tomar en cuenta que en el caso de las dosis los saltos son discretos, donde los tratamientos toman múltiples de la mínima dosis definida en el modelo, por ejemplo, si la dosis mínima es cien y se coloca el intervalo cincuenta a 250 el rango de dosis validas será [100,200], en el caso del rango de horas los saltos son de a una hora, por ejemplo, para seis entre ocho se tomará [6,7,8].



Finalmente debe de elegir el algoritmo a utilizar, existen dos posibles algoritmos, el algoritmo completo probara todas las combinaciones de dosis e intervalos entre dosis, por tanto si existen d dosis posibles, i intervalos posibles y t como tiempo por simulación de tratamiento, aproximadamente la simulación inversa tardara $d \cdot i \cdot t$, en el caso del algoritmo performante el tiempo de ejecución es aproximadamente $d \cdot L(i) \cdot t$, este algoritmo es más rápido pero se asume que para una dosis fija el output es decreciente en función del intervalo entre dosis, por tanto si no se cumple esta condición no esta garantizado que se obtenga el tratamiento mas adecuado, por otra parte el algoritmo completo es más lento pero siempre está garantizado que se

obtiene el mejor tratamiento.

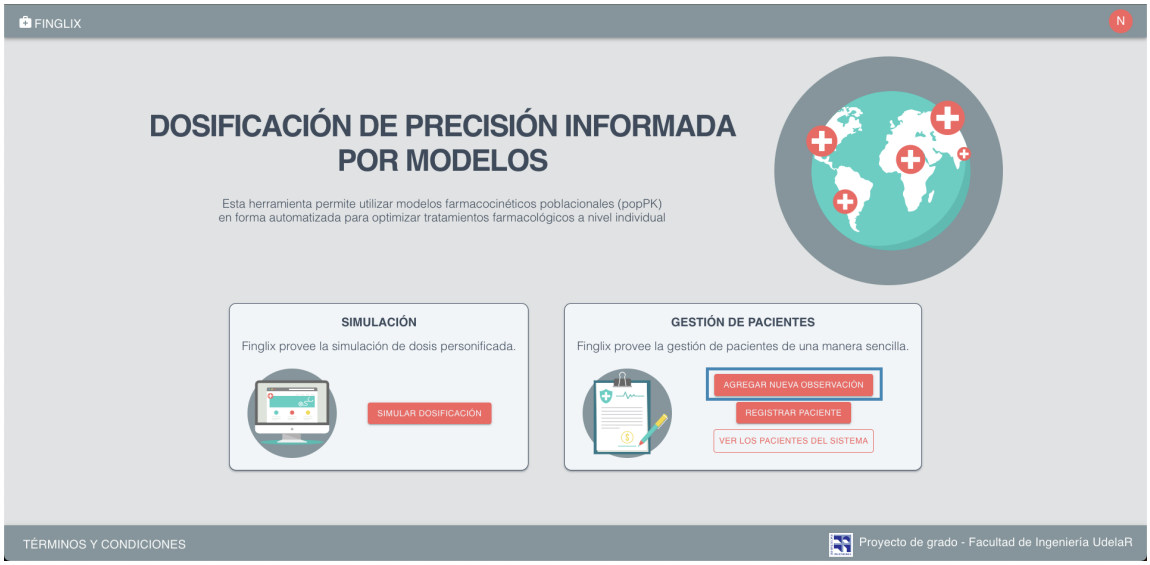
Finalmente presione el botón “SIMULAR”.



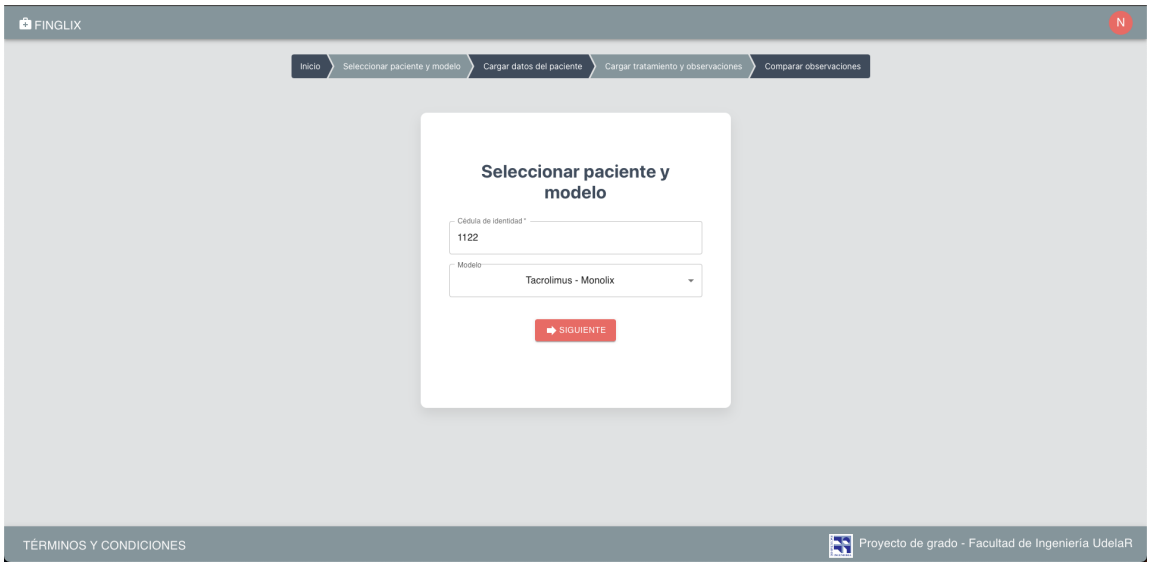
Finalmente, en la sección inferior izquierda se muestra el tratamiento recomendado.

A.4. Observaciones

Seleccione la opción “AGREGAR NUEVA OBSERVACIÓN”



Ingrese la cédula del paciente que debe de estar previamente registrado en el sistema, seleccione el par plugin modelo y presione “SIGUIENTE”



Ingrese los datos del paciente y presione “SIGUIENTE” Ingrese los datos del tratamiento

Cargue los datos de la observación y presione “AGREGAR”, repita el proceso por cada observación.

Finalmente puede presionar “GUARDAR OBSERVACIÓN” y se ejecutara en segundo plano el proceso de la observación o puede presionar “GUARDAR Y SIMULAR”

En el caso de que se presionó "GUARDAR Y SIMULAR", se realizara una simulación con los parámetros reestimados y con el tratamiento ingresado, además se graficarán las observaciones agregadas, permitiendo contrastar como se ajusto el modelo con los nuevos datos ingresados.

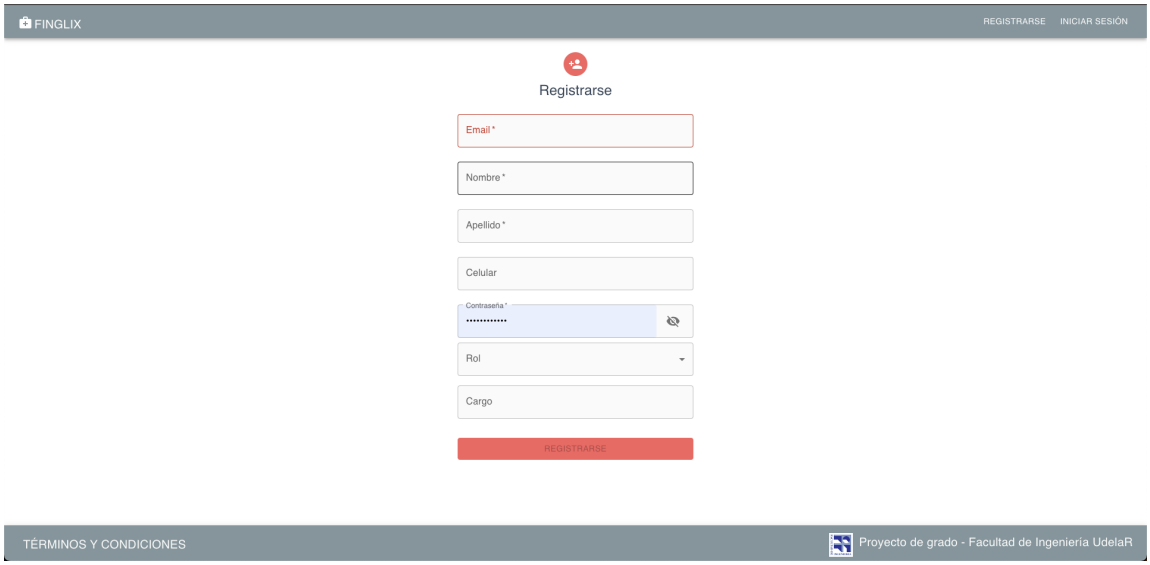
A.5. Usuarios

En el sistema existen tres tipos de usuarios los farmacéuticos, los médicos y el usuario amín, el usuario médico solamente puede loguearse en el frontend y realizar simulaciones, los farmacéuticos pueden realizar todas las acciones sobre el sistema excepto gestionar plugins, por último, el usuario amín tiene acceso a backoffice con acceso total.

A.5.1. Registrar un médico/farmacéutico

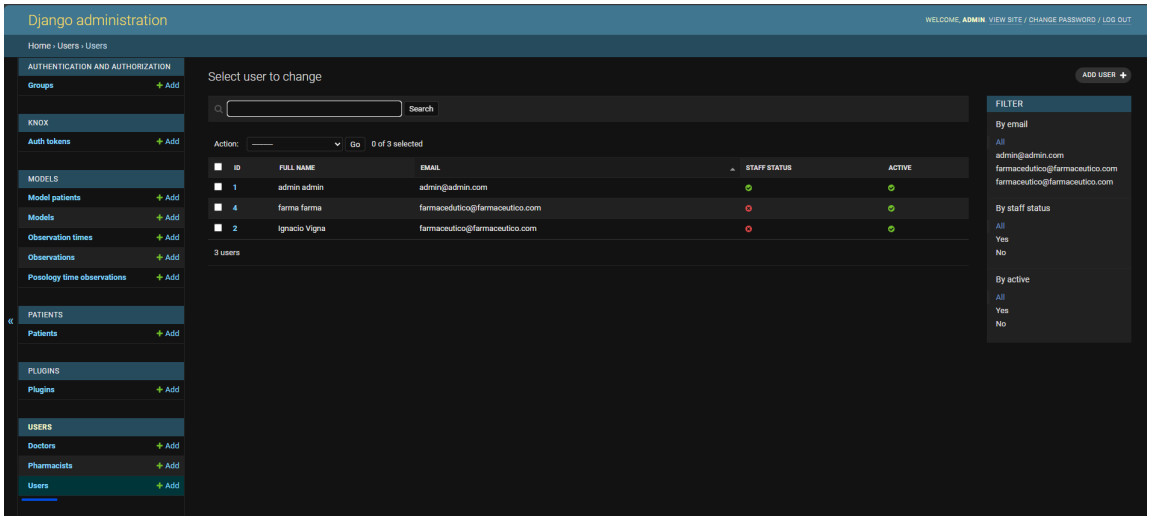
Para registrar un médico/farmacéutico ingrese a la sección "REGISTRARSE"

Ingrese el email y contraseña que serán utilizados para ingresar al sistema, ingrese nombre, apellido, celular, rol, cargo y presione “REGISTRARSE”

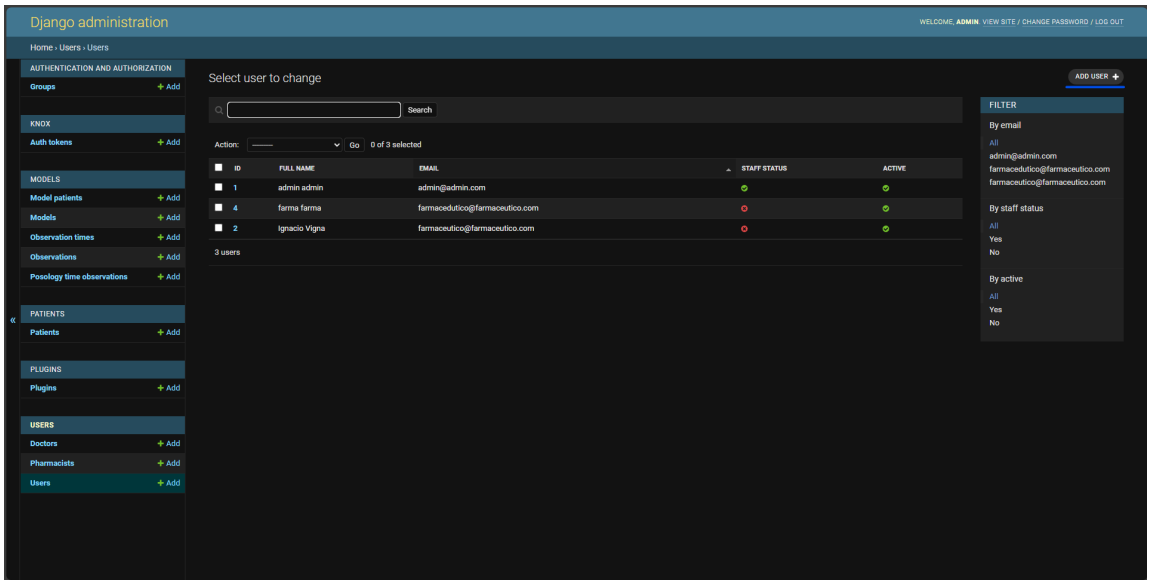


A.5.2. Registrar un admin

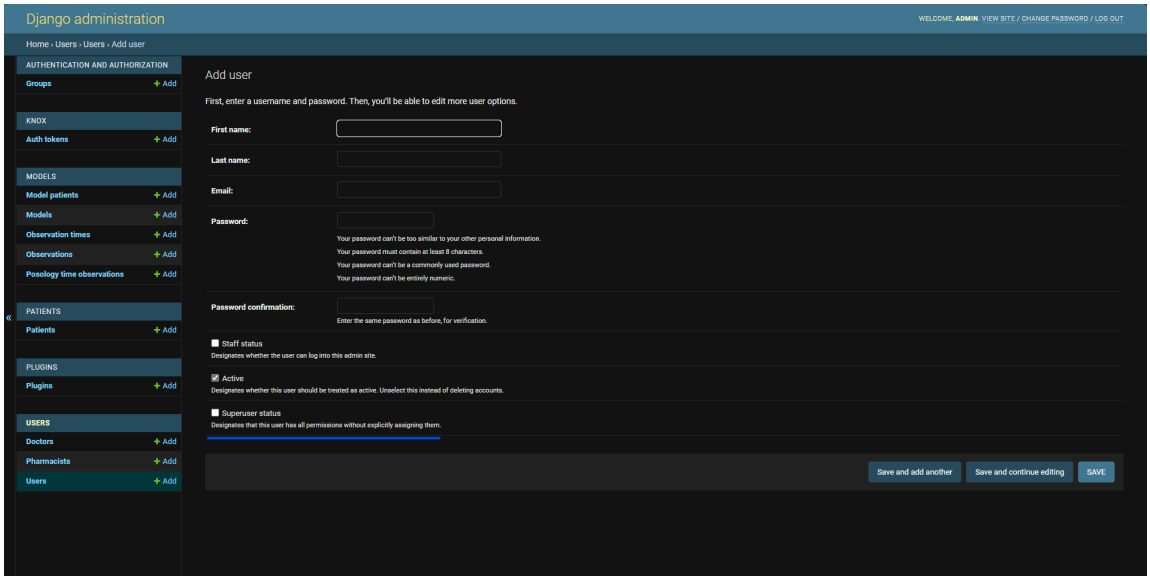
Ingrese a la sección de usuarios



Seleccione la opción "+ ADD USER"



Ingrese los datos correspondientes y marque la opción "Superuser status" y presione "SAVE"


Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Users › Users › Add user
AUTHENTICATION AND AUTHORIZATION
Groups + Add
KNOX
Auth tokens + Add
MODELS
Model patients + Add
Models + Add
Observation times + Add
Observations + Add
Posology time observations + Add
PATIENTS
Patients + Add
PLUGINS
Plugins + Add
USERS
Doctors + Add
Pharmacists + Add
Users + Add
Add user
First, enter a username and password. Then, you'll be able to edit more user options.
First name:
Last name:
Email:
Password:
Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.
Password confirmation:
Enter the same password as before, for verification.
Staff status
Designates whether the user can log into this admin site.
Active
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.
Superuser status
Designates that this user has all permissions without explicitly assigning them.
Save and add another
Save and continue editing
SAVE

Apéndice B

Configuración

B.1. Introducción

A continuación se detalla cómo instalar la aplicación (tanto backoffice como frontoffice) en un entorno para un entorno de desarrollo. Esta instalación se hizo utilizando Ubuntu 22.04.3 LTS. Esta versión también coincide con la que se encuentra en el Virtual Private Server de Antel al momento de la realización del proyecto.

B.2. Lenguaje R

Para el correcto funcionamiento de los plugins de MonolixSuite y de Mrgsolve se debe instalar R. La versión utilizada al momento de realizar el proyecto es la 4.3.2. El requerimiento de instalar este lenguaje de programación es debido a que los plugins anteriormente mencionados hacen uso de él. Para la correcta instalación de R en Ubuntu recomendamos seguir el manual publicado en el sitio oficial: The Comprehensive R Archive Network (unlp.edu.ar)

B.3. PostgreSQL

Además, el backend utiliza la base de datos PostgreSQL, para ello puede utilizarse el comando:

```
sudo apt-get install postgresql postgresql-contrib
```

Luego se deberá iniciar el servicio a través del comando:

```
service postgresql start
```

B.4. RabbitMQ

Se debe instalar RabbitMQ para la ejecución de tareas asíncronas, para ello, puede utilizarse el comando:

```
sudo apt-get install rabbitmq-server
```

Luego se puede levantar el servicio utilizando

```
service rabbitmq-server start
```

B.5. Instalación del frontend

Para poder iniciar el Frontend de la aplicación se debe instalar NVM

```
nvm install 14.15.0
```

Luego se debe instalar NPM:

```
npm install -g npm@7.24.1
```

Luego debemos posicionarlos en la carpeta "front" donde hemos clonado el repositorio y correr el comando

```
npm install
```

Esto instalará las dependencias que requiere el frontend. Finalmente podemos iniciarlo a través del comando

```
npm start
```

B.6. Backend

En la carpeta back/backend se deberá crear (en caso de no existir) un archivo llamado ".env" Cuyo contenido deberá ser el siguiente:

Además, posicionado en la carpeta back, se deberá correr el comando:

```
pip3 install -r requirements.txt
```

esto instalará las dependencias de Python que requiere el backend para poder funcionar correctamente.

B.6.1. Crear usuario admin

Para crear el usuario de administración que permitirá acceder al backoffice, se deberá correr el comando:

```
python3 manage.py createsuperuser
```

B.7. Iniciar backend

Para iniciar el backend, se deberá posicionar en la carpeta back/backend y luego ejecutar el comando:

```
python3 manage.py runserver
```

Para permitir las tareas asíncronas, deberemos ejecutar (posicionados en la misma carpeta mencionada anteriormente) el comando

```
celery -A backend.celery worker -l INFO
```

B.8. Instalación del plugin Monolix

A continuación se describe el proceso de instalación del plugin de Monolix y el software requerido para su funcionamiento.

B.8.1. Instalación de MonolixSuite

Para poder instalar el Plugin de Monolix, previamente se deberá tener instalado MonolixSuite en el sistema. Recomendamos la versión monolixSuite2023R1 que es la versión con la cual fue probado el funcionamiento de los modelos.

Al momento de la instalación, se solicitará el *activation-key*, el cual se obtiene a través de un formulario en su sitio web Downloads - Lixoft

B.8.2. Instalación de la librería de R

Una vez instalado MonolixSuite, se deberá ingresar a la consola del lenguaje R que hemos instalado previamente y ejecutar los siguientes comandos:

```
install.packages("RJSONIO")
install.packages('ggplot2')
install.packages('gridExtra')
```

Y finalmente

```
install.packages(packagePath, repos = NULL, type="source", INSTALL_opts
    ="--no-multiarch")
```

Donde *packagePath* corresponde a la ruta donde se encuentra el archivo *lixoftConnectors.tar.gz* el cual se encuentra en la raíz de la ruta donde se instaló Monolix dentro de la carpeta *connectors*.

B.8.3. Carga del plugin

Una vez realizado los pasos anteriores, estamos en condiciones de cargar el plugin de Monolix al sistema. Para ello, debemos dirigirnos al apartado Plugins en el backoffice, seleccionar el botón "add plugin" y cargar el archivo Monolix.zip El campo configs no debe de modificarse, es decir con "{}", sin modificaciones ya que este plugin no requiere configuraciones extras.

B.9. Instalación del plugin Mrgsolve

Para la instalación de la librería en el lenguaje R se debe correr el comando:

```
install.packages("mrgsolve", dependencies = TRUE)
```

Además, en el caso de utilizar el paquete Mapbayr, para realizar estimaciones de parámetros, ese se puede instalar descargándolo desde CRAN.

B.9.1. Carga del plugin

Una vez realizado los pasos anteriores, estamos en condiciones de cargar el plugin de Mrgsolve al sistema. Para ello, debemos dirigirnos al apartado Plugins en el backoffice, seleccionar el botón "add plugin" y cargar el archivo "Mrgsolve.zip" El campo configs no debe de modificarse, es decir con "{}", sin modificaciones ya que este plugin no requiere configuraciones extras.

B.10. Instalación del plugin Nlmixr2

Para la instalación de la librería en el lenguaje R se debe correr el comando:

```
install.packages("nlmixr2",dependencies = TRUE)
```

Si nos encontramos en una versión de R inferior a la 4.2, si se obtiene error, es necesario ejecutar el comando:

```
remotes::install_version("symengine", version = "0.1.6")
```

Si luego de lo anterior se sigue obteniendo un error, o la instalación se queda en bucle, puede ocurrirque falta alguna dependencia. Es recomendable mirar la consola hasta el momento en que la instalación no entró en bucle ya que ahí está la dependencia faltante.

A continuación, se presenta el listado de dependencias que observando la salida de la consola de la instalación es posible visualizar que era necesario instalar, esto en el servidor de Antel con Ubuntu 16.02, puede tener ligeras variaciones en otras instalaciones de Ubuntu:

```
libxml2-dev, curl, cmake, libfontconfig1-dev, libharfbuzz-dev,
    libfribidi-dev, libcurl4-openssl-dev, libssl-dev, libfreetype6-dev,
    libpng-dev, libtiff5-dev, libjpeg-dev, libcairo2-dev, libmpfr-dev,
    gfortran, libblas-dev, liblapack-dev, libgsl0-dev, m4 \\
```

Las anteriores están disponibles desde apt-get.

Adicionalmente debe instalarse gmp-6.2.1.

Para instalarlo, ejecutar los siguientes comandos:
1- Crear un directorio para descargar gmp4:

```
mkdir /home/USUARIO/gmp4
```

2-Moverse al directorio creado

```
cd /home/USUARIO/gmp4
```

3-Setear la siguiente variable de entorno:

```
PREFIX=/home/USUARIO/gmp4
```

4-Descargar gmp4

```
curl -fLO https://gmplib.org/download/gmp/gmp-6.2.1.tar.xz
```

5-Descomprimir el archivo descargado

```
tar -xf gmp-6.2.1.tar.xz
```

6-Moverse al directorio que se generó mediante la descompresión del archivo anterior

```
cd gmp-6.2.1
```

7-Ejecutar la instalación a través de los comandos:

```
./configure --prefix=\$PREFIX
make all
make install
```

8-Copiar el archivo gmpxx.h a la carpeta include donde se realizó la instalación

```
cp gmpxx.h \$PREFIX/include
```

9-Definir las siguientes variables de entorno

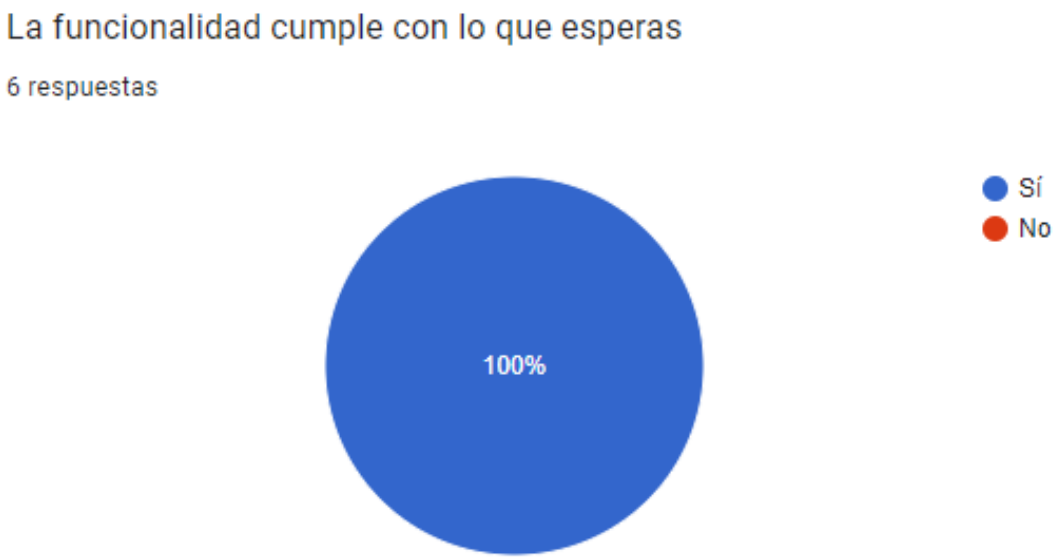
```
export GMP\_INC=\$PREFIX/include
export GMP\_LIB=\$PREFIX/lib
```

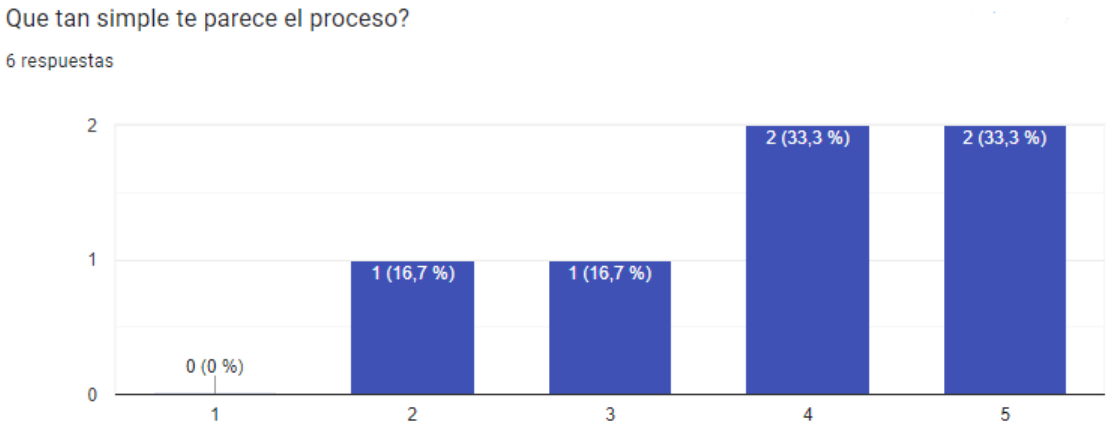
Apéndice C

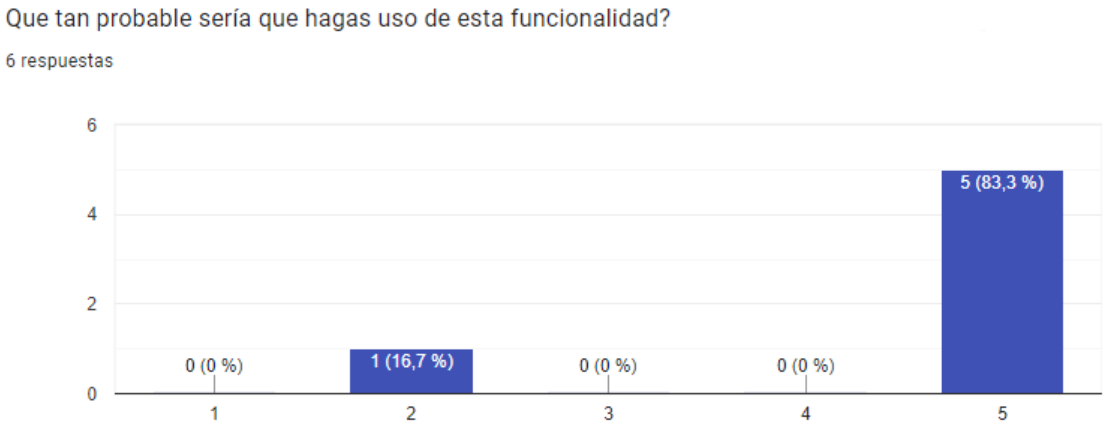
Pruebas

A continuación se adjuntan los resultados de la encuesta de Google que fue realizada en la reunión de validación con usuarios finales: médicos, farmacéuticos y administradores del sistema.

C.1. Simulación







C.1.1. Cambiarías algo de la funcionalidad

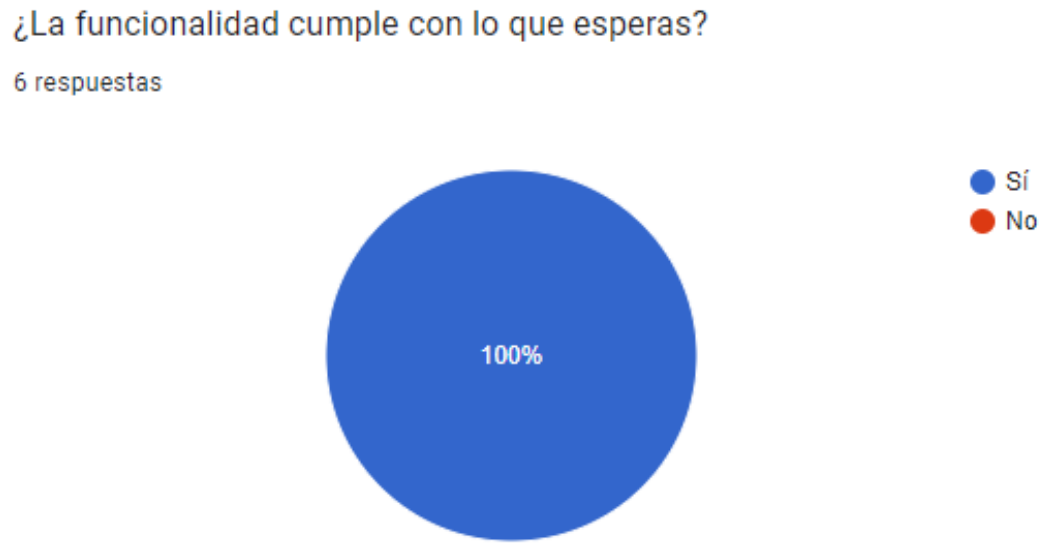
Los encuestados sugieren:

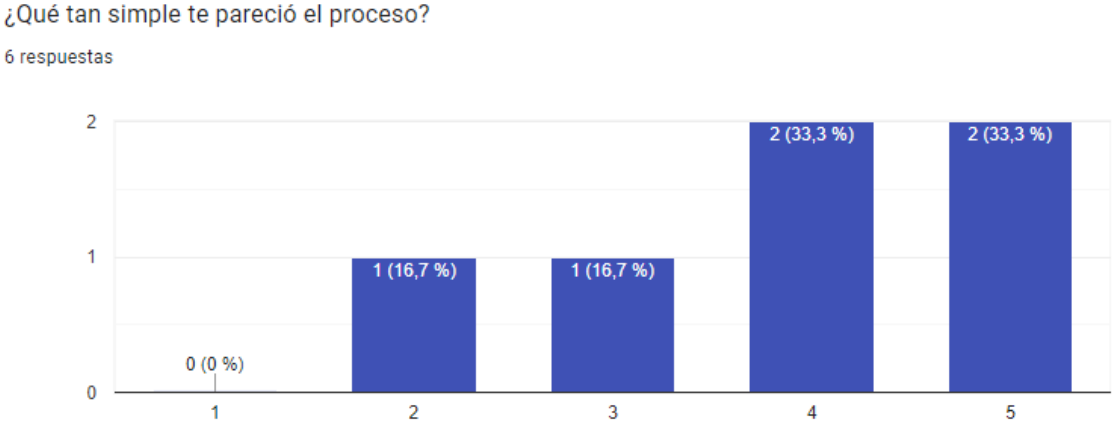
1. Modificar el sistema para que las observaciones se agreguen por fármaco en lugar de por modelo, permitiendo que se cree se registren los datos acumulativa por fármaco. Esto facilitaría probar diferentes modelos sobre los mismos datos.

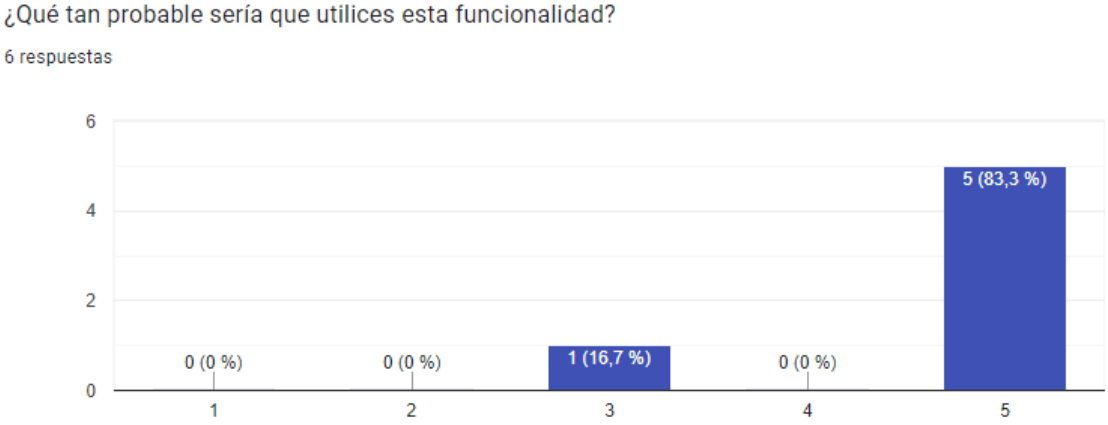
2. Incluir unidades en las covariables del modelo.

3. Mejorar el reporte unificando las unidades (por ejemplo, "h.$^{\text{en}}$ lugar de "hs") y agregando una función que ordene los tratamientos según la dosis diaria total.

C.2. Simulación objetivo





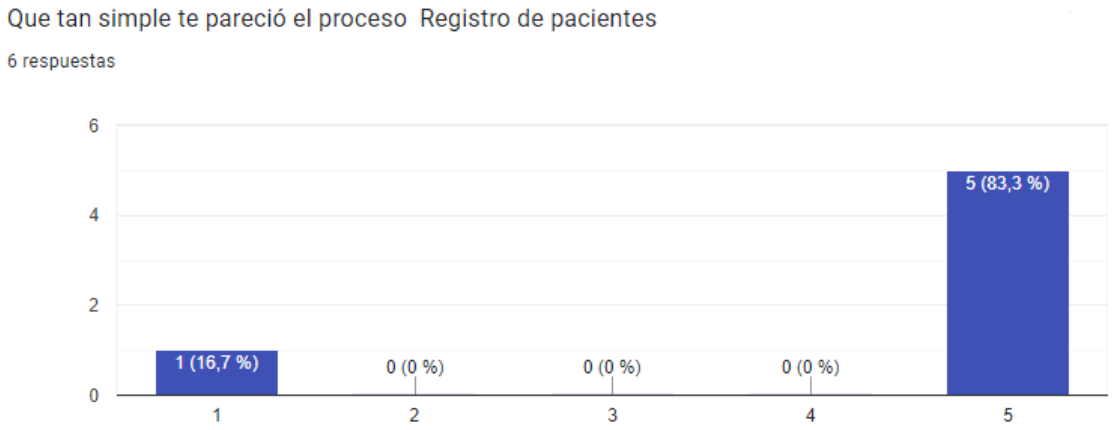


C.2.1. ¿Cambiarías algo en la funcionalidad?

Los encuestados sugieren:

1. Añadir una descripción de cuándo usar cada algoritmo, o una recomendación para facilitar la decisión médica.
2. Dosis variables en el intervalo de administración
3. Restringir la búsqueda de tratamientos a intervalos de 48, 24, 12, 8 y 6 horas de administración.
4. Asegurar que las unidades se actualicen según el objetivo (AUC o concentración máxima).

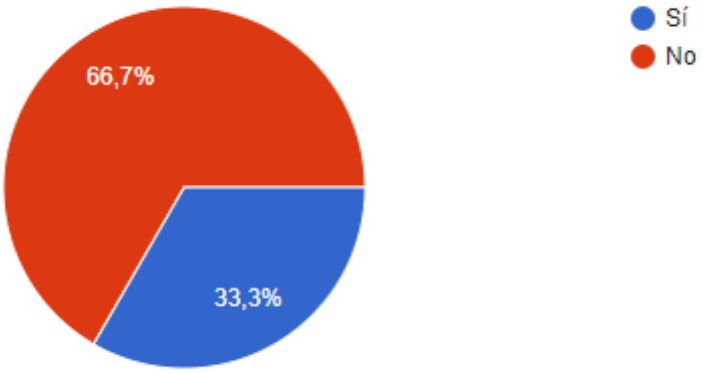
C.3. Registro de pacientes



C.4. Observaciones

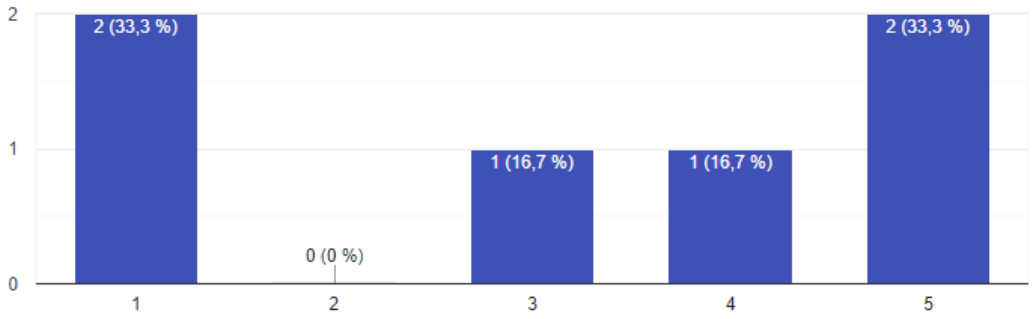
¿La funcionalidad cumple con lo que esperas?

6 respuestas



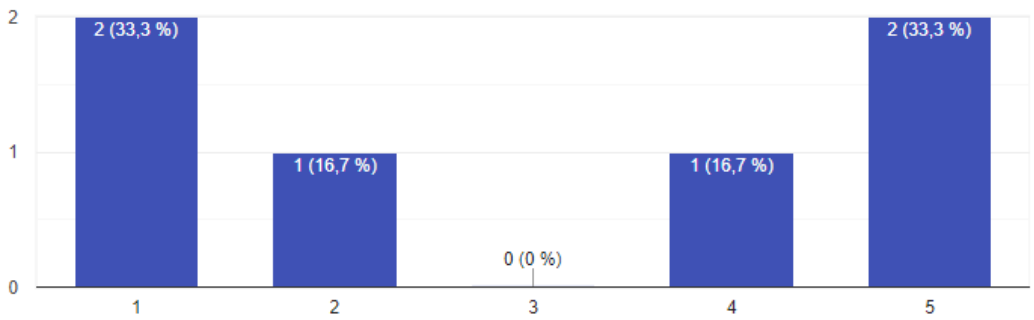
¿Qué tan simple te pareció el proceso?

6 respuestas



¿Qué tan claro te pareció la forma de mostrar los datos generados a partir de la carga de observaciones?

6 respuestas

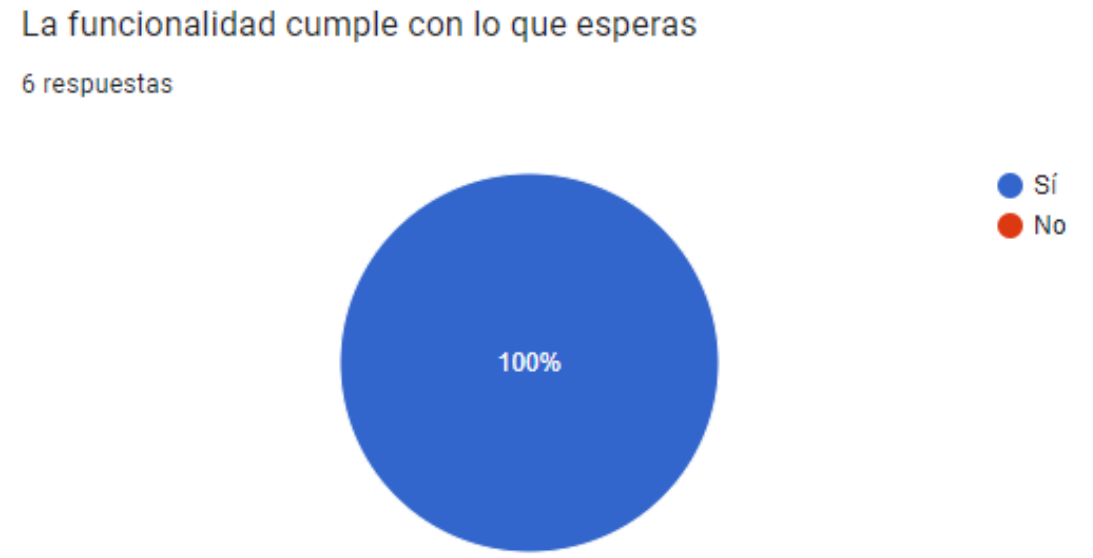


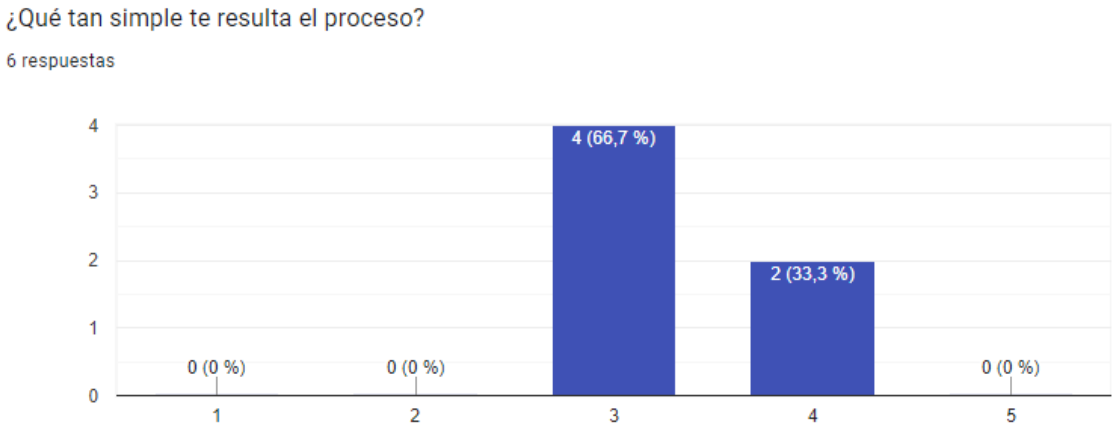
C.4.1. ¿Cambiarías algo en la funcionalidad?

Los encuestados sugieren:

1. Incluir una descripción que indique qué datos ingresar. Si en el futuro se cambia el sistema a observaciones por fármaco (y no por modelo), solicitar todas las covariables y regresores utilizados en los modelos para ese fármaco.
2. Permitir seleccionar la unidad de dosis (µg, mg, g).
3. Permitir ver todas las observaciones para un sujeto en un período de tiempo seleccionable y expresar la dosis en mg, la unidad más comúnmente utilizada.

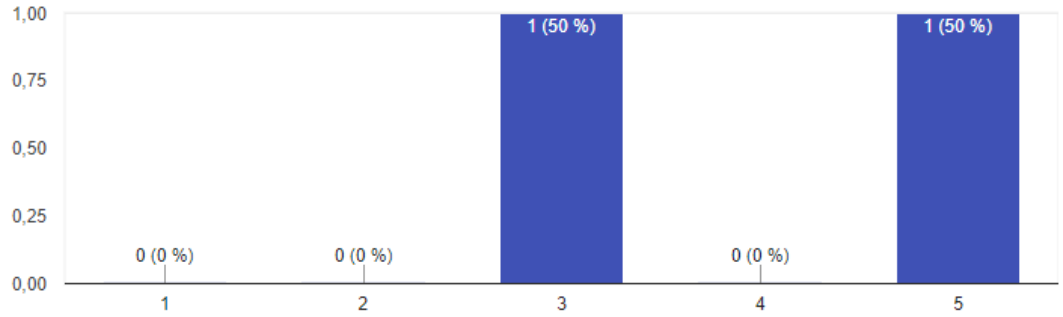
C.5. Carga de modelos





Respecto al proceso anterior de carga de modelos, te pareció más simple? (en caso de no conocer el proceso anterior no es necesario contestar)

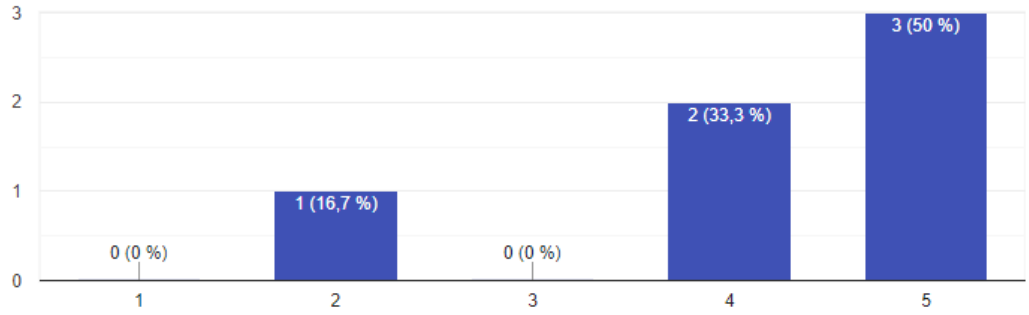
2 respuestas



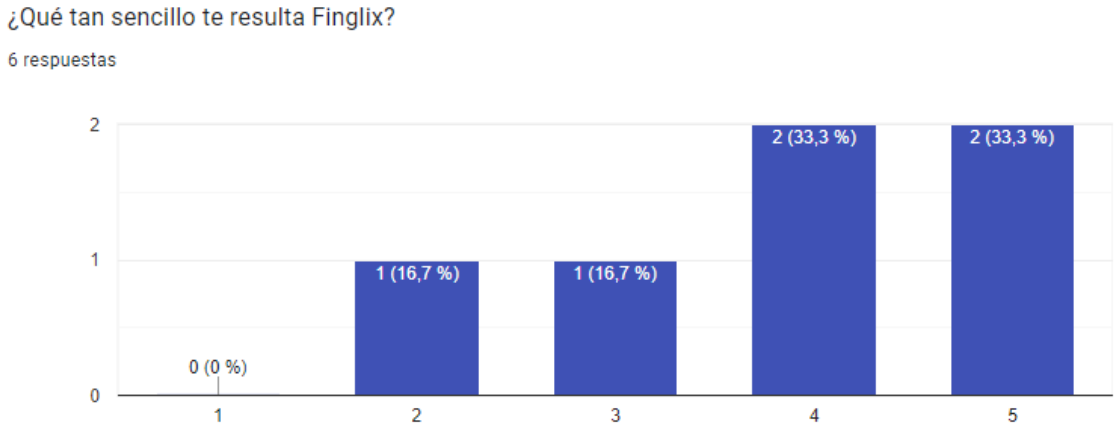
C.6. Plugins

Qué nivel de utilidad le ves a futuro a esta funcionalidad?

6 respuestas



C.7. Generales



C.7.1. ¿Qué cambiarías de Finglix?

Los encuestados sugieren:

1. Modificar las descripciones técnicas para que sean más accesibles, por ejemplo, cambiar .ªgregue las covariables"por ïngrese las características del paciente". También, indicar claramente las unidades requeridas para cada dato y uniformar las unidades (como usar mL/min para el clearance de creatinina y kg para el peso).

2. Incluir un tutorial o comentarios explicativos durante el proceso de agregar un modelo, para guiar al usuario sobre qué datos se necesitan.

3. Fomentar la interconexión entre las diferentes funciones de la plataforma, permitiendo modificar características del paciente, modelo o simulación desde un solo lugar sin tener que volver al inicio.

Apéndice D

Despliegue

A continuación se detalla las tecnologías usadas en el despliegue, como se desplegó el proyecto durante el proceso de implementación y el despliegue final.

D.1. Detalles Técnicos del Despliegue

Al igual que en el proyecto anterior se realizó el despliegue en un *Virtual Private Server* provisto por Antel[1]. Este se ofrece a través de la plataforma denominada Mi Nube[2]. En el marco del proyecto de grado la tutora obtuvo una cuenta gratis en principio por un año que luego se extendió a casi dos para abarcar la duración del proyecto de esta plataforma.

Se hizo un análisis del consumo de recursos de la aplicación, observándose que desplegar el *frontend* en modo desarrollo (no despliegue final, en las siguientes secciones se detalla en este concepto) que permita incorporar cambios rápidos al servidor, trayendo los cambios directamente desde Gitlab[3] y además desplegar el servidor backend con Celery consume en torno a los cuatro Gb de memoria RAM. Entre los planes ofrecidos por Antel, se consideró que el que mejor se ajusta a estos requerimientos es el que ofrece ocho Gb de memoria RAM, cuatro núcleos y 150 Gb de espacio en disco.

En las siguientes secciones se introducen las tecnologías utilizadas para el despliegue en la nube ya que posteriormente se hará mención a ellas.

D.1.1. uWSGI

uWSGI es un conjunto de servidores rápidos y compilados con una gran cantidad de configuraciones y capacidades. Implementa varios servidores de aplicaciones para distintos lenguajes de programación y plataformas: WSGI, PSGI, Rack, LUA, WSAPI, CGI, PHP, Go y protocolos. La versatilidad, el rendimiento, el bajo uso de recursos y la fiabilidad son los puntos en los que se basa el proyecto [61] [62].

[1] `https://www.antel.com.uy`
[2] `https://minubeantel.uy`
[3] `https://gitlab.fing.edu.uy`

D.1.2. Nginx

Nginx es un servidor web de código abierto el cual también es usado como *proxy* inverso, caché de HTTP y balanceador de carga. Fue creado originalmente como una respuesta al problema C10K el cual se refiere al problema de manejar el número de 10.000 conexiones de manera concurrente. Debido a que sus raíces busca la optimización del rendimiento a grandes escalas, Nginx a menudo supera a otros populares servidores web en pruebas de rendimiento [63].

D.1.3. Tmux

Tmux es un multiplexor de terminales que permite alternar entre varios programas en una misma terminal. También permite desacoplarlos para que sigan ejecutándose en segundo plano y volver a acoplarlos en una terminal diferente [64].

D.1.4. Tilix

Tilix es un emulador de terminal que permite organizar múltiples terminales dentro de una misma ventana, ya sea dividiéndolas horizontal o verticalmente. Permite reordenar las terminales arrastrándolas y soltándolas, tanto dentro de la misma ventana como entre diferentes ventanas. También es permite separar una terminal y moverla a una nueva ventana. Entre sus características permite sincronizar la entrada de texto entre varias terminales, de modo que los comandos que se escriben en una se replican en las otras. Además, permite guardar y cargar la disposición de tus terminales desde el disco.

En resumen, Tilix es una herramienta que permite gestionar múltiples terminales de manera eficiente y personalizada, con muchas opciones para adaptar la experiencia a tus necesidades [65].

D.2. Despliegue anterior

Para el despliegue en la plataforma, el equipo anterior utilizó un VPS con Ubuntu 16.04, la última versión del sistema operativo que se ofrecía en los servidores de Antel hasta ese momento.

Esto implicó el uso de MonolixSuite 2020 ya que la versión 2021 no era compatible con dicha versión de Ubuntu.

Para poder instalar MonolixSuite con la licencia provista a estudiantes, se debe contar con interfaz gráfica. Para solucionar esto, el equipo anterior instaló Ubuntu Desktop y VNC[4] para conectarse remotamente a la interfaz gráfica y así poder introducir la clave del producto.

Se utilizó uWSGI para servidor de aplicación del backend programado con Python. También se instaló RabbitMQ ya que es requerido por el backend para la ejecución

[4] `https://www.realvnc.com`

de tareas asíncronas.

Para el acceso al frontend, se instaló NPM que es el administrador de paquetes para Node.js, que facilita la instalación y gestión de bibliotecas y herramientas JavaScript para desarrollar aplicaciones del lado del servidor [66] y se instaló las librerías necesarias.

Además, se menciona que para mantener estos tres servicios operativos (uWSGI, RabbitMQ y Node) se utilizó Tmux. Por lo cual se tuvieron tres terminales ejecutando.

D.3. Despliegue durante el desarrollo

Para el despliegue durante el desarrollo, se tuvo con la misma limitación que el equipo anterior: Antel ofrecía solo hasta la versión de Ubuntu 16.04.

Además la instalación de Ubuntu Desktop rompe las configuración de las interfaces de red del servidor, lo que hacía que quedara inaccesible, además de que tampoco se podía recrear la instancia desde el panel. Esto implicó la creación de múltiples *tickets* de soporte a Antel solicitando la reinstalación del VPS de forma manual.

Además, a esto se sumó que había una inestabilidad en el servicio proporcionado por Antel, lo cual hacía que el VPS funcionara de manera lenta.

Cuando Antel resolvió dicha inestabilidad, se migró a otro VPS, administrado desde otro Panel llamado Virtuozzo Power Panel[5]. Y el equipo de Antel notificó que se soportarían nuevas versiones de sistemas operativos, incluyendo Ubuntu 22.04.4 LTS. Esta nueva versión de Ubuntu es la misma que se utilizó para programar. Esto implicó que ahora era soportada la última versión de MonolixSuite y no existiría esta dificultad que sí tuvo el equipo anterior.

Para el despliegue dado el problema que se tuvo con la instalación de Ubuntu Desktop, fue necesario buscar otra manera de introducir la clave de activación de Monolix.

Se instaló XRDP que proporciona un inicio de sesión gráfico en máquinas remotas utilizando el protocolo RDP (Protocolo de Escritorio Remoto de Microsoft)[6].

Luego se intentó conectarse mediante escritorio remoto para ver si existía alguna interfaz gráfica mínima instalada por defecto en el servidor de Antel. Este fue el caso, ya que al acceder se observa que está instalado Xorg[7] con una terminal Tilix.

De esta manera fue posible instalar MonolixSuite ya que al poseer una interfaz gráfica, permitió que el instalador se inicie y se pueda introducir la clave de activación.

[5] `https://docs.virtuozzo.com/virtuozzo_powerpanel_readme/pp-overview.html`
[6] `https://www.xrdp.org/`
[7] `https://www.x.org/wiki/`

Para el despliegue durante el desarrollo no se utilizó Tmux, a diferencia del equipo anterior.

En su lugar, se configuraron tres *cron job* con la etiqueta *"reboot"*, es decir, que ejecutan al iniciar el servidor de Antel. Estos corresponden a tres *scripts* hechos en bash (extensión sh).

A continuación, se muestra la configuración de crontab del servidor:

```
@reboot sh /root/bash/runbackend.sh
@reboot sh /root/bash/runcelery.sh
@reboot sh /root/bash/runfrontend.sh
```

Para el *backend* se utilizó el *script* runbackend.sh:

```
#!/bin/bash
cd /root/repositoriounico
python3 back/manage.py runserver 0.0.0.0:8000 > /root/bash/salida.txt
```

Esto inicializa un servidor web básico que responde en el puerto 8000, accesible desde cualquier IP.

Para el frontend se utilizó el *script* runfrontend.sh

```
#! /bin/bash
export NVM\_DIR="$HOME/.nvm"
[ -s "$NVM\_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM\_DIR/bash\_completion" ] && \. "$NVM_DIR/bash_completion"
cd /root/repositoriounico/front
export NODE_ENV=development
npm start
```

Esto inicializa un servidor web básico para levantar la aplicación hecha en React.

Por último, un tercer *script* en bash que inicializa celery llamado runcelery.sh:

```
#!/bin/bash
cd /var/www/backend
service rabbitmq-server start
celery -A backend.celery worker -l INFO
```

Es necesario destacar que este despliegue no está bien realizado. Es decir, esto es solo utilizable en un entorno de desarrollo ya que los servidores web utilizados no están preparados para un entorno de producción. Sin embargo, a efectos prácticos, como solo estaba siendo utilizado por pocas personas fue suficiente mientras duró el desarrollo y evitaba tener que estar constantemente recompilando *frontend* y *backend* ante cualquier cambio.

D.4. Deploy final

Para el deploy final, se debía escoger un servidor web que sea capaz de correr aplicaciones web hechas en Python.

En primer lugar, se optó por Apache, ya que uno de los integrantes del equipo poseía conocimiento en despliegue de aplicaciones en este servidor web, el cual posee la ventaja de ser es el más utilizado y además es de código abierto. Para ello se instaló mod_wsgi[8] y se configuraron los virtual host[9] tanto para el *backend* como para el *frontend*. En cuanto a este último, se consideró que el equipo anterior no hizo un despliegue del *frontend* apto para producción, ya que el comando *"npm start"* inicializa un servidor web ligero, que no está diseñando ni optimizado para manejar grandes volúmenes de tráfico. En su lugar, se generó un compilado de la aplicación React utilizando el comando *"npm build"*. Esto genera archivos html, javscript, entre otros, que son aptos para poner en un servidor web de producción y finalmente se colocó en una carpeta a la cual apunta el *virtual host* apache, en el puerto ochenta.

Sin embargo, no fue posible ejecutar simulaciones de manera correcta. Es decir, si bien era posible acceder a la web (*frontend*) y al django admin (*backend*) y el *frontend* se comunicaba correctamente a los *endpoints* del *backend*, fallaba únicamente la ejecución de simulaciones.

Luego de múltiples intentos los cuales implicó creación/lectura de logs, cambios de permisos y usuario en archivos, cambio de entorno virtual de Python, reinstalación de dependencias, reinstalación de Monolix Suite en una carpeta de distinto a root, creación de un proyecto Django de cero que únicamente utilizaba Rpy2, se observó que en todos los casos siempre fallaba el plugin Rpy2 de Python, en la línea de código donde estuviera su *import*. Dedicando una considerable cantidad de horas y sin lograr un despliegue exitoso mediante esta plataforma, se optó por buscar otro servidor web apto para aplicaciones Python.

Finalmente se optó por realizar el deploy del *backend* al igual que el equipo anterior utilizando uWGSI

Además, no se utilizó Tmux, en su lugar, se creó un servicio de Linux. En este sentido, utilizando el comando *"service uwsgi start"* se inicializa el servidor de aplicación uWSGI que sirve el *backend*.

Uno de los puntos que mencionaba el equipo anterior a mejorar era utilizar Nginx como servidor web *proxy* inverso y siendo que con Apache no pudimos hacer un despliegue del *backend* exitoso, se aprovechó para instalar Nginx en el servidor de Antel y hacer un despliegue del frontend sobre Nginx, desinstalando Apache.

Se configuró Nginx como servidor proxy inverso del servidor uWSGI. En este sentido, tanto el *backend* como el *frontend* son servidos por Nginx.

[8] `https://modwsgi.readthedocs.io/en/master/`
[9] `https://httpd.apache.org/docs/2.4/vhosts/`

Habiendo hecho pruebas y observando el correcto funcionamiento del despliegue, finalmente se armó un *script* en bash con todos los pasos del despliegue el cual realiza las siguientes tareas:

1. Git pull de los últimos cambios
2. Borrar el despliegue del *backend* anterior (archivos)
3. Borrar el despliegue del *frontend* anterior (archivos)
4. Copiar archivos de *backend* clonados de git a la carpeta /var/www/backend/
5. Actualizar archivo api-service.ts para que el frontend apunte a la IP del servidor de Antel
6. Correr el comando "npm build"
7. Copiar el "build" de la aplicación react a la carpeta /var/www/frontend
8. Generar archivos static necesarios para el Django Admin Panel
9. Reiniciar servidor Nginx
10. Reiniciar servidor Uwsgi

Por último, es necesario destacar que con este despliegue se logró alcanzar un mejor rendimiento que el obtenido durante la fase de desarrollo. Esto se debe a que uWSGI se configuró para correr con más de un proceso y un único hilo de ejecución. Si bien se intentó también aumenta la cantidad de hilos, se observa que al hacer esto falla la extensión RPY2.

En particular, se obtiene el siguiente error:

Conversion rules for rpy2.robjects appear to be missing. Those rules are in a Python contextvars.ContextVar. This could be caused by multithreading code not passing context to the thread. Check rpy2 documentation about conversions.

Investigando en foros de Rpy2, esto ocurre porque Python utiliza un bloqueo (conocido como GIL) para evitar la ejecución concurrente de bytecode, y al llamar a C (que es lo que ocurre al ejecutar código R a través de rpy2), por defecto, GIL no se libera hasta que ese código regrese.

Ahora bien, es posible liberar el GIL desde una extensión en C, pero R es muy poco amigable con el multi hilo (en el sentido de que "producirá un fallo de segmentación muy rápidamente"). Debido a esto, el esfuerzo para implementar una liberación del GIL y manejar todas las posibles formas en que la evaluación concurrente de R podría ocurrir. En su lugar, se recomienda utilizar procesos y no hilos, esto es lo que se hizo en el despliegue en el servidor de Antel para que escale en la cantidad de usuarios. Por lo tanto, en el archivo de configuración de uWSGI se estableció:

```
\# Procesos y subprocesos de uWSGI \\
processes = 48 \\
threads = 1 \\
```

Se declaran 48 procesos ya que el servidor de Antel si bien inicialmente debía ser un plan de cuatro núcleos, por alguna razón finalmente luego de los múltiples cambios en la plataforma de Antel se nos estableció en 48, se entiende que puede haber sido un error por parte de la empresa estatal [67] [62] [68].