UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SISTEMA DE CAPTURA Y ANÁLISIS SIMULTÁNEO DE DATOS MÉDICOS NO ESTRUCTURADOS PARA INVESTIGACIONES EN FALP USANDO REDCAP

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

ALEJANDRO IGNACIO TORRES SALAS

PROFESOR GUÍA:
NELSON BALOIAN TATARYAN

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
RODRIGO FREZ PULGAR

SANTIAGO DE CHILE
2024

Resumen

La Fundación Arturo López Pérez (FALP) es una institución que entrega un servicio integral de tratamiento del cáncer y que posee un área de investigación para mejorar la terapia contra el mismo. El proceso de estudiar nuevos procedimientos para tratar el cáncer en FALP requiere de la obtención de variables específicas para cada uno de los pacientes pertenecientes a una investigación. Tales datos son adquiridos a través de los reportes clínicos de cada paciente en un sistema de registros clínicos electrónicos (RECs) y son agregados a un formulario perteneciente a un proyecto de investigación en la plataforma REDCap, que es un sistema de captura de datos electrónicos (EDC). Sin embargo, el procedimiento previamente mencionado presenta dos problemas: La eficiencia en la obtención de variables a partir de datos médicos no estructurados y la privacidad y ética en la revisión de informes médicos en el sistema de RECs. Esto sucede, respectivamente, a que, por un lado, el sistema de registros clínicos electrónicos utilizado por FALP llamado REMI puede llegar a tardar entre uno a tres minutos en la obtención de un reporte clínico, y generalmente, se desea revisar una cantidad variable de informes y de pacientes de tamaño considerable. Por otro lado, la plataforma de RECs permite buscar la información médica no anonimizada de cualquier paciente por parte de los investigadores, lo que puede causar un problema legal en los estudios realizados. Además, REMI al ser propiedad de una empresa externa, no permite implementar una característica que limite el alcance de los usuarios al acceso de reportes clínicos. Por ello, el objetivo de este trabajo es desarrollar un sistema de captura y análisis de datos médicos no estructurados que permita a los investigadores una revisión eficiente de las fichas clínicas anonimizadas y el llenado de formularios de REDCap dentro de la misma aplicación. Con este fin, se desarrolló una aplicación que considera usuarios de tipo administrador e investigador y que contiene cuatro vistas, las cuales, para cualquier usuario, permiten iniciar sesión dentro del sistema, visualizar y entrar a los proyectos de REDCap asignados a un investigador, ver el detalle de registros de un estudio en particular, y capturar y analizar datos médicos no estructurados para un paciente usando, respectivamente, un formulario asociado a un registro de REDCap y un apartado que posibilita revisar los informes clínicos usando un buscador y filtros. Finalmente, se logró construir el sistema y probarlo con usuarios, pudiéndose restringir el acceso a la información clínica de pacientes fuera de las investigaciones del usuario y anonimizar el RUT de los pacientes estudiados en la visualización de los datos médicos. Sin embargo, no se pudo evaluar la eficiencia de la estructuración de datos, ya que, sucedió un bloqueo de dirección IP por parte del servidor de REDCap de FALP debido a un exceso de consultas hacia este último. Queda como trabajo futuro desplegar el sistema en producción con las correcciones ya realizadas y probar la aplicación para comprobar que la disminución del tiempo de captura y análisis de datos médicos es efectiva.

A mi familia y las personas que me acompañaron durante este proceso.

Tabla de Contenido

1. Introducción **1**

1.1. Contexto 1

1.2. Problema 1

1.2.1. Eficiencia en el proceso de investigación 2

1.2.2. Privacidad y ética en la revisión de fichas clínicas 3

1.2.3. Limitaciones 3

1.2.4. Recursos actuales y proceso de captura de datos existente 4

1.3. Objetivos 4

1.3.1. Objetivo general 4

1.3.2. Objetivos específicos 5

1.4. Descripción de la solución planteada 5

1.5. Resumen de resultados obtenidos 6

1.6. Estructura de la memoria 6

2. Estado del arte **7**

2.1. Metodologías de desarrollo 7

2.2. Tecnologías consideradas 9

2.2.1. Backend 9

2.2.2. Frontend 10

2.2.3. Despliegue (*Deployment*) 11

2.2.4. REDCap y su API 11

3. Concepción de la solución **13**

3.1. Proceso de estructuración de datos 13

3.2. Pila tecnológica (*Tech stack*) 14

3.2.1. Backend 14

3.2.2. Frontend 14

3.2.3. Despliegue (*deployment*) 15

3.3. Identificación de requisitos de usuario y validación 16

3.3.1. Usuarios 16

3.3.2. Requisitos de usuario 16

3.4. Requisitos de *software* 17

3.5. Arquitectura de *software* 18

3.5.1. Backend 18

3.5.2. Frontend 18

3.6. Páginas de la plataforma 19

3.6.1. Común (Investigadores y Administradores) 19

3.6.2. Administradores 19

3.7. Arquitectura física 20

3.8. Modelo de datos 21

4. Implementación de la solución **23**

4.1. Creación del proyecto 23

4.2. Configuración de despliegue local con Docker 24

4.3. Configuración de despliegue en producción con Docker 25

4.4. Prototipos iniciales de *login* y vista de formulario y RECs 26

4.5. Autenticación y permisos de puntos finales 26

4.6. Vista formulario e información médica con búsqueda y filtros 27

4.6.1. Formulario 27

4.6.2. Información médica con búsqueda y filtros 30

4.7. Vista de un proyecto 35
4.8. Vista de proyectos 38
4.9. Vista de inicio de sesión 41
4.10. Barra de navegación y botones de volver atrás 42
4.10.1. Barra de navegación 42
4.10.2. Botones de volver atrás 43

5. Evaluación **44**
5.1. Marco evaluativo 44
5.2. Resultados 45

6. Conclusión **46**
6.1. Trabajo realizado y reflexión 46
6.2. Trabajo futuro 47

Bibliografía **48**

Índice de Ilustraciones

3.1. Esquema de la arquitectura de la solución junto al *stack* tecnológico 19

3.2. Esquema de la arquitectura física 20

3.3. Diagrama entidad relación 21

3.4. Modelo relacional para usuario y proyecto 22

4.1. Formulario con componentes de fecha, selección, verificación, texto, radio, numérico, de cálculo y campo de texto deshabilitado 29

4.2. Formulario con nombre de instrumento y evento junto a el número de instancia de repetición 29

4.3. Menú inferior del formulario 30

4.4. Botones opcionales para guardar el registro y avanzar 30

4.5. Interfaz de historial clínico con componentes de acordeón 32

4.6. Datos del paciente sin información personal 32

4.7. Filtros de informes clínicos 33

4.8. Buscador por palabra, coincidencias y ordenamiento 34

4.9. Resaltado de palabras 34

4.10. Botón al seleccionar texto 34

4.11. Vista de formulario e información clínica del paciente 35

4.12. Paginación con dos botones de tres puntos 38

4.13. Vista de un proyecto longitudinal 38

4.14. Cuadro para añadir usuarios con uno de ellos agregado 40

4.15. Componente para agregar un nuevo proyecto 40

4.16. Vista de proyectos para un administrador 41
4.17. Interfaz inicio de sesión 42

Capítulo 1

Introducción

1.1. Contexto

FALP (Fundación Arturo López Pérez) es "una Institución sin fines de lucro que busca otorgar acceso oportuno -en especial a las familias más vulnerables- a un tratamiento integral en un Centro Oncológico de alta especialización" [1]

Esta institución, además de prestar un servicio de tratamiento integral del cáncer, se apoya de la investigación y la docencia para desarrollar mejores tratamientos oncológicos [1] , y para ello, cuentan con distintas unidades de investigación como lo son la Unidad de Investigación Epidemiológica y Clínica (UIEC), la Unidad de Investigación de Drogas Oncológicas, la Unidad de Informática Médica y Data Science (IMDS) y la Unidad de Docencia.

Cada propuesta de estudio, que puede ser realizada tanto por una unidad de investigación como por cualquier profesional de la fundación, pasa por un proceso común:

Primero, se recibe un asesoramiento por parte de la UIEC para evaluar la factibilidad del proyecto y escoger una metodología de investigación adecuada [2]. Luego, se evalúan y revisan los aspectos éticos del proyecto por el Comité Ético Científico (CEC) quienes se aseguran de que el proyecto cumpla con las normativas éticas. Posteriormente, la IMDS extrae la mayor cantidad de información necesaria para los investigadores, que a su vez, crea los instrumentos de investigación y recolección de datos a través del sistema de captura electrónica de datos (EDC) REDCap, que permite manejar encuestas y bases de datos [3]. Finalmente, los investigadores a cargo del estudio completan los formularios con la información necesaria con los datos obtenidos por la IMDS y se ejecuta el proyecto.

1.2. Problema

Sin embargo, en el flujo descrito anteriormente suelen presentarse dos problemas:

1.2.1. Eficiencia en el proceso de investigación

Durante el asesoramiento por parte de la UIEC, si este último resuelve que la factibilidad del estudio no es posible de determinar debido a que hay incertidumbre acerca de la cantidad o de la existencia de las variables de investigación requeridas, se delega a la IMDS el análisis de disponibilidad de datos a través de los sistemas que posee esta unidad.

En el caso de que la información se encuentre estructurada, el proceso es manejable para el equipo de *Data Science* y pueden concluir acerca de la factibilidad. Sin embargo, si los datos no están estructurados, se realiza un análisis de la existencia o de la cantidad de información en los antecedentes de las fichas clínicas de los pacientes, que posteriormente debe ser corroborada por los investigadores, revisando las fichas de los pacientes al no tenerse certeza previa de que los datos identificados sean los necesitados.

Luego, posterior a la evaluación de factibilidad y aprobación del proyecto por parte del CEC, el equipo de *Data Science* procede a extraer, sistematizar y cargar aquellos datos identificados en el análisis de viabilidad del proyecto.

Si todos los datos necesitados se encuentran estructurados, entonces, se llenan los formularios creados por los investigadores a través de un proceso de carga de datos integrado con REDCap.

Sin embargo, si los datos no se encuentran estructurados, el equipo de la IMDS procesa los datos con cierta incertidumbre para luego revisar junto a los investigadores las muestras obtenidas a partir de las fichas clínicas, o bien, se entregan los campos que no están estructurados a los encargados del estudio para que evalúen caso a caso las fichas de los pacientes y estructuren las variables que son necesarias para la investigación.

Respecto a los casos en los que es necesario revisar las fichas de los pacientes, cuando se debe corroborar la factibilidad con los investigadores o cuando se procesan datos no estructurados, para visualizar los distintos documentos los examinadores utilizan una versión modificada del *software* ehCOS de NTT Data, denominada REMI, que es una "suite de soluciones de salud digital para Hospitales y redes de salud"[4] que dentro de sus funcionalidades, permite almacenar un Registro Clínico Electrónico de pacientes. Este registro permite almacenar los documentos de los pacientes, principalmente en archivos PDF, y en particular, aquellos pertenecientes a las fichas clínicas. Además, este *software* posee conexiones a otros sistemas internos para rescatar información adicional de los pacientes.

Sin embargo, la navegación y la obtención de archivos en el *software* es relativamente lenta, llegándose a demorar entre uno a tres minutos en obtener la visualización de un archivo, lo que retrasa el proceso de investigación al tener que ser revisada generalmente una gran cantidad de pacientes por estudio (60 por ejemplo) y un número de documentos variable. Adicionalmente, los datos buscados por los investigadores a veces son difíciles de obtener, y por tanto, se deben solicitar al equipo de Tecnologías de la Información (TI), que puede demorarse de uno a más días en entregar una respuesta.

1.2.2. Privacidad y ética en la revisión de fichas clínicas

Las investigaciones realizadas por la unidad de investigación pueden ser de carácter retrospectivo o prospectivo según el momento en el que ocurre su registro. Respectivamente, el primero se refiere a cuando el estudio se fundamenta en datos ocurridos en el pasado, mientras que la segunda, a cuando la información es anotada a la par que ocurre el fenómeno o los hechos programados para observar [5]

Cuando la investigación realizada es de prospectiva, se suele realizar un consentimiento informado (CI) para cumplir con la ley N°20.584 que regula los derechos y deberes de las personas respecto a acciones vinculadas a su atención de salud [6], para luego obtener los datos y que estos queden disponibles para su utilización dentro del estudio. Sin embargo, cuando es de retrospectiva, se deben revisar datos históricos para realizar la investigación, que en este caso, corresponden a las fichas clínicas.

Respecto a las bases legales del último procedimiento mencionado, la Comisión Ministerial de investigación en salud (CMEIS) en el año 2017 publicó un análisis normativo que indica que bajo la normativa legal vigente, el uso del contenido de las fichas clínicas para fines de investigación requiere de la aprobación de un Comité Ético Científico acreditado, de la autorización del director del establecimiento donde se realiza el estudio (custodio de las historias clínicas) y de la anonimización de la información si el uso es con fines investigativos [7]

Por ello, para la obtención de la información en una investigación en la que los pacientes no pueden firmar el CI por un motivo importante, generalmente se le es solicitado al custodio de los datos, que en este caso es el director médico, el acceso a las fichas clínicas de las personas seleccionadas para el estudio. Sin embargo, los datos médicos de los pacientes entregados por el custodio, no se encuentran anonimizados en las plataformas REDCap y REMI. Por un lado, REMI no limita a los usuarios a ver únicamente aquellos pacientes que fueron seleccionados para el estudio, lo que es un problema de privacidad ético y legal al tenerse de que un investigador podría revisar los documentos de cualquier otro paciente tratado en la institución. Por otro lado, en REDCap los investigadores pueden revisar el RUT de los pacientes que permite identificar a quién pertenece cada encuesta creada en la plataforma.

1.2.3. Limitaciones

Una limitación respecto al *software* en el que se almacenan los Registros Clínicos Electrónicos (RECs), es que la plataforma REMI pertenece a una empresa externa a la organización, y por tanto, no es posible de modificar para mejorar la eficiencia en la obtención de los documentos pertenecientes a las fichas clínicas, o bien, limitar el acceso a los investigadores al buscador de pacientes por RUT. Pedir el desarrollo de estas características al proveedor tendría un alto costo en HH y presupuesto, y por tanto, no es una posible solución para la organización.

Si debido a la limitante previamente mencionada se quisiera cambiar el *software* REMI que mantiene los RECs por alternativas como Nimbo [8], Reservo [9] o NextGen Office [10], esto no sería una solución, ya que, si bien, pueden presentar una posible disminución de los tiempos de carga de las fichas clínicas, al ser utilizado el sistema que almacena los RECs por gran parte de la organización de la FALP y no solo por los investigadores de los estudios, se requeriría de una capacitación masiva a todos los usuarios junto con la adaptación de los sistemas de FALP y el traspaso de datos al nuevo *software*. Además, reemplazar la aplicación REDCap por alternativas como OpenClinica [11] o Castor EDC [12] necesitaría el traspaso de todas las bases de datos e información de los proyectos al *software* reemplazante, además de volver a conectar los sistemas y capacitar a los investigadores.

1.2.4. Recursos actuales y proceso de captura de datos existente

Respecto a los recursos actuales, existe una base de datos creada y llenada periódicamente por el equipo de la IMDS, en la cual se almacenan registros médicos provenientes de distintos sistemas de información de la FALP. Entre estos *softwares* se encuentra la plataforma REMI que es la encargada de mantener los registros clínicos electrónicos y centralizar la información de los otros sistemas existentes a través de accesos directos.

Uno de los usos de la base de datos es el llenado de información en los registros de REDCap a través de un proceso denominado *Clinical Data Pull* (CDP) que es parte de la caracteristica *Clinical Data Interoperability Services* (CDIS) de REDCap [13]. CDP permite importar información de manera automática o manual desde un REC a la aplicación de REDCap a través de la asociación de un campo de la fuente externa a uno o más identificadores de registro de REDCap, lo que se debe realizar para cada uno de los proyectos existentes. Para la importación manual, se debe agregar en la interfaz del REC una vista que permita acceder a REDCap (por ejemplo, a traves de un elemento iframe), y de esta forma, llenar los formularios a la par que se revisan los informes médicos [14].

Sin embargo, por un lado, la importación automática de campos, solo se puede realizar con las variables que se tienen certeza de que son las buscadas por los investigadores, y por otro lado, debido a las limitaciones previamente mencionadas, no es posible implementar en REMI la integración con REDCap que ofrece CDIS.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar un sistema de captura y análisis de datos médicos no estructurados para facilitar las investigaciones realizadas en FALP que permita, por un lado, a los investigadores de manera anónima y eficiente la revisión de fichas clínicas y el llenado de formularios de investigación, y por otro lado, a los administradores gestionar los proyectos e investigadores asociados a estos últimos.

1.3.2. Objetivos específicos

1. Diseñar e implementar un *back-end* que utilice tanto las bases de datos que contienen la información de las fichas clínicas de los pacientes, como la *Application Programming Interface* (API) de la plataforma REDCap para el manejo de proyectos, instrumentos y registros.
2. Diseñar e implementar un *front-end* que permita a los investigadores analizar los datos no estructurados a partir de las fichas clínicas y completar los registros de cada instrumento de investigación de REDCap únicamente para los pacientes asignados. Además, debe permitir a los administradores gestionar los proyectos de REDCap.
3. Conectar el *front-end* y el *back-end* para que la plataforma sea funcional, que los investigadores puedan realizar el proceso completo de análisis de datos no estructurados de las fichas clínicas junto con el ingreso de la información buscada a REDCap, y que los administradores puedan gestionar proyectos.
4. Verificar tanto la mejora de al menos un 50 % de eficiencia respecto del proceso de análisis de datos no estructurados al ser utilizada la plataforma construida por los investigadores, como también, que en la plataforma implementada las fichas clínicas de los pacientes de cada estudio sean anonimizadas.

1.4. Descripción de la solución planteada

En suma, existen dos problemas descritos: Por un lado, la eficiencia del proceso de análisis de datos no estructurados por parte de los investigadores, y por otro lado, la exposición de la información de los pacientes a los examinadores en la plataforma REMI.

Para solucionar el primer problema respectivamente, se construirá una nueva plataforma que muestre rápidamente las fichas clínicas de los pacientes a través del uso de informes consolidados en una base de datos, que permita buscar datos en las fichas clínicas a través de palabras clave y un menú de filtro por tipo de documento, y que posibilite a la vez completar los formularios de REDCap usando los datos encontrados.

En cuanto al segundo, en la plataforma antes mencionada se permitirá a los investigadores el acceso a las historias clínicas anonimizadas solo de los pacientes asignados a los estudios a los que pertenecen. Únicamente los administradores podrán visualizar las fichas clínicas con el RUT de los pacientes.

Entregar las soluciones planteadas a los problemas descritos, por un lado, permitiría disminuir el tiempo de las investigaciones médicas realizadas en la FALP, y con ello, mejorar la eficiencia del personal de las unidades de investigación a cargo. Por otro lado, imposibilitaría problemas legales y éticos debido a una inadecuada utilización de la plataforma REMI por parte de algún investigador.

1.5. Resumen de resultados obtenidos

Se logró construir la aplicación mencionada anteriormente y se obtuvo retroalimentación de parte de los usuarios, pero no se pudo realizar una evaluación de la eficiencia del proceso de estructuración de datos al no poderse probar en producción debido a una limitación en la cantidad de peticiones realizadas al servidor de REDCap de FALP.

1.6. Estructura de la memoria

El presente informe de memoria contiene un total de 6 capítulos. El primero es el actual y corresponde a la "Introducción". Luego, se presenta el segundo, denominado "Estado del arte", en el que se discute y presenta tanto la metodología de desarrollo usada en el proyecto, como las tecnologías utilizadas en la aplicación. A continuación, se encuentra el tercer capítulo "Concepción de la solución", que, primero, define y establece el alcance de este trabajo respecto al proceso de estructuración de datos, y luego, describe y justifica las tecnologías, arquitecturas, interfaces y modelos considerados en la aplicación. Posteriormente, se describe el cuarto capítulo llamado "Implementación de la solución", que presenta la solución desarrollada justificando su diseño en distintas secciones. En seguida, está el quinto capítulo denominado "Evaluación", en el que se presenta la forma que se debe evaluar el trabajo desarrollado y los motivos de por qué no pudo ser evaluado el trabajo realizado. Por último, Se encuentra el sexto capítulo de nombre "Conclusión", en el que se resume el trabajo realizado, se reflexiona sobre las lecciones y resultados obtenidos de la presente memoria, y se da a conocer trabajo futuro para mejorar la aplicación.

Capítulo 2

Estado del arte

En el presente capítulo se discutirá y presentará la metodología de desarrollo usada para el proyecto, se mostrarán las tecnologías consideradas para la confección del *backend*, *frontend* y despliegue, y finalmente, se describirá tanto REDCap como su API.

2.1. Metodologías de desarrollo

La ingeniería de *software* se puede definir como el "establecimiento y puesta en práctica de los principios y metodologías que nos lleven a un desarrollo eficiente de software en todas las etapas" [15]. Debido a que esta última permite realizar un desarrollo eficiente, es necesario establecer cuál metodología utilizar correctamente dentro de todas las posibles a escoger.

Existen distintas metodologías de desarrollo de *software* que pueden ser aplicadas dependiendo de ciertas variables del proyecto, como el tamaño del proyecto, la implicación del cliente, restricciones de tiempo y flexibilidad de los requisitos [16].

Estas metodologías se pueden separar en dos categorías: Tradicionales y ágiles. Las primeras corresponden a aquellas que establecen procesos rigurosos, lineales y que suelen ser poco flexibles ante cambios. Además, requieren de una amplia documentación de todo el proyecto, planificación y control del mismo, y definición de requisitos, tal que todo lo mencionado previamente es establecido en una fase inicial del proyecto. Ejemplos de aquello son el modelo de cascada, incremental y espiral [17].

Las metodologías ágiles surgen como una alternativa a las tradicionales y buscan evitar los problemas principales que surgen de estas últimas. Se basan en que los procesos de desarrollo sean adaptativos [15] y se caracterizan por su metodología iterativa, es decir, que en cada ciclo se realiza una entrega que pueden probar los usuarios y entregar retroalimentación para los siguientes ciclos. Cada ciclo se compone de fases como "la adquisición de requisitos, diseño, verificación y entrega" [18].

Para elegir una metodología adecuada y un plan de trabajo efectivo, ya sea tradicional o ágil, se analizaron las variables previamente mencionadas para el proyecto:

- Flexibilidad de los requisitos: Si bien, existe un alto grado de especificidad respecto a los requisitos de usuario que debe cumplir la aplicación, estos requerimientos son únicamente especificados por el cliente que corresponde a una parte de los usuarios finales. Por tanto, es necesario que los requisitos tengan cierta flexibilidad.
- Implicación del cliente: Es necesaria una alta implicación del cliente en el diseño y construcción del *software*, pues, este último es parte de los usuarios finales de la aplicación y es quien conoce a fondo la problemática que llevó al planteamiento de este proyecto. Por tanto, es crucial que se encuentre para otorgar retroalimentación de los procesos de desarrollo del sistema.
- Tamaño del proyecto: Es pequeño, pues presenta un alcance acotado y se encuentra bien definido desde un comienzo.
- Restricciones de tiempo: El proyecto tiene un plazo máximo previsto de 10 semanas.

De las características mencionadas, por una parte, la flexibilidad de los requisitos y la implicación del cliente serían manejadas de mejor forma por una metodología ágil, debido a que, estas últimas permiten una mayor flexibilidad respecto a los requisitos e involucran tanto al cliente como los usuarios en el proceso iterativo de desarrollo. Por otra parte, el tamaño del proyecto y las restricciones de tiempo permiten la utilización de una metodología tradicional o ágil.

El desglose de variables apunta a que una **metodología ágil** sería lo más adecuado para realizar el proyecto, permitiendo probar el *software* con distintos usuarios de la aplicación, dar flexibilidad a los requisitos y obtener retroalimentación temprana de las características implementadas en cada iteración.

Por ello, se escogió en un inicio la metodología **Kanban**, la cual es centrada en la mejora continua y utiliza con este propósito un tablero en el que se tiene una lista de tareas pendientes, y se extraen para ser realizadas en un flujo constante de trabajo. Al tablero previamente mencionado se le denomina "Kanban" y se configura tradicionalmente de tres columnas que representan tareas "por hacer", "en progreso" y "finalizadas", aunque pueden ser más. Las tareas son representadas por tarjetas y van pasando por las columnas hasta llegar a ser finalizadas [19].

Se eligió la metodología Kanban debido a sus principales prácticas, como permitir visualizar el estado del proyecto, gestionar el flujo de trabajo e implementar ciclos de comentarios. Respectivamente, el primero ayuda a conocer el estado del proyecto de manera visual y tomar decisiones de manera temprana respecto al tiempo o dificultad de las tareas. El segundo, posibilita la optimización del trabajo realizado durante la semana al poder avanzar en más de una tarea a la vez, cuando una de ellas se encuentra pendiente de revisión o no se sabe exactamente cómo avanzar con esta. El último permite obtener comentarios del cliente de las tareas realizadas respecto a la correctitud de estas y de la existencia de errores en el trabajo desarrollado.

Si bien, esta fue la metodología escogida para realizar el proyecto, al avanzar con el mismo no se terminó de cumplir con uno de los principios fundamentales descritos en el manifiesto ágil, el cual es "satisfacer al cliente mediante la entrega temprana y continua de

software con valor" [20]. Esto último ocurrió, debido a que, no se logró entregar *software* funcional que puedan utilizar usuarios de la aplicación en un tiempo menor a dos meses. La razón principal de este suceso es que varias de las tareas especificadas necesitaron más del tiempo estimado para su realización, y que integrar pruebas de usuarios junto a una retroalimentación posiblemente haría sobrepasar el plazo máximo para entregar todas las funcionalidades indispensables de la aplicación.

Por los motivos previamente mencionados, no se realizaron entregas continuas utilizables del *software* construido, asemejándose más la metodología a una tradicional de **cascada** pero conservando el uso de un tablero Kanban para seguir el progreso de las tareas junto con las reuniones semanales con el cliente para recibir comentarios de los avances.

El modelo de cascada divide el proceso de desarrollo de *software* en distintas fases que siguen una continuidad lineal, tal que cada etapa depende de la anterior a excepción de la primera. Este proceso es conocido por ser uno de los menos flexibles debido a su estructura de fases en forma de "cascada" [21]. La metodología de cascada, si bien, originalmente se describió con 5 procesos [22], se ha descrito posteriormente con un total de 7 fases, las cuales son de requerimientos, análisis, diseño, implementación y programación, pruebas, operación y desarrollo, y mantención [23].

Debido a que en el momento en el que se decidió optar por esta última metodología ya se habían obtenido los requisitos del proyecto, el diseño y parte de la implementación de las funcionalidades, se tomó esta metodología desde la fase de implementación y programación hasta la etapa de mantención.

2.2. Tecnologías consideradas

2.2.1. Backend

Para el *backend*, se debe considerar tanto la tecnología que concentrará la lógica de negocios, como el sistema gestor de bases de datos a utilizar para la aplicación.

Lógica de negocios

Respecto a la herramienta que permitirá obtener y entregar información al *frontend*, se tomó en consideración el factor de conocimiento del equipo de desarrollo de FALP respecto al lenguaje a ser utilizado y las librerías. Debido a esta restricción, se evaluaron dos posibles *frameworks* del lenguaje Python, los cuales son Flask y Django. Por una parte, Flask corresponde a un *microframework* de Python, es decir, que no requiere de herramientas o librerías adicionales para funcionar. Si bien, no posee una capa de abstracción para bases de datos u otras funcionalidades que suelen ofrecer *frameworks* de *backend*, sí permite añadir librerías de terceros que realicen tales tareas [24]. Por otra parte, Django es un *framework* de alto nivel de desarrollo *web* gratis y de código abierto que provee distintas características que son comunes en el desarrollo de aplicaciones alojadas en la red. Por tanto, esta opción necesita

mayor espacio para ser instalado, pero a cambio permite obtener un desarrollo más rápido del sistema [25].

Base de datos

A causa de los requerimientos del proyecto, la mayoría de los gestores de bases de datos relacionales más populares y usados en el mercado pueden cubrir las necesidades del mismo, ya que, la cantidad de información a ser almacenada y su estructura no es alta ni compleja respectivamente. Desde un principio, se decidió usar o implementar SQL Server debido conocimiento acerca de este gestor por parte del equipo de FALP. Sin embargo, esto no se pudo llevar a cabo, ya que, por un lado, la base de datos usada para mantener los informes clínicos y antecedentes del paciente no se puede utilizar por motivos de seguridad, y, por otro lado, que SQL Server necesita una licencia de pago para ser usada en producción.

Por ello, se tomaron en consideración dos de los gestores de bases de datos más usados del mercado [26] a excepción de SQL Server: MySQL y PostgreSQL. Por un lado, MySQL es un gestor de bases de datos relacional desarrollado en C y C++ basado en una arquitectura cliente-servidor [27], y, por otro lado, PostgreSQL es un sistema de bases de datos relacional orientado a objetos *open source* desarrollado en C [28]. Ambos gestores poseen en común la característica de vistas, procedimientos almacenados, desencadenantes, varios tipos de datos y transacciones. Sin embargo, PostgreSQL presenta características más avanzadas en cada uno de los aspectos. Además, este último resulta ser más adecuado cuando se necesitan realizar escrituras frecuentes y consultas complejas, mientras que, MySQL es más propicio para proyectos con menos usuarios o que necesitan más lecturas sobre el contenido almacenado que escrituras [29].

2.2.2. Frontend

Para elegir la herramienta que permitirá la interacción de los distintos usuarios con la aplicación, se centró la búsqueda en dos aspectos esenciales, que son la rapidez en el desarrollo y la mantención del *software* creado por parte del equipo de *Data Science* de FALP. Si bien, existen distintas librerías y entornos de trabajo que permiten un rápido desarrollo como lo son React, Vue y Django, debido a que la arquitectura comúnmente utilizada por el equipo de FALP es cliente-servidor entre *backend* y *frontend*, se decidió solo elegir entre React y Vue. Respectivamente, El primero corresponde a una librería de Javascript centrada en la construcción de interfaces de usuario tanto *web* como nativas a partir de piezas más pequeñas denominadas componentes [30]. El segundo es un *framework* de Javascript, que al igual que React, permite construir interfaces de usuario *web*. Se caracteriza por ser altamente optimizado, versátil respecto a su tamaño y accesible por su amplia documentación [31].

2.2.3. Despliegue (*Deployment*)

Para desplegar la aplicación en producción, se optó por utilizar herramientas que permitan servir el sistema, incluso si este último aumenta de tamaño, es decir, que posibiliten la flexibilidad del *software*. Estas tecnologías consideradas corresponden a NGINX, Gunicorn y Docker.

En primer lugar, NGINX es un *software* que posee diversas funcionalidades, como hacer de servidor *web*, de balanceador de cargas o de *proxy* inverso. En segundo lugar, Gunicorn es un servidor de Python que usa el estándar WSGI y el protocolo HTTP para Python [32], y permite la conexión entre un servidor *web* como NGINX con una aplicación de Python que puede utilizar *frameworks* como Django o Flask. Por último, Docker corresponde a una plataforma que permite "desarrollar, distribuir y ejecutar aplicaciones". Esta herramienta separa la infraestructura del *software* desarrollado, para acelerar el proceso de prueba y despliegue de las aplicaciones [33].

2.2.4. REDCap y su API

Una restricción del proyecto presentado por FALP, corresponde al uso del sistema de recolección de datos REDCap. Para ello, la solución que será planteada posteriormente utiliza esta plataforma, pero a través de una API que sirve para realizar transacciones de datos entre la aplicación que será construida y REDCap.

Para comprender la API de REDCap y su uso, primero es necesario conocer para qué sirve el sistema de recolección de datos y cuáles son sus principales funcionalidades, las que se describen a continuación:

REDCap

Se define como una "metodología de flujo de trabajo y una solución de software diseñada para el rápido desarrollo y despliegue de herramientas de captura electrónica de datos en apoyo de la investigación clínica y traslacional" [34].

REDCap permite crear y gestionar proyectos de investigación a través de una plataforma que posee distintas configuraciones disponibles según los requerimientos del estudio. Un proyecto se compone principalmente de formularios denominados **instrumentos**, que son diseñados a partir de componentes de entradas de datos denominadas **campos**, como por ejemplo, entradas de texto, casillas de verificación, selectores de fecha, entre otros. Cada instancia de llenado de formularios, se denomina **registro** y normalmente cada instrumento es completado una única vez para un solo registro. Sin embargo, si la recopilación de datos del proyecto se configura como **longitudinal**, los instrumentos pueden tener la característica de ser **repetibles**, es decir, que para un registro en particular, pueden existir distintas instancias de llenado para el mismo formulario.

Si bien, los proyectos pueden consistir únicamente de varios instrumentos, REDCap en-

trega hasta dos niveles de abstracción más al seleccionar un diseño longitudinal: Eventos y Ramas. Los **eventos** corresponden a sucesos programados (o no), durante los cuales se capturan datos utilizando un conjunto de instrumentos designados (es decir, son un grupo de formularios). Mientras que las **ramas** son conjuntos de eventos que principalmente son utilizados cuando se emplean distintos grupos de tratamiento (por ejemplo, de control y experimental), o bien, cuando se realiza un estudio multicéntrico [35].

API de REDCap

La API de REDCap es un servicio *web* RESTful (que sigue los principios de una arquitectura REST, como ser de tipo cliente-servidor) [36] para desarrollar *softwares* que se comuniquen con la interfaz de usuario de REDCap guardando u obteniendo datos de esta última. Para poder interactuar con esta API, las aplicaciones que hagan uso de ella deben especificar una cadena de texto denominada ***token*** de 32 caracteres que es específico para un solo usuario y un único proyecto de REDCap [37].

La comunicación establecida con la API de REDCap es a través del protocolo HTTP con el método POST para ejecutar cualquier método disponible de la API, que se especifica dentro del cuerpo de la petición POST. Dependiendo del método especificado, se pueden establecer distintos parámetros habilitados para guardar u obtener distinta información de un proyecto de REDCap. Ejemplos de programas que pueden ser ejecutados en la API son: exportar la información de un proyecto, exportar instrumentos, importar registros, entre otros.

Capítulo 3

Concepción de la solución

En este capítulo, por un lado, se define y establece la forma en la que se abordará el proceso de estructuración de datos, y, por otro lado, se describen los resultados de la fase de concepción de la solución. Este último proceso mencionado, respectivamente, fue realizado a través de reuniones sucesivas con el equipo de FALP, obteniéndose información clave del proyecto que se tradujo en las diferentes secciones que se mostrarán a continuación. Estas corresponden a las tecnologías a ser utilizadas (pila tecnológica), identificación de requisitos de usuario y de *software*, la arquitectura de *software*, las interfaces de la plataforma, la arquitectura física y el modelo de datos del sistema. A continuación, se presentará cada uno de los puntos mencionados por sección en el orden descrito.

3.1. Proceso de estructuración de datos

Como vimos anteriormente, el problema principal de las investigaciones en FALP corresponden a la existencia de datos no estructurados y que gran parte de la información buscada por los examinadores se encuentra en estos registros. En particular, se estima que cerca del 80 % de los EHR son datos no estructurados que son "difíciles de procesar para un uso secundario"[38].

La solución presentada por la organización propone resolver el problema antes mencionado a través de la revisión de informes y llenado de formularios de forma manual. Sin embargo, actualmente el proceso de estructuración de datos está siendo también abordado de manera automatizada a través del uso de programas de procesamiento del lenguaje natural (NLP) [39]-[41], que corresponde a una rama de la informática que "combina la lingüística computacional con modelos estadísticos, de machine learning y deep learning", y que juntas permiten procesar el lenguaje en forma de texto para distintos propósitos [42].

Una posible mejor solución incluiría el procesamiento automático de datos mediante NLP junto a una parte de revisión manual, debido a que las aplicaciones de NLP para el procesamiento de texto tienen una precisión limitada [43]. Pese a ello, no se realizará un procedimiento automático debido a la restricción del tiempo y que en cualquier caso se necesitaría

implementar un sistema de revisión manual de los informes clínicos.

3.2. Pila tecnológica (*Tech stack*)

En el proceso de obtener los requisitos de usuario del proyecto, se definieron las tecnologías necesarias para construir la aplicación y que se ajustarían a las limitaciones y restricciones del proyecto. A continuación, se muestran las tecnologías escogidas y su justificación por secciones:

3.2.1. Backend

Para el diseño e implementación del *back-end*, se utilizará el *framework* de Python Django, el gestor de paquetes y dependencias para Python PDM, y el conjunto de herramientas Django REST framework (DRF). Se seleccionó Django por sus tres características principales: Permitir rapidez en el desarrollo, disponibilidad de diferentes mecanismos de seguridad en la aplicación y posibilidad de escalar el proyecto [25].

En primer lugar, la rapidez del desarrollo es crucial debido a la cantidad de tiempo requerida para la construcción de la aplicación y el número de características que esta debe tener. En segundo lugar, la seguridad de la aplicación es crucial debido a que se manejan datos sensibles de pacientes. En tercer lugar, la escalabilidad del proyecto es necesaria debido a que, es posible que se deseen agregar más características en un futuro, y por tanto, se necesite crecer y modificar la API.

Además de la API de Django, se utilizará el sistema de gestión de bases de datos MySQL, ya que, no se necesitan características avanzadas para almacenar y estructurar la información de la aplicación, se realizarán más lecturas que escrituras frecuentes en la base de datos y la cantidad de usuarios que utilizarán el *software* no es muy alta. Respecto a las otras tecnologías mencionadas previamente, por un lado, el gestor de paquetes y dependencias de Python PDM fue utilizado debido a dos de sus principales características, que son instalar y administrar paquetes de Python de manera similar a como lo hacen otros sistemas gestores de paquetes como NPM, y la creación de *scripts* de usuario [44] para, por ejemplo, dar formato al código del *backend*.

Por otro lado, se añadió la dependencia de Django REST framework [45], ya que, permite convertir fácilmente a Django a una *REST API* que es personalizable, que incluye políticas de autenticación y que posee una amplia documentación.

3.2.2. Frontend

En primer lugar, para el diseño e implementación del *front-end* se utilizará el *framework* de JavaScript Vue junto con el gestor de paquetes *Node Package Manager* (NPM) [46] y el *framework* Bootstrap [47] de CSS. Vue fue el framework seleccionado debido a dos de sus

principales características: Rendimiento gracias a su sistema de renderizado optimizado y su versatibilidad y escalabilidad [31]. Por una parte, debido al carácter reactivo de Vue y de su alta optimización, se podrán mostrar varios elementos en el navegador (como por ejemplo, una gran cantidad de documentos de texto de una ficha clínica) conservando la fluidez de la aplicación en su uso. Por otra parte, la naturaleza incremental de Vue permitirá escalar la aplicación en un futuro añadiendo librerías o nuevas componentes.

En segundo lugar, se escogió el gestor de paquetes NPM, por un lado, debido a que Vue es provisto por este administrador de dependencias junto a una amplia variedad de paquetes que suelen ser requeridos para la construcción de aplicaciones *web* [46]. Por otro lado, porque permite añadir *scripts* para, por ejemplo, construir las aplicaciones y dar un formato al código.

En tercer lugar, se eligió el *framework* Bootstrap, ya que, posee componentes visuales ya disponibles para su uso y que ahorran tiempo de desarrollo. Esta fue escogida debido a su amplio uso y documentación, además de el sistema de grilla [47] y la existencia de algunas componentes útiles para el proyecto como el "Accordion".

Otra de las razones por las que se escogió Vue y Django para el *frontend* y *backend* respectivamente, corresponde a que la organización posee otros proyectos en los que también se ocupan estos *frameworks*. De esta manera, se permitirá la mantención y el desarrollo futuro de la plataforma por el equipo de la IMDS.

3.2.3. Despliegue (*deployment*)

Por un lado, para permitir un despliegue de la aplicación consistente tanto en el ambiente de desarrollo como el de producción, se utilizará la plataforma Docker. De esta manera, el despliegue local y de producción serán equivalentes respecto a la máquina virtual en la que corren, permitiendo una integración y despliegue continuo de la aplicación. Además, este despliegue continuo es deseable, pues, si se encuentran características que mejoren el proceso de estructuración de datos, estas van a poder ser probadas en producción sin tener que realizarse mayores ajustes al despliegue de la aplicación para que esta funcione.

Por otro lado, para mejorar la velocidad de respuesta del navegador al momento de desarrollar la aplicación y de inicio del servidor del front end, se decidió utilizar la herramienta de construcción Vite. Esta última presenta estas dos mejoras y posee una plantilla para poder generar un proyecto de Vue que se compile con las características previamente mencionadas [48].

Si bien, el despliegue local y de producción son similares respecto a las aplicaciones que corren en el *backend* y *frontend*, para producción se utilizará NGINX para servir Vue, y Gunicorn en conjunto a NGINX para permitir el uso de Django.

3.3. Identificación de requisitos de usuario y validación

Antes de comenzar con el desarrollo del proyecto, se procesó la información entregada en las primeras tres reuniones que se mantuvieron con parte del equipo de la IMDS de FALP y se identificaron los tipos de usuarios de la aplicación junto a los requisitos de usuario claves del proyecto. A continuación, se explicitarán los tipos de usuarios del sistema y los requerimientos de usuario.

3.3.1. Usuarios

Existirán dos usuarios con capacidades distintas en la aplicación, que son los siguientes:

- **Investigador**: Puede visualizar y acceder a los proyectos a los que está asignado, llenar los formularios de REDCap para cada uno de los participantes del estudio asociados a un proyecto en particular y leer para cada paciente perteneciente a una investigación, su información anonimizada.
- **Administrador**: Puede crear proyectos de REDCap y designar investigadores a cada uno de los proyectos, adicionalmente a lo que puede hacer un investigador.

3.3.2. Requisitos de usuario

1. **Autenticación de usuarios**: Tanto los investigadores como los administradores pueden entrar a la plataforma mediante sus credenciales.

   Criticidad: Alta

   Prioridad: Media

   Usuarios: Administrador e investigador

2. **Llenado de formularios de REDCap**: Los administradores e investigadores deben poder llenar los formularios (instrumentos) de REDCap a través de la plataforma y guardarlos.

   Criticidad: Alta

   Prioridad: Media

   Usuarios: Administrador e investigador

3. **Asignación de investigadores**: Administradores pueden asignar uno o más investigadores a los estudios.

   Criticidad: Alta

   Prioridad: Media

   Usuarios: Administrador

4. **Ver y acceder a proyectos existentes de REDCap**: Administradores e investigadores deben poder ver y acceder a los proyectos en los que se encuentran asignados. Los administradores deben poder ver aquellos que han creado.

Criticidad: Alta

Prioridad: Media

Usuarios: Administrador e investigador

5. **Visualización de historial clínico**: Un Administrador o investigador puede ver la ficha clínica completa del paciente a ser analizado y navegar a través de los documentos.

   Criticidad: Alta

   Prioridad: Alta

   Usuarios: Administrador e investigador

6. **Anonimización de pacientes**: Investigadores no deben de poder identificar ni por nombre ni por RUT a los pacientes del estudio en todo el proceso de estructuración de datos. Administradores sí pueden ver el RUT en la vista de un proyecto.

   Criticidad: Alta

   Prioridad: Alta

   Usuarios: Administrador e investigador

7. **Búsqueda en ficha clínica**: Un administrador o investigador puede buscar identificadores o palabras clave dentro de los documentos del historial clínico.

   Criticidad: Alta

   Prioridad: Alta

   Usuarios: Administrador e investigador

8. **Integración de ficha clínica con formularios REDCap**: Un administrador o investigador debe ser capaz de visualizar el historial clínico y llenar el formulario de REDCap en conjunto en la misma página.

   Criticidad: Alta

   Prioridad: Alta

   Usuarios: Administrador e investigador

3.4. Requisitos de *software*

En reuniones posteriores, se fueron estableciendo los distintos requisitos de *software* a medida que se iban analizando los de usuario. Este proceso fue ocurriendo en conjunto con el desarrollo del sistema, a través de revisiones semanales con el cliente.

Como resultado de este proceso, se crearon distintos requisitos de *software* que permitieron concretar los requerimientos de usuario especificados a través de las tecnologías descritas previamente, conformando las distintas vistas de la aplicación que serán detalladas más adelante en este informe.

3.5. Arquitectura de *software*

Luego de ser identificados los requisitos de usuario y software preliminares junto a su importancia, se procedió a planificar la arquitectura del *software* de la solución que permita cumplir con las necesidades del cliente respecto al proyecto. Aquel esquema corresponde a una arquitectura de tres capas, que son interfaz de usuario, lógica de negocios y espacio de datos. A continuación, se mostrará la arquitectura, pero agrupando la lógica de negocios y espacio de datos en lo que se conoce como "backend" y la interfaz de usuario como "frontend" [49]:

3.5.1. Backend

La API de Django REST Framework se conecta a una instancia del gestor de bases de datos SQLServer y a otra de MySQL. La primera de ellas, corresponde a la base de datos de FALP que permite obtener el historial clínico de los pacientes junto a su información personal. La segunda, es una instancia local de MySQL que almacena a los usuarios, que pueden ser administradores o investigadores, junto a los proyectos creados. Django puede añadir nuevos usuarios o proyectos y generar asignaciones entre estos dos últimos.

Además, Django se conecta a la API de Google tanto para la autenticación de usuario como para otorgar el acceso a los *endpoints* de la aplicación, a la API de REDCap para manejar la información de proyectos, eventos, instrumentos y registros, a la base de datos de FALP para obtener el historial clínico del paciente, a MySQL con el fin de guardar u obtener información de usuarios y proyectos, y a la instancia de Vue para efectuar operaciones con los datos entregados por el *frontend*. Para establecer una comunicación con Vue, se utilizan distintos *endpoints* que permiten transferir información con los sistemas de información y aplicaciones previamente mencionados.

3.5.2. Frontend

En este apartado, únicamente hay una instancia de Vue que permite el uso de las diferentes vistas de la aplicación, mostrando características según el usuario que utilice el *software*. El frontend se conecta con Django para obtener información necesaria de REDCap, de las fichas clínicas y datos de pacientes, y de información del usuario que ingresó a la aplicación junto a proyectos asignados. También esta conexión permite modificar datos de REDCap o de usuarios y proyectos, como por ejemplo, agregar un nuevo usuario a la aplicación si es que este último no existía previamente.

3.6. Páginas de la plataforma

3.6.1. Común (Investigadores y Administradores)

- **Vista de inicio de sesión**: Se implementará una vista de acceso común para todos los usuarios de la aplicación.
- **Vista de proyectos**: Se confeccionará una vista que permita la visualización y el acceso a los proyectos e instrumentos de REDCap asignados.
- **Vista de un proyecto**: Se desarrollará una vista para un proyecto en específico, que muestre un resumen de registros del proyecto para cada uno de los instrumentos que posee.
- **Vista formulario e información médica**: Se creará una página que posibilite el llenado de un formulario de REDCap junto a la visualización de información médica de un paciente.

3.6.2. Administradores

- Dentro de la vista de visualización y acceso a proyectos se agregará, únicamente para los administradores, un apartado que permita crear proyectos de investigación y asignar a los examinadores que pertenecerán a cada estudio

A continuación, se muestra una figura que resume la arquitectura de *software*, las páginas de la plataforma, los usuarios y la pila tecnológica, sin considerar las herramientas de despliegue:

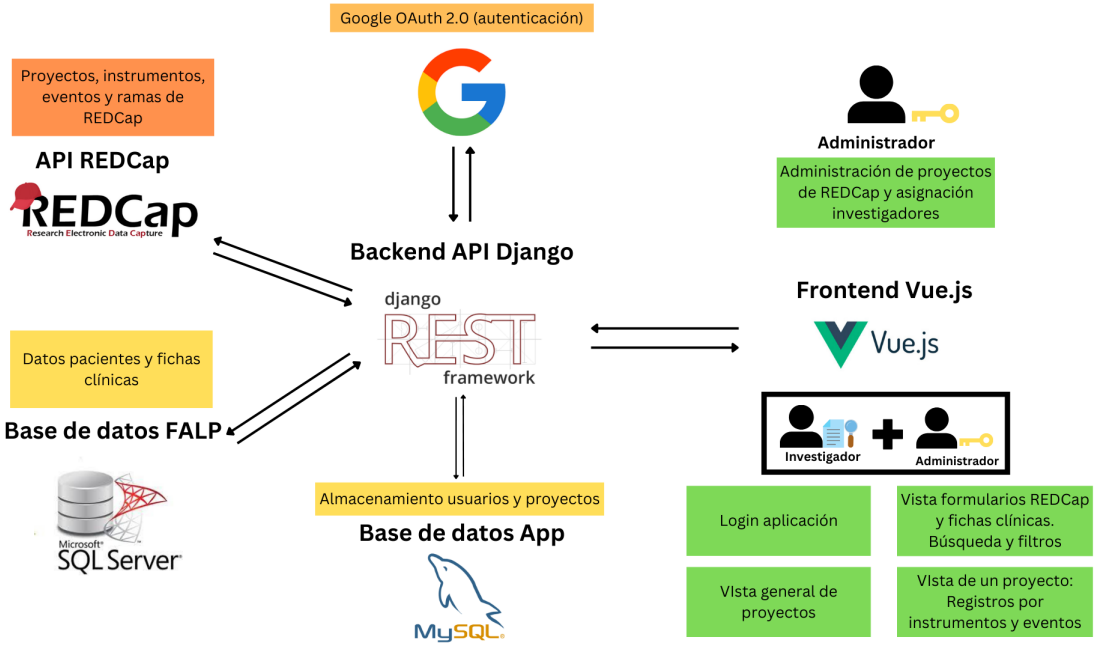

Figura 3.1: Esquema de la arquitectura de la solución junto al *stack* tecnológico

3.7. Arquitectura física

La arquitectura física se compone principalmente de tres componentes: El navegador del usuario, la red de FALP y el servidor de Google Cloud.

Respectivamente, el primero corresponde al navegador del dispositivo mediante el cual se conecta un usuario. Este último debe establecer una conexión previa con la red de FALP a través de un cliente de VPN. De esta forma, el navegador del usuario puede transmitir y recibir información hacia y desde el servicio de Docker que se encuentra en la red de FALP, que contiene a la aplicación realizada.

El segundo, es la red compuesta por todos los equipos y dispositivos computacionales de FALP. Dentro de esta, se encuentra el sistema de estructuración de datos que puede ser accedido a través del proceso de Docker que contiene un servidor de Django, Vue y MySQL en distintos puertos. Además, dentro de la red de FALP está el servidor que despliega la base de datos que contiene a los informes médicos utilizados en la aplicación junto a la información de los pacientes, y el servicio de REDCap de FALP.

El tercero es uno de los servidores de Google Cloud que contiene la configuración establecida para comunicarse de forma segura con la instancia de Django y Vue de la aplicación, y permitir la autenticación de usuarios a través del mecanismo de autorización provisto por la API de Google.

Los servicios y servidores, junto al navegador del usuario, se comunican a través del protocolo HTTP. A continuación, se muestra un diagrama que resume la arquitectura física de la aplicación:

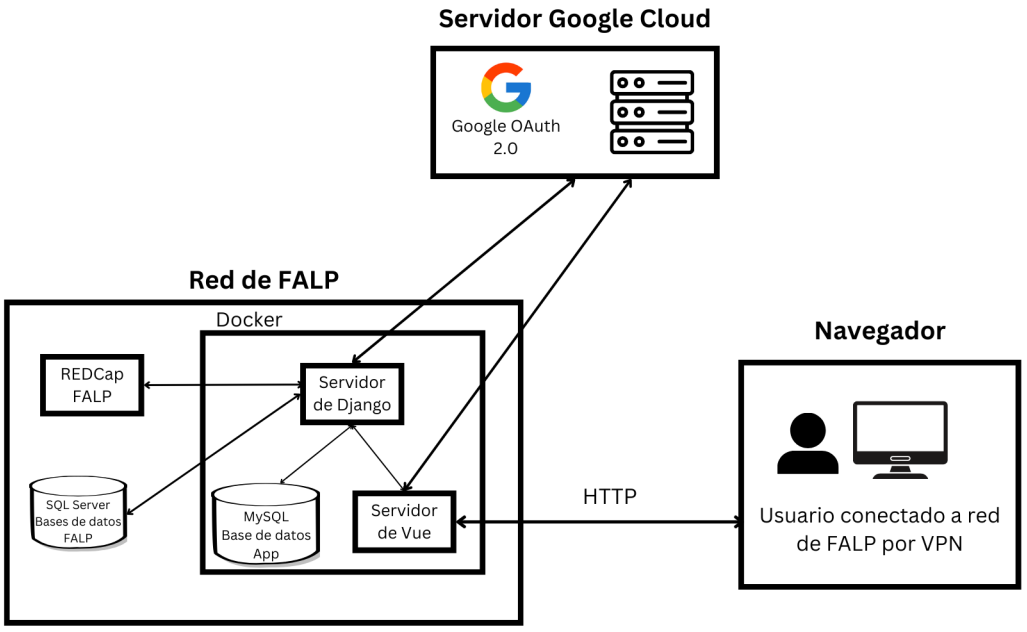

Figura 3.2: Esquema de la arquitectura física

3.8. Modelo de datos

El modelo de datos para el almacenamiento de la información de la aplicación, fue definido a través de la creación de un modelo entidad relación. En este proceso, se identificaron las entidades de Usuarios y Proyectos, y la relación Tienen, que se describen como sigue:

- Usuarios: La entidad Usuarios posee como atributos la clave primaria ID, nombre y admin.
- Proyectos: La entidad Proyectos tiene los atributos ID como clave primaria, token y creador.
- Tienen: La relación Tienen, posee como atributos la clave primaria ID, ID usuario e ID proyecto, tal que, estos dos últimos corresponden a identificadores de la entidad Usuario y Proyecto respectivamente.

Respecto a la cardinalidad entre las entidades y relaciones, los Usuarios pueden tener 0 o más proyectos, mientras que, los Proyectos poseen 1 o más Usuarios. A continuación se describe el diagrama entidad relación descrito previamente:

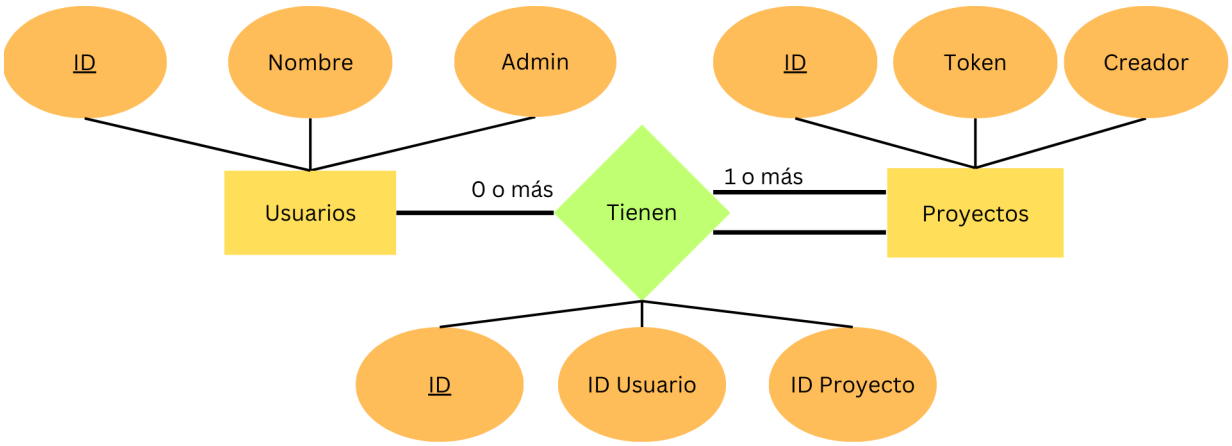

Figura 3.3: Diagrama entidad relación

El diagrama presentado se transformó en un modelo relacional a través de un esquema para cada entidad y relación, considerando las restricciones entre ellas. La implementación de los esquemas a partir de las entidades y relaciones se detalla a continuación:

- users_user: Se corresponde con la entidad Usuarios. “id” se asocia a ID y es la llave

principal de tipo “bigint”, “name” se corresponde a Nombre y es una llave única de tipo “varchar(64)”, y “admin” se asocia a Admin y es de tipo “tinyint(1)”.

- users_project: Se asocia a la entidad Proyectos. “id” se corresponde a ID y es la clave principal de tipo “bigint”, “token” se asocia a Token y es una llave única de tipo “varchar(32)”, y “creator_id” se corresponde a Creador y es una llave foránea de “id” del esquema users_user de tipo “bigint”.
- users_user_projects: Se corresponde a la relación Tiene. “id” se asocia a ID y es la llave principal de tipo “bigint”, “user_id” se corresponde a ID Usuario y es una llave foránea de id del esquema users_user de tipo “bigint”, y “project_id” se asocia a ID Proyecto y es una clave foránea de id del esquema users_project de tipo “bigint”.

El modelo relacional descrito se presenta como sigue, tal que las líneas segmentadas conectan una llave foránea al atributo original al que se relaciona:

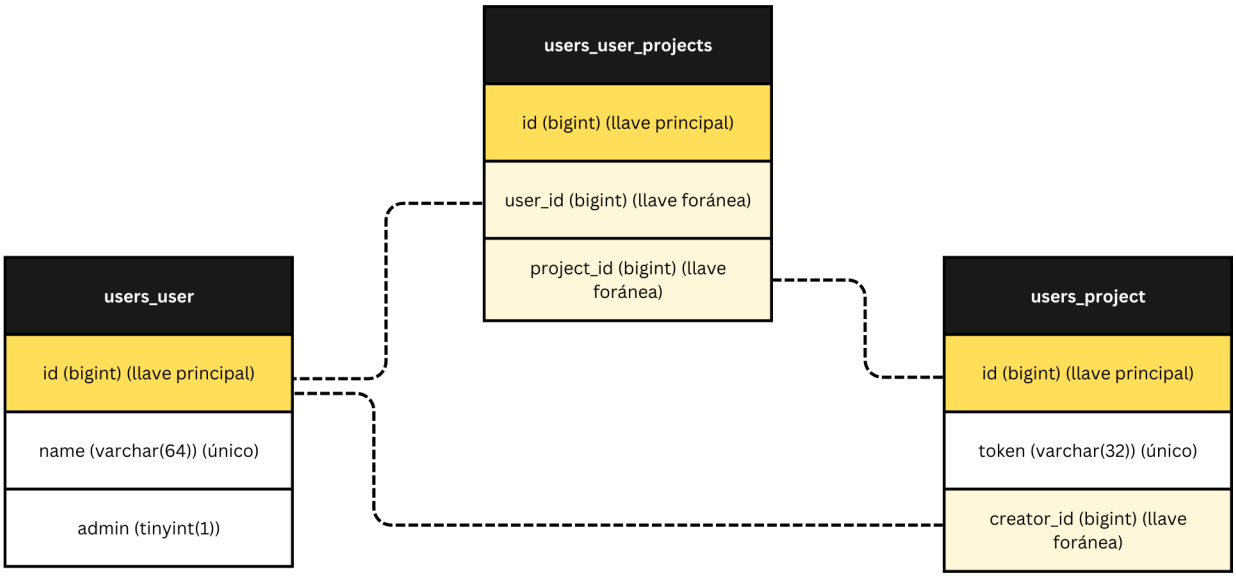

Figura 3.4: Modelo relacional para usuario y proyecto

Capítulo 4

Implementación de la solución

En el presente capítulo, se describe la solución implementada y se justifica su diseño en distintas secciones. Primero, se presenta el proceso de creación del proyecto, y los paquetes, librerías y lenguajes más importantes usados en la aplicación. Luego, se describen las configuraciones de despliegue local y producción, sus diferencias en los servicios utilizados, y la estructura utilizada. A continuación, se comenta acerca de los prototipos de *login* y de la interfaz de formularios e informes clínicos realizados en un comienzo, y cómo es que estos cambiaron luego de la validación del cliente. En seguida, se describe la autenticación y permisos requeridos para acceder a los *endpoints* de la API creada en Django. Finalmente, se describen, justifican y muestran las diferentes interfaces realizadas en la aplicación, tanto en el desarrollo del *frontend* con Vue como del *backend* con Django.

4.1. Creación del proyecto

Posterior al establecimiento de requisitos de usuario, se procedió a comenzar con el desarrollo del proyecto. Para ello, se creó un repositorio privado en Github para resguardar y almacenar la aplicación, y facilitar la continuación del desarrollo del mismo cuando se deseen agregar archivos u otros elementos. Para no adicionar ficheros que son generados al momento de desplegarse la aplicación, se configuraron archivos '.gitignore' que son reconocidos por Git, y que permiten omitir elementos que no deben ser añadidos al subir una nueva actualización del código fuente.

Luego, para comenzar con el *frontend* se inicializó un proyecto de Vue a través NPM y la herramienta de construcción Vite con una plantilla de Typescript. Esto se realiza con el propósito de, por un lado, obtener la característica de Vite Hot Module Replacement (HMR) que facilita el desarrollo al actualizar la vista en la que se está trabajando sin tener que actualizar la página, y por otro lado, permitir el uso de las características ofrecidas por Typescript como las interfaces o el establecimiento de tipos de funciones y variables.
Además, se instalaron y configuraron los paquetes Eslint y Prettier para dar un formato consistente y estandarizado al código realizado, y se instaló el *framework* Bootstrap para agilizar el proceso de creación de interfaces.

Posteriormente, para el *backend* se creó una aplicación de Django con el gestor PDM principalmente para administrar los paquetes y dependencias del proyecto a través de la línea de comandos, sin depender de la escritura de versiones en un archivo de requerimientos. Finalmente, se añadió el paquete de Django REST Framework para facilitar la creación de una API con Django y las dependencias Black, Flake8 e Isort para, al igual que en el *frontend*, dar un formato consistente y estandarizado al código.

4.2. Configuración de despliegue local con Docker

Luego de la creación del proyecto, se configuraron archivos que usa la plataforma Docker para permitir un desarrollo y despliegue eficiente y consistente de la aplicación. Para el *frontend* y *backend*, se creó un documento denominado "Dockerfile_local" en el proyecto de Vue y "Dockerfile" en el de Django respectivamente.

Por una parte, para la construcción del contenedor local de Vue, se utilizó una imagen de Node basada en Alpine que instala los paquetes declarados en el archivo "package.json", copia los archivos que se encuentran en el mismo directorio al contenedor y sirve localmente la aplicación de Vue.

Por otro parte, para la creación de la imagen local de Django, se utilizó una imagen de Python 3.11.4 basada en Debian en la que primero se instalan dependencias que no vienen con la imagen como Netcat y Curl, luego se establece el gestor PDM, se copian los archivos de configuración de este último y se instalan los paquetes declarados en tales documentos, y finalmente se corre el servidor de Django con el gestor PDM.

Con el objetivo de levantar los contenedores de Docker y configurar sus volúmenes y variables de ambiente respectivos, se creó un documento de Docker Compose en formato YML, en el cual, se definen los servicios de "*backend*", "*frontend*" y "*mysql*", y se le entregan configuraciones e indicaciones para operar, como por ejemplo, contexto de construcción, puertos, volúmenes, entre otros.

Respectivamente, para el primero, se configuraron manualmente los volúmenes para cada aplicación creada en Django para obtener un reinicio del servicio cada vez que se realicen cambios en el código y se asoció un archivo de variables de ambiente.

En cuanto al segundo, se configuró un volumen apuntando al directorio "src" donde se encuentran los archivos fuente del proyecto de Vue para permitir la característica HMR de Vite y obtener cambios en la visualización al modificar los documentos. Además, se configuró un archivo de variables de entorno.

Respecto al último, se añadió un volumen compartido para mantener la información almacenada en MySQL y se asoció un archivo de variables de ambiente para la configuración del servicio.

4.3. Configuración de despliegue en producción con Docker

El despliegue de producción de la aplicación, presenta algunas diferencias con la implementación local respecto a los archivos de construcción de los contenedores, los servicios y el documento de Docker Compose.

En primer lugar, en cuanto a los cambios en los archivos Dockerfile, por un lado, para el *backend* se confeccionó un archivo similar al de despliegue local denominado "Dockerfile.prod", con la diferencia de que se crean carpetas de archivos para almacenar documentos estáticos de Django, y que se crea y establece un usuario "app" que no tiene privilegios elevados, el cual se transforma en propietario de los archivos de la aplicación con fines de seguridad.

Por otro lado, para el *frontend* se realizó un archivo llamado "Dockerfile" que tiene construcciones en dos fases. La primera realiza los mismos pasos del archivo de Docker local a diferencia de la última línea, pues en vez de servir la aplicación de forma dinámica, lo hace de manera estática compilándose la aplicación y almacenando los archivos en un directorio de nombre "dist".

La segunda etapa obtiene una imagen de NGINX, crea un directorio principal, copia los archivos que fueron compilados en la fase anterior y copia un archivo de configuración de NGINX que define un servidor HTTP para servir los archivos de la aplicación de Vue. La razón de hacerse en dos fases, es que permite construir un contenedor final más ligero, sin mantener las herramientas que se utilizaron para compilar el programa y que no son necesarias para el despliegue en producción.

En segundo lugar, además de los servicios de Django, Vue y MySQL, se creó uno de NGINX para hacer de servidor de la instancia de Django. Este último servicio es construido a través de un archivo Dockerfile que usa una imagen de NGINX, elimina una archivo de configuración por defecto y copia una documento de configuración. Este último archivo define un servidor de *proxy* reverso que redirige las peticiones del navegador del usuario a la instancia de Django añadiendo encabezados del *host* original.

Por último, se construyó un archivo de Docker Compose similar al realizado para desarrollo local, definiendo los servicios "*backend*", "*frontend*", "mysql" y "nginx".

Respectivamente, para el primero, a diferencia de la configuración local, se cambió el archivo Dockerfile utilizado, se modificó el comando ejecutado por uno que corre un servidor de Gunicorn que redirige las peticiones a distintas instancias de la aplicación Django. También, se cambió el archivo de variables de ambiente por uno de producción y se definió un único volumen para los archivos estáticos de DRF. Para el segundo, se modificó el archivo Dockerfile por el de despliegue en producción y se eliminaron los volúmenes que habían sido definidos en el archivo de Docker Compose local. El tercero se mantuvo igual respecto al documento de composición local, mientras que, el último es un servicio que no existía previamente en el que se establece el archivo Dockerfile definido previamente para NGINX y se configura como volumen el mismo directorio utilizado para servir archivos estáticos en el servicio *backend*.

4.4. Prototipos iniciales de *login* y vista de formulario y RECs

Para establecer un diseño de la solución propuesta se realizaron prototipos a través de la plataforma Figma. Esto se realizó únicamente para el inicio de sesión y la vista de formulario e información médica.

Estos prototipos no fueron considerados en la solución posterior, pues, por un lado el inicio de sesión constaba de la creación y uso de credenciales de usuario y un sistema de recuperación de contraseña, el cual, a diferencia de la autenticación por cuenta de Google tomaría más tiempo de desarrollo. Por otra parte, si bien, la interfaz para los formularios fue validada, esto no fue así con el apartado de visualización de datos médicos, ya que, estos podrían ser demasiados y no sería lo óptimo su visualización y búsqueda. A cambio, se presentó la alternativa de mostrar la información en componentes desplegables (acordeón) con distintos colores según su fuente.

4.5. Autenticación y permisos de puntos finales

A fin de permitir el acceso a los *endpoints* creados en Django únicamente a los usuarios de la aplicación que pueden llamar a tales puntos finales, se añadió un archivo de autenticación para manejar el acceso a cualquier punto final de Django y entregar la instancia de usuario en la solicitud del flujo de autentificación, y otro de permisos para restringir el acceso a los *endpoints* según sea necesario.

Por un lado, en el documento de autenticación, se creó una única clase que hereda las propiedades de la autenticación base que entrega por defecto DRF, la cual, debe revisar si la solicitud del usuario presenta un código de acceso válido. Para ello, se reemplazó el método "authenticate" que recibe como parámetro un objeto de "request", del cual, se extrae el encabezado "HTTP_AUTHORIZATION" de la solicitud HTTP, que puede contener un *token* de refresco de la aplicación. En el caso de que exista una llave de refresco y sea validada con las credenciales de Google OAuth definidas en la aplicación, se obtiene la instancia de usuario desde la base de datos del sistema a través de su nombre de usuario, que es conseguido a través de la dirección de correo electrónico que es parte de la respuesta de la validación con el servidor de autenticación de Google. Si no hay una llave de refresco, entonces el usuario del flujo de autenticación es "None".

Por otro lado, respecto al archivo de permisos de puntos finales, en ella se creó una clase para verificar la existencia de un usuario en el flujo de autenticación denominada "IsAuthenticated", y otra para comprobar que la persona que ingresó es un administrador llamada "IsAdminUser". Ambas clases heredan de un permiso base que entrega DRF por defecto, y que posee un método denominado "has_permission" que recibe la solicitud y la vista del punto final desde la que se llama. Respectivamente, en la primera clase se comprueba la existencia de una instancia de usuario no nula en el objeto de solicitud, mientras que, en la segunda se verifica que la variable booleana "admin" del usuario obtenido en el parámetro de solicitud sea verdadera.

4.6. Vista formulario e información médica con búsqueda y filtros

Con las observaciones obtenidas con la presentación del prototipo se comenzó a confeccionar la vista de formulario y RECs.

Esta vista posee a un costado izquierdo el formulario de REDCap que debe ser llenado por el investigador, y a la derecha, una vista que permite navegar en los documentos de la ficha clínica del paciente actualmente analizado, utilizando un buscador de palabras y un menú para filtrar según la categoría de la información buscada. Sobre estas dos interfaces, se mostrará un cuadro con datos útiles del paciente para el investigador.

El proceso de construcción de la interfaz se dividió según las dos componentes principales de la vista: Formulario e información médica con búsqueda y filtros.

4.6.1. Formulario

Para construir el formulario, en primer lugar, se realizó una exploración de los *endpoints* provistos por la API de REDCap para identificar aquellos que permiten obtener la información de los formularios y registros, y la publicación de registros a REDCap. En esta búsqueda se encontraron los siguientes:

- **Exportar info. del proyecto**: Entrega el título, id, hora de creación y atributos de configuración del proyecto
- **Exportación de instrumentos**: Entrega una lista de cada instrumento creado dentro del proyecto, obteniendo el nombre de variable y de etiqueta del formulario.
- **Exportar eventos**: Entrega una lista de todos los eventos del proyecto. Cada evento posee un nombre de variable, de etiqueta y otros elementos. Sólo funciona para proyectos longitudinales.
- **Exportar los registros**: Entrega una lista de todos los registros de un instrumento en específico, es decir, de todas las respuestas de un formulario.
- **Importar los registros**: Dada información de un registro en formato JSON, CSV o XML permite almacenarlo en REDCap
- **Exportar metadatos**: Dado un instrumento, entrega sus metadatos, es decir, sus campos asociados y configuraciones establecidas en REDCap, como por ejemplo, tipo de validación de texto o mostrar número de deslizador.

En segundo lugar, para enviar las solicitudes a la API de REDCap según la información requerida por el formulario, se crearon *endpoints* que se definen en este informe por su método, ruta de acceso y el listado de clases de permisos utilizados respectivamente. Los puntos finales realizados para esta interfaz son los siguientes:

- GET 'forms_and_reports/record/' [IsAuthenticated]: Dado el número de proyecto, nombre de variable de un instrumento, número de registro y posiblemente nombre de evento y número de instancia de repetición, entrega un registro junto a sus metadatos, nombre del evento y etiqueta del instrumento.
- POST 'forms_and_reports/post_record/' [IsAuthenticated]: Dado un registro (en formato JSON por defecto) y el número de proyecto, guarda los datos del registro entregado dentro del proyecto.

En tercer lugar, se crearon las componentes necesarias para mostrar el formulario en Vue utilizando el *framework* de Bootstrap. Para ello, por un lado, se construyó una componente de Vue 'RedcapForm' para manejar el estado de todos los campos o entradas del formulario y realizar el envío de datos establecidos a REDCap utilizando el *endpoint* que permite el guardado de un registro. Los registros y metadatos del formulario son proporcionados por la vista 'FormAndDataView' que es la encargada de organizar el formulario y los datos de informes médicos

Por otro lado, se confeccionaron los campos mencionados con componentes de Vue, debido a que, en un futuro se agregarán más tipos de entradas que poseen distintas configuraciones y funcionalidades. Estas componentes creadas representan entradas de tipo texto, fecha, radio, multilínea de texto, número, verificación, selección, cálculo y texto deshabilitado para aquellas que no se encuentran actualmente implementadas, y fueron confeccionadas con componentes de Bootstrap a las que se les entregan los datos necesarios a través de los atributos enlazados 'v-bind' y 'v-model' de Vue. A continuación se muestra un formulario llamado "Form 1" con algunas de las componentes mencionadas previamente.

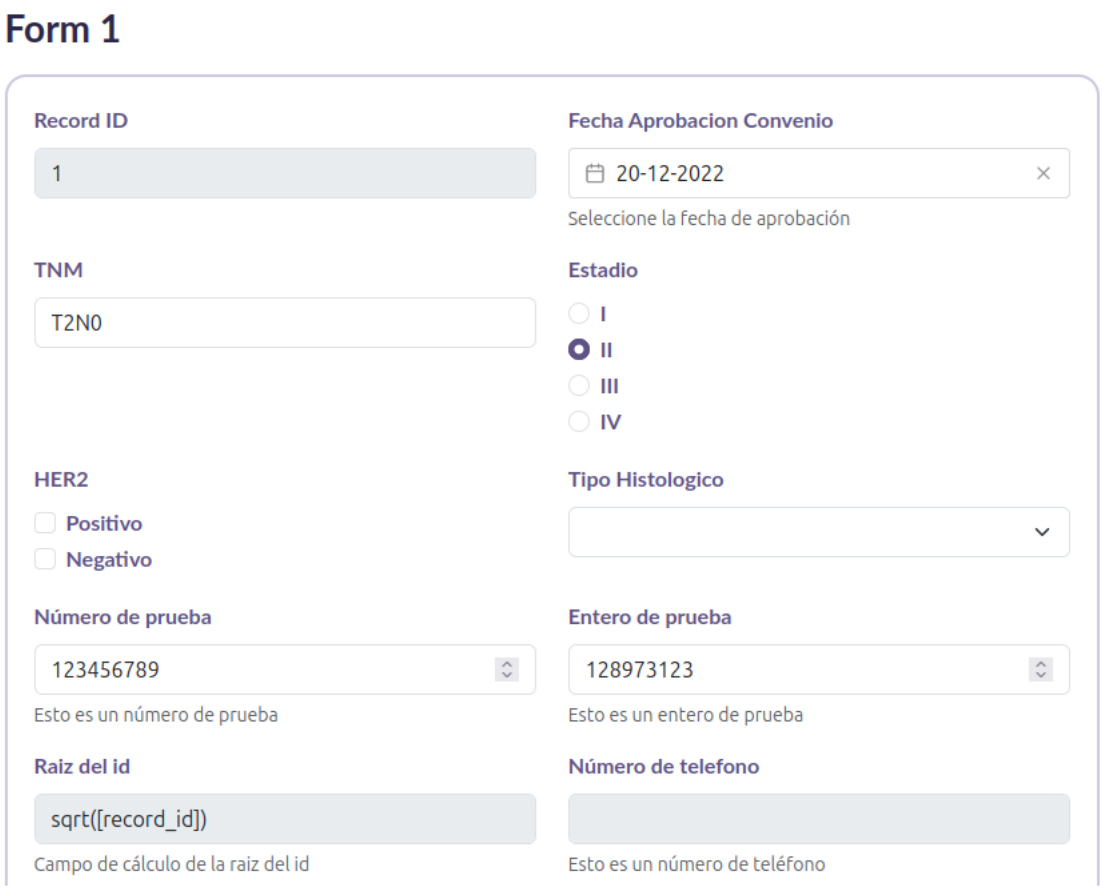

Figura 4.1: Formulario con componentes de fecha, selección, verificación, texto, radio, numérico, de cálculo y campo de texto deshabilitado

Todos los formularios poseen un campo de texto deshabilitado que muestra el número de registro para el instrumento actual. El título del instrumento aparece sobre el formulario junto al nombre del evento si se está en un proyecto longitudinal e instancia de repetición si es que el instrumento es repetible. Un ejemplo donde se muestran todos estos datos es el siguiente:

Figura 4.2: Formulario con nombre de instrumento y evento junto a el número de instancia de repetición

Debido a que las componentes de Vue mencionadas suelen repetir código de Typescript

respecto a sus propiedades, funciones e inicialización, se crearon distintos archivos de tipo *composable* de Vue, que permiten reunir y exponer la lógica común de las componentes. En estos archivos se almacena una interfaz que representa los metadatos de un formulario y otra que contiene los tipos de las propiedades (*props*) de las componentes que representan campos; una función de inicialización de componentes de entradas que no son de selección y otra para las que sí lo son; un método para obtener los valores de las componentes que representan un campo de REDCap; y una función para validar todas las entradas instanciadas para un formulario en particular.

Por último, dentro de la componente "RedcapForm" se utilizó una componente de botón para enviar la información actual del formulario a REDCap con validación previa de cada entrada. Además de este botón de guardado, se implementaron botones con opciones adicionales al almacenamiento del registro que también existen dentro de la plataforma REDCap. Estas alternativas permiten ir al siguiente formulario o ir al siguiente registro, y en el caso de tener instrumentos repetibles, ir a la siguiente instancia de repetición. Al seleccionar una de las alternativas, esta queda guardada dentro del almacenamiento local del navegador. El menú de botones se encuentra al final del formulario y se detalla en las siguientes imágenes.

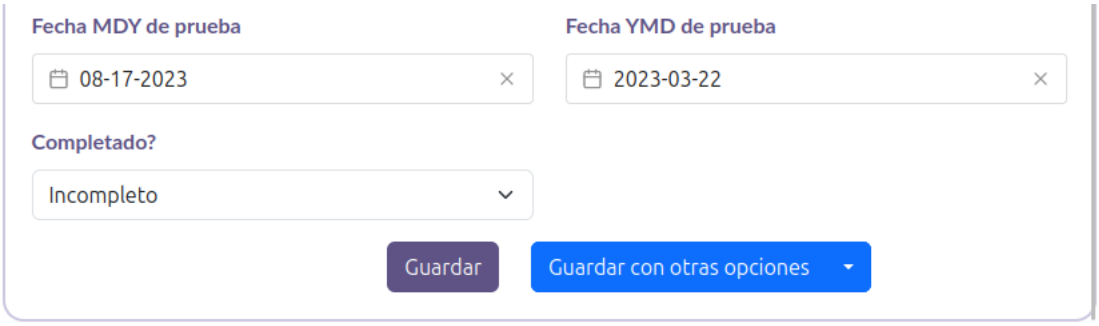

Figura 4.3: Menú inferior del formulario

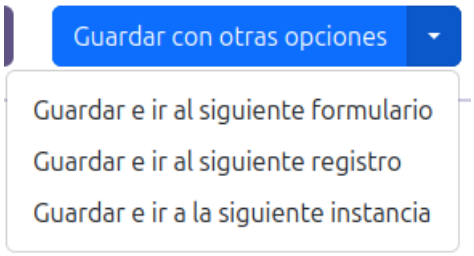

Figura 4.4: Botones opcionales para guardar el registro y avanzar

4.6.2. Información médica con búsqueda y filtros

Para este apartado se indicará cómo se obtiene la información de la historia clínica del paciente junto a sus datos personales. Luego, Se mostrará cómo se confeccionó la interfaz que muestra los informes clínicos, la información del paciente, los filtros y la búsqueda. Finalmente se mostrarán dos características que fueron agregadas luego de obtener retroalimentación de usuarios investigadores de la plataforma.

Obtención de informes médicos

En primer lugar, para recibir la información médica de un paciente y desplegarla, primero, se realizó una función que establece una conexión con la base de datos de SQL Server de FALP utilizando parámetros obtenidos a través de variables de ambiente. A continuación, se confeccionó un método que dada una consulta SQL y otras variables como el RUT, realiza una petición hacia la base de datos de FALP, obtiene la información y la procesa para entregar las columnas y filas obtenidas de la consulta, y un "tag" que generalmente es la fecha del informe dentro de un diccionario que tiene como llave los nombres de las tablas donde se encuentran los informes médicos. Luego, se creó una función que realiza varias consultas paralelas a través de la creación de múltiples *threads* que ejecutan el método previamente descrito, puesto a que, una consulta puede demorar varios segundos en realizarse. Posteriormente, para obtener los datos personales del paciente, se genera una consulta en la que se modifican los datos obtenidos para ser entregados correctamente al *frontend*.

Finalmente, se entregan los informes y la información del paciente en un diccionario a Vue a través del *endpoint* que se describe como sigue:

- GET 'forms_and_reports/ehr/' [IsAuthenticated] : Dada la posición de un proyecto y un número de registro, devuelve toda la información médica y personal registrada de un cierto paciente desde la base de datos de FALP.

Interfaz informes clínicos

En segundo lugar, para la interfaz que presenta los informes clínicos, se creó una componente de Vue que muestra un acordeón de Bootstrap dado un título, un cuerpo, un "tag", y un color que depende del tipo de texto médico. Se eligió este tipo de componente para mostrar la información debido a que se facilita la navegación por los documentos al tenerse que el texto puede ser abierto u ocultado, y que al mostrarse el "tag" del documento se puede identificar la fecha del mismo y otra información útil dependiendo del tipo de informe.

Esta componente de acordeón es utilizada para todos los reportes clínicos que son obtenidos por el *endpoint* previamente mencionado en la vista "FormAndDataView" y en esta última se generan tantas componentes de acordeón como sean necesarias para desplegar la totalidad de la información. Dicha componente se visualiza a continuación con el "tag" anonimizado para las fechas:

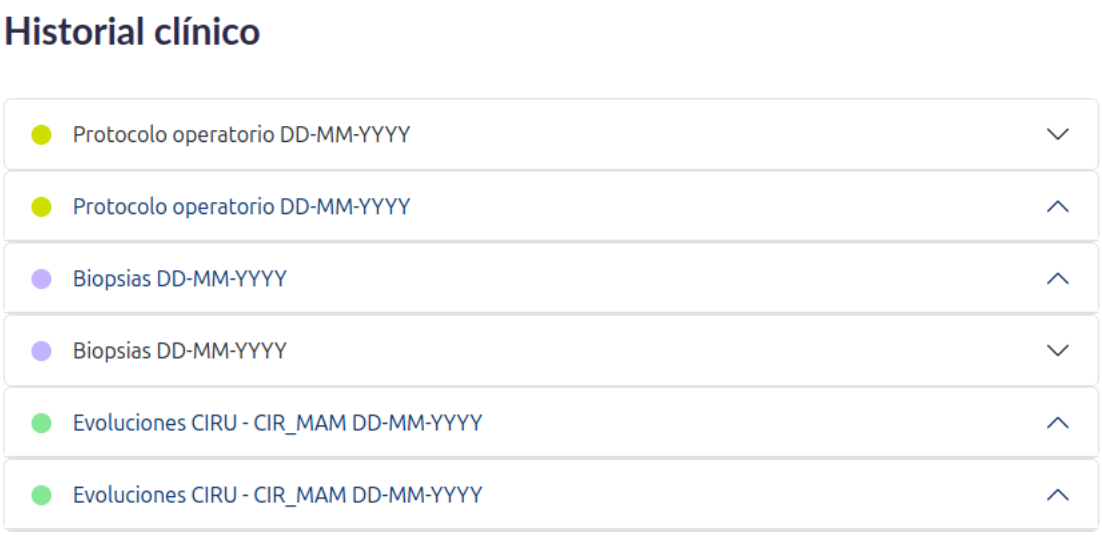

Figura 4.5: Interfaz de historial clínico con componentes de acordeón

Información del paciente

En tercer lugar, se muestra la información del paciente con datos como el sexo, fecha de nacimiento, estado vital, convenio y previsión. Además, si el estado vital es fallecido, se agrega el lugar y fecha de defunción. Estos datos se despliegan sobre todas las demás componentes de esta vista. De esta manera, se facilita el reconocimiento de estos datos de forma inicial previo al llenado de formularios de REDCap. En la primera fila, se encuentran los datos del paciente, mientras que en la segunda, los títulos de cada dato. La siguiente imagen muestra la componente, pero sin información de un paciente en particular:

Figura 4.6: Datos del paciente sin información personal

Filtros

En cuarto lugar, se implementaron los filtros de los archivos en una componente llamada "InfoFilters" a través de botones que representan cada tipo de informe que al presionar *click* sobre ellos, muestran o ocultan los documentos clínicos de cierta clase. Además se agregó un botón "Todos" que permite habilitar y deshabilitar todos a los documentos usando una propiedad calculada de Vue que depende del estado de los demás botones.

Al habilitar un tipo de informe se modifica un diccionario de valores booleanos donde las llaves son el nombre de la clase del texto. Este mapa es pasado desde la componente padre "FormAndDataView" hacia los hijos "EHR" e "InfoFilters", y recibe los cambios que son realizados en esta última componente a través de un evento emitido desde "InfoFilters" con el nombre del tipo del informe y el nuevo valor booleano. El diccionario es pasado a

la componente “EHR” que contiene los distintos informes clínicos desplegados a través de acordeones de Bootstrap para, finalmente, mostrar solo aquellos documentos tales que su tipo esté establecido en verdadero en el mapa. El conjunto de botones de filtrado es el siguiente:

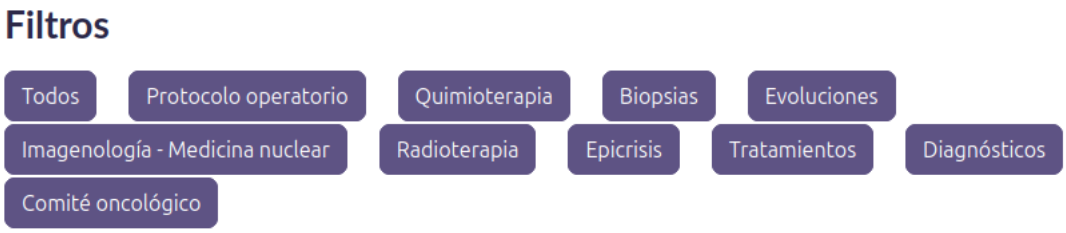

Figura 4.7: Filtros de informes clínicos

Búsqueda

En quinto lugar, se implementó la búsqueda de los informes médicos dentro de la componente “EHR” utilizando un elemento HTML de entrada de tipo búsqueda y una librería llamada “minisearch” que implementa una búsqueda rápida de palabras sobre documentos de texto que caben en la memoria local. Esto permite cumplir con el objetivo de tener una búsqueda en tiempo real de documentos, es decir, que a la par que se escribe en el buscador una letra, se actualicen los informes médicos que contienen la palabra descrita hasta ese momento.

Para utilizar esta herramienta, se confeccionó una función que se ejecuta una vez se obtienen los informes clínicos desde la base de datos de FALP. En esta función, primero, se configuran distintas instancias del buscador para cada tipo de informes, de forma que, las palabras al ser indexadas en el buscador sean normalizadas al igual que cuando son comparadas con el término buscado. Luego, a partir de cada documento se crea un diccionario específico para ser usado en el buscador, con un índice y distintos pares llave y valor, donde la clave es una columna de la tabla del informe y el valor es el dato de la fila. Finalmente se añaden los documentos de un cierto tipo a una instancia del buscador en específico para esos informes.

Los informes son buscados a partir de otro método que se ejecuta cada vez que se modifica la palabra ingresada y el largo de esta última es mayor que cero. Esta función, primero, normaliza la palabra establecida en la entrada y utiliza todas las instancias de buscadores para filtrar cada uno de los informes que contienen una o más palabras separadas por espacio.

Además del elemento *input* de tipo búsqueda de HTML, por un lado, se añadió una línea de texto que indica la cantidad de documentos que coinciden con la búsqueda de la palabra escrita. Y por otro lado, se agregó un botón que permite cambiar el orden de los documentos de orden descendente a ascendente, debido a que, existen tipos de informes que tienen una gran cantidad de documentos y a veces es mejor visualizarlos en un orden distinto al mostrado. Este cambio del orden se realiza a través de la función de ordenamiento “sort” de Javascript sobre los documentos según el índice entregado cuando se crearon los buscadores a través de los informes que ya se encontraban en orden descendente.

A continuación se muestra la interfaz del buscador luego de escribir la palabra “tumor”:

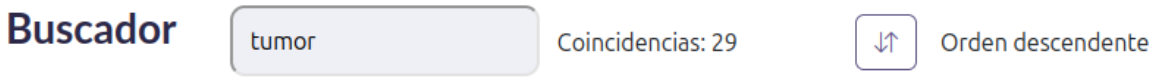

Figura 4.8: Buscador por palabra, coincidencias y ordenamiento

Resaltado de palabras buscadas y botón de copiar al seleccionar texto

Por último, luego de obtener retroalimentación de parte de usuarios investigadores de la vista de formulario e información médica, se incorporaron dos características que están enfocadas en facilitar aún más el proceso de estructuración de datos, que son, la implementación de resaltado de palabras al buscar uno o más términos en el buscador y un botón que se despliega al seleccionar texto en un informe médico y que permite copiar el fragmento.

Por un lado, para el destacado de palabras buscadas, se utilizó una librería denominada “vue-word-highlighter” que permite destacar una expresión buscada dentro de una componente de texto con distintas opciones ya incluidas, como por ejemplo, dividir la cadena de texto en espacios para buscar cada término, que en este caso, fue utilizada. El cuerpo de cada informe se encapsuló con esta componente para resaltar el texto buscado. A continuación se muestra un ejemplo del resaltado al escribir en la entrada “ASESOR genetico”:

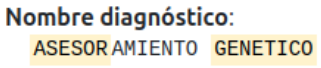

Figura 4.9: Resaltado de palabras

Por otro lado, para implementar el botón que copia una selección de texto en un informe clínico, se agregó un botón que únicamente aparece luego de realizar una selección de más de un carácter dentro del espacio de un documento. Para ello, se añadieron funciones oyentes a los eventos “mousedown” y “mouseup”, tal que, el método para la primera cierra el botón de copiar únicamente si se hace *click* fuera del mismo y está siendo mostrado, mientras que, la segunda, obtiene el texto seleccionado, y en caso de que el largo sea mayor a cero, calcula la posición, ancho y alto del botón, y muestra el elemento con un icono de copiar. Un ejemplo de uso de esta característica se puede ver en la siguiente imagen:

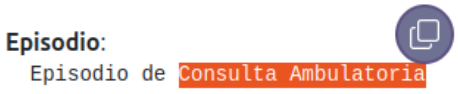

Figura 4.10: Botón al seleccionar texto

A continuación, se muestra la vista completa de formulario e información médica, ocultando los datos como fechas e información del paciente:

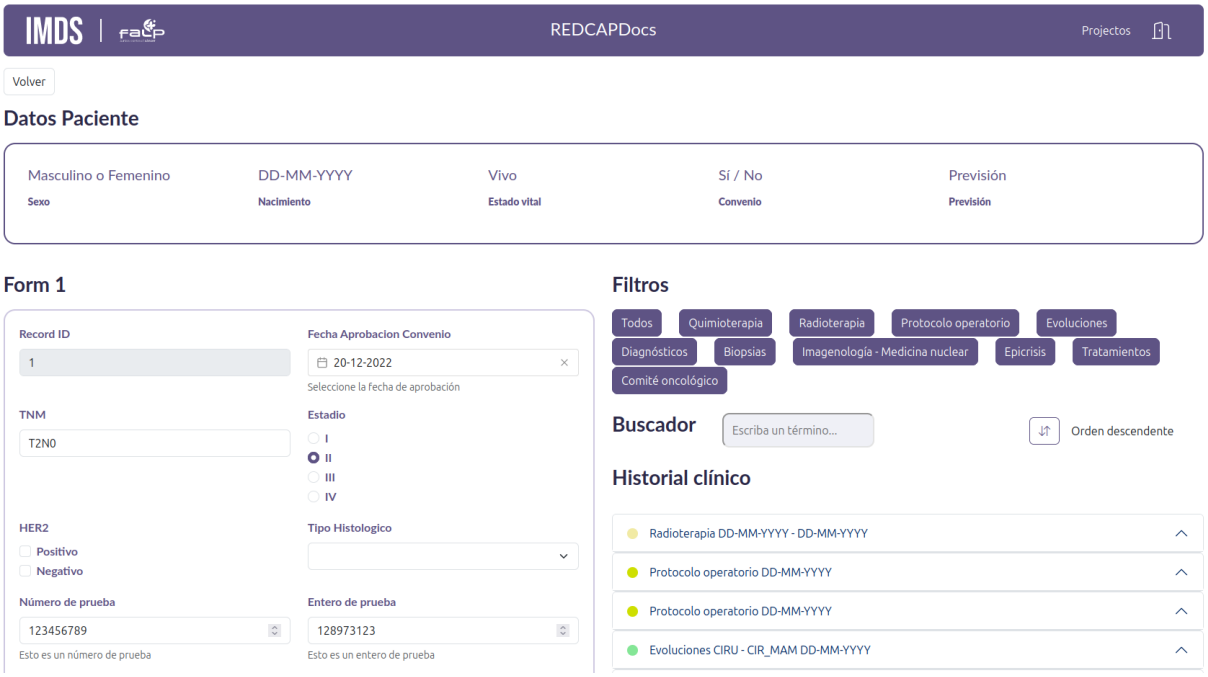

Figura 4.11: Vista de formulario e información clínica del paciente

4.7. Vista de un proyecto

La vista de un proyecto muestra y permite acceder a cada instancia de un formulario para cualquier registro existente, y agregar una instancia de repetición nueva en el caso de existir un instrumento repetible. Estas ocurrencias de formularios se encuentran resumidas en una tabla, en la cual, las filas representan los registros y las columnas los instrumentos a excepción de la primera columna, que representa el número de registro. En el caso de que el usuario sea administrador, también se añade una columna de RUT para identificar a los pacientes previo a entrar a una instancia de formulario. Adicionalmente, si el estudio es longitudinal, los instrumentos se agrupan por eventos en la misma tabla.

Al acceder a una ocurrencia de un formulario, se redirecciona el navegador del usuario a la vista de formulario e información médica cargando el instrumento elegido y los datos del paciente a través del número de registro de la instancia del formulario, que está asociado a un RUT en específico (que solo pueden ver los administradores).

Para confeccionar esta vista, en primer lugar, se buscaron todos los *endpoints* de la API de REDCap que son necesarios para poder mostrar todos los registros ordenados por instrumentos y eventos. Luego de realizar esta exploración, aquellos que sirven para este propósito y que fueron previamente analizados corresponden a **exportar metadatos**, **exportar los registros**, **exportación de instrumentos** y **exportar eventos**. Los *endpoints* que también son necesarios pero no han sido descritos previamente son los siguientes:

- **Exportar los instrumentos y eventos que se repiten**: Entrega una lista de diccionarios que poseen el nombre del instrumento, y si el proyecto es longitudinal, también

del evento al que pertenece la instancia del formulario.

- **Generar el siguiente nombre de registro**: Al llamar este *endpoint* se obtiene el número del siguiente registro a ser generado en el proyecto (y que aún no existe).
- **Exportar asignaciones de instrumentos y eventos**: Dada opcionalmente una o más ramas del proyecto, entrega una lista de diccionarios que representa una instancia de formulario que tiene el nombre del mismo, del evento asociado, y de la rama a la que pertenece.
- **Exportar info del proyecto**: Entrega un diccionario que posee información del proyecto y configuraciones establecidas del mismo, como por ejemplo, nombre del estudio, id, fecha y hora de creación, si es longitudinal, entre otros.

En segundo lugar, para obtener los registros agrupados por instrumentos y eventos desde el *frontend*, se creó un *endpoint* que se describe a continuación:

- GET "projects/id" [IsAuthenticated] : Dado un número de proyecto "id", obtiene todos los registros para cada instancia de instrumento. Si el proyecto es longitudinal, entonces, se le agrega el nombre del evento al que pertenece. En caso de que el usuario sea administrador, se agrega adicionalmente una lista de RUTs para cada uno de los registros.

Para la implementación del *endpoint* descrito, se crearon dos funciones principales que utilizan la API de REDCap para obtener la información necesaria. La primera de ellas, dependiendo de si el proyecto es longitudinal (conocido al usar el punto final exportar info del proyecto), agrupa los registros según instrumentos, o bien, a través de las asignaciones de instrumentos y eventos. Para realizar cualquiera de los dos procesos mencionados, se llama una única vez al *endpoint* de REDCap "exportar los registros" para evitar superar el límite de peticiones del servidor.

Dentro de esta función principal, si bien, se obtiene toda la información de una sola vez, es necesario realizar un procesamiento adecuado para organizar los registros según cada instrumento y evento, ya que, un registro que es representado como diccionario, tiene como llaves a todos los instrumentos del proyecto; puede contener la información de más de un registro; y no aparece si no hay información asociada al mismo en REDCap, lo cual, es necesario de generar para poder llenar el registro en el *frontend*.

Para resolver los problemas anteriormente descritos, se confeccionaron dos métodos. El primero, obtiene a partir de un diccionario de REDCap cada uno de los registros contenidos en el mismo objeto, a través de buscar las llaves de los instrumentos mediante una expresión regular para acceder a ellos y comprobar que existen. Además, se agrega un registro extra en el caso de que se encuentre una instancia de repetición de un formulario para permitir agregar una nueva ocurrencia de repetición. El segundo, completa una lista de registros entregada como parámetro para un instrumento en particular, con diccionarios que representan un registro que no se encuentren dentro del arreglo. De esta forma, para cada instrumento, existe al menos una instancia de formulario de cada número de registro.

Luego, en la función principal, para cada diccionario de registro de REDCap del proyecto, se utiliza el primer método para separar el objeto en los distintos registros que representa guardándose en la lista del instrumento o del formulario y evento correspondiente en caso de ser longitudinal. Finalmente, se itera por cada una de las listas de diccionarios y se usa el segundo método para completar aquellos registros faltantes.

La segunda función principal, utiliza el primer método para obtener los registros agrupados, y que, junto a los puntos finales de instrumentos repetibles, instrumentos, número del siguiente registro, eventos y asociaciones de eventos e instrumentos en el caso de tenerse un proyecto longitudinal, procesa cada registro y lo convierte a un diccionario que contiene la información necesaria para ser mostrado en el *frontend*, como por ejemplo, el estado del registro y a qué formulario y evento pertenece.

Por ultimo, se creó una vista de Vue denominada "ProjectView" para mostrar los registros del proyecto y cada una de las instancias de instrumentos agrupadas por posiblemente eventos. Para ello, se confeccionó una tabla donde las filas son los registros y las columnas son los instrumentos. Adicionalmente, si el proyecto es longitudinal, se muestran los eventos sobre los instrumentos a los que pertenecen.

Respecto a los registros de la tabla, cada instancia de llenado de un instrumento en particular se muestra como un botón circular con un color rojo, amarillo o verde dependiendo del estado del formulario que puede ser incompleto, no verificado y completado respectivamente. Este esquema de colores es semejante al utilizado en la plataforma REDCap, permitiendo así, tener una similitud a la "consola de registros" existente en la aplicación.

En el caso de que un instrumento sea repetible, en una casilla donde usualmente se encuentra una única instancia de llenado de formulario, pueden aparecer cero o más ocurrencias del mismo instrumento junto a un botón también circular con un icono de suma (+) que permite añadir una nueva instancia de llenado del formulario.

Cada uno de los botones de la tabla (incluido que añade una nueva instancia de repetición) tienen una ruta asociada que lleva a la vista de formulario e informes clínicos, en la cual, se muestra el mismo cuestionario asociado de REDCap y los datos del paciente que están asociados al registro de la fila de la tabla por un campo "rut" que poseen los proyectos. Para poder navegar hacia la vista, se utilizó la componente "router-link" que es provista por el enrutador de Vue. Se utilizó un objeto router-link para cada instancia de llenado de formulario, especificándose la dirección a seguir y distintos parámetros que permiten identificar el número de proyecto, el registro, detalles del instrumento y del evento si corresponde.

Para el caso en que la cantidad de registros sea muy grande, se añadió paginación, de forma que, se dividen los registros en listas (páginas) de un máximo de tamaño dado por el usuario y los elementos mostrados en la tabla corresponden solo a uno de estos arreglos que corresponden a una parte del total de registros. Esto último se realizó creando una propiedad computada de Vue a partir de la lista completa de registros, tomando en consideración la cantidad de elementos a ser mostrados en la tabla, que puede ser modificada mediante un menú de opciones ubicado sobre la esquina superior izquierda de la tabla.

A fin de permitir la elección de una página en la tabla, se implementó la componente de Bootstrap de paginación, que posee botones para ir hacia la página anterior, siguiente y

hasta 5 páginas distintas. Dependiendo de la lista de registros seleccionada, si la cantidad de páginas hacia la izquierda o a la derecha es de más de tres, se agrega un botón de tres puntos "..." en reemplazo de las páginas que deberían estar entre la primera y la anterior a la actual, o bien, entre la posterior a la actual y la última página respectivamente. El botón de tres puntos simboliza una o más listas de registros. Un ejemplo de paginación con varios registros tanto a la izquierda como a la derecha de la lista de registros actual es el siguiente:

Figura 4.12: Paginación con dos botones de tres puntos

A continuación, se muestra la vista de un proyecto longitudinal, donde se identifica el título del proyecto, el menú de filas a mostrar, los botones de paginación y una tabla de registros en la que se alcanza a visualizar cinco registros, cuatro eventos y trece instrumentos, de los cuales, cuatro son repetibles:

Figura 4.13: Vista de un proyecto longitudinal

4.8. Vista de proyectos

La vista de proyectos posibilita el acceso a distintos estudios de REDCap mostrando para cada uno, el nombre del mismo, el total de registros, el total de instrumentos y la fecha de inicio. Además, en el caso de ser administrador, permite añadir investigadores o administradores a un proyecto utilizando el nombre de usuario de la dirección de correo electrónico, y agregar un nuevo estudio mediante el *token* del proyecto.

A fin de confeccionar la interfaz, en primer lugar, se procedió a buscar los *endpoints* de REDCap que sean necesarios para obtener la información de los proyectos y posibilitar la creación de un nuevo estudio validando el *token* ingresado desde la plataforma. De esta exploración, todos ellos fueron utilizados previamente, los cuales son **generar el siguiente nombre de registro**, **exportación de instrumentos** y **exportar info del proyecto**.

En segundo lugar, para poder obtener la información del proyecto, asignar usuarios a estudios y crear proyectos, se confeccionaron los *endpoints* detallados a continuación:

- GET "projects/" [IsAuthenticated] : Obtiene todos los proyectos que posee el usuario que envió la petición y entrega un diccionario con el id del estudio, nombre del proyecto, el largo de los registros a través de usar el *endpoint* "generar el siguiente nombre de registro" y restarle uno al número obtenido, el total de instrumentos del estudio usando el punto final "exportación de instrumentos", y la fecha de inicio del proyecto.
- POST "projects/add" [IsAuthenticated, IsAdminUser]: Si un administrador llama a este punto final, dado un *token* de un proyecto, que es validado por su largo y por su existencia en REDCap al obtenerse su información a través del *endpoint* "exportar info del proyecto", crea una instancia del estudio en la base de datos asociando al usuario que envió la petición.
- GET "users/get_project_users" [IsAuthenticated, IsAdminUser]: Si un administrador llama este *endpoint*, dado el número de un proyecto, devuelve una lista con los nombres de usuario de todos los investigadores y administradores que tienen el estudio asignado.
- POST "users/assign_project" [IsAuthenticated, IsAdminUser]: Si un administrador llama a este punto final, dado el número de un proyecto y el nombre de un usuario a ser asignado, asocia la instancia de investigador o administrador al estudio.

Por último, respecto a la construcción de la interfaz de la vista de proyectos, por un lado, se creó una tabla que tiene como columnas el ID del proyecto, nombre, total de registros, total de instrumentos, fecha de inicio y una columna donde va un botón con icono de lápiz para acceder al proyecto. Cada fila de la tabla corresponde a una instancia de un estudio.

En el caso de que el usuario sea administrador, se añade a la tabla una columna en la cual por cada fila hay un botón con el icono de una persona para agregar usuarios en el proyecto. Además, se agrega un botón rectangular en la esquina superior derecha del cuadro para agregar un proyecto.

Por un lado, al hacer *click* en un botón para añadir un usuario al estudio, se abre una componente "modal" de Bootstrap con una entrada de texto para ingresar el nombre del investigador o administrador a añadir al proyecto, y una tabla con los nombres de los usuarios que ya han sido añadidos al estudio a excepción del administrador que abrió la interfaz. Un ejemplo del "modal" es la siguiente imagen:

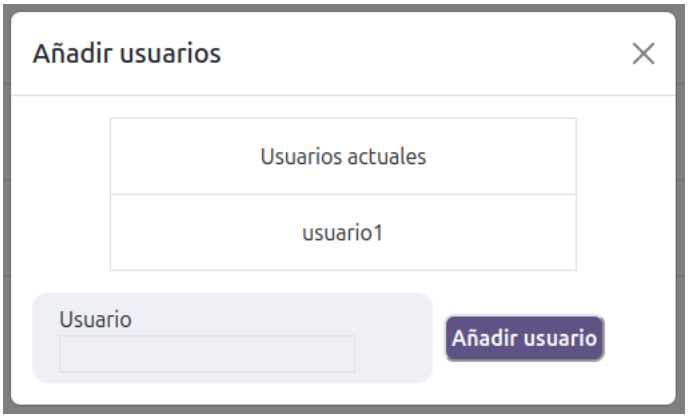

Figura 4.14: Cuadro para añadir usuarios con uno de ellos agregado

Por otro lado, al presionar el elemento que permite agregar un nuevo proyecto, se abre un "modal" similar al previamente descrito, pero que solo tiene una entrada de texto para ingresar el *token* del estudio y un botón para añadirlo finalmente. La componente descrita se visualiza a continuación:

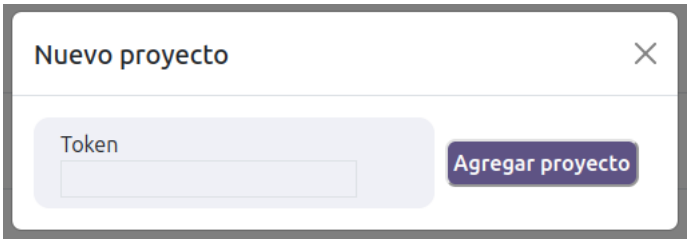

Figura 4.15: Componente para agregar un nuevo proyecto

La vista de proyectos para un usuario administrador que tiene tres estudios creados es la siguiente:

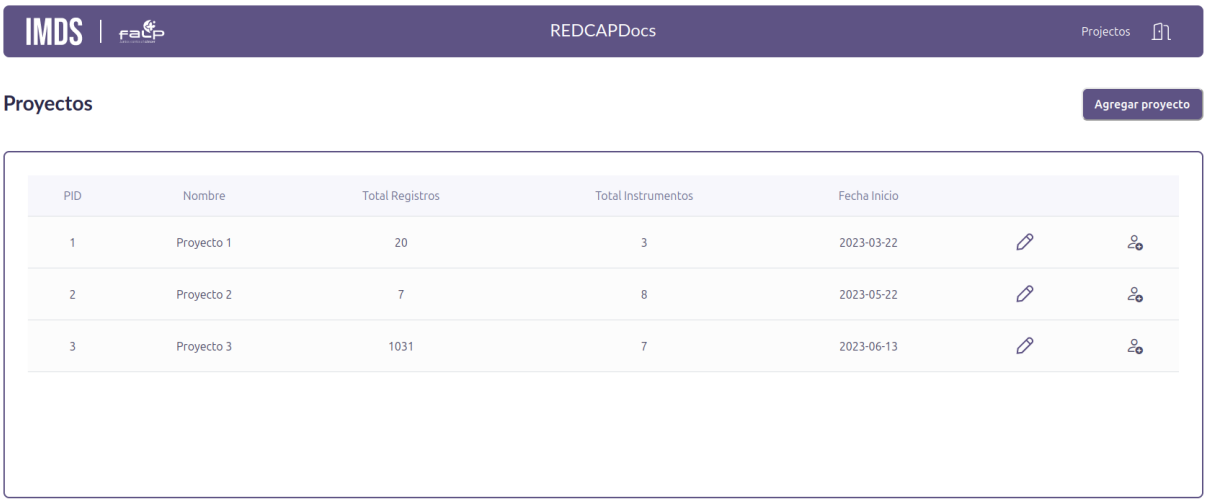

Figura 4.16: Vista de proyectos para un administrador

4.9. Vista de inicio de sesión

La vista de inicio de sesión permite el ingreso de los usuarios a la aplicación usando un único botón para iniciar sesión con una cuenta de dominio FALP de Google.

Respecto a la lógica de negocios, se realizó un único *endpoint* para el proceso de autenticación de usuario, que es el siguiente:

- POST "users/validate_auth": Dado un código de autorización de Google, primero, se realiza una validación de este código, y luego, se intercambia la clave de acceso con el servidor de autenticación de Google a cambio de *tokens* de refresco y acceso a la API de Google. Esto último se realiza utilizando las librerías "google-auth" y "google-auth-oauthlib" de Python usando una configuración establecida en el servidor de autenticación de Google. En el caso de obtener las claves de refresco y autorización, se crea u obtiene el usuario y se devuelve el código de refresco junto a una variable booleana que indica si el usuario creado u obtenido es de tipo administrador.

Para confeccionar la interfaz de inicio de sesión en el *frontend*, se creó la vista "LoginView" que despliega un fondo, el título de la aplicación "REDCAPDocs" y un botón para iniciar sesión con una cuenta Google de dominio FALP. Este último elemento corresponde a una componente de la librería "vue3-google-login", la cual se instancia en este caso con las propiedades "callback" y "popup-type" que sirven para establecer un método para manejar la respuesta de inicio de sesión y especificar que el tipo de autenticación es mediante un código de acceso.

Luego de obtener el código de acceso a causa de que el usuario se autenticó correctamente, este *token* se envía al *backend* para verificar si corresponde con el de un usuario de FALP y obtener a cambio una clave de refresco junto con un booleano que indica si el usuario es administrador de la plataforma. Finalmente, el *token* y la variable de administrador son guardadas en el almacenamiento local del navegador. La clave de acceso es utilizada para llamar a todos los *endpoints* de la aplicación a través de un método interceptor del paquete "axios" a excepción del usado para ingresar al sistema, mientras que, el booleano que indica si el usuario es administrador permite mostrar ciertas componentes e información dentro de la plataforma.

A continuación, se muestra la vista de inicio de sesión:

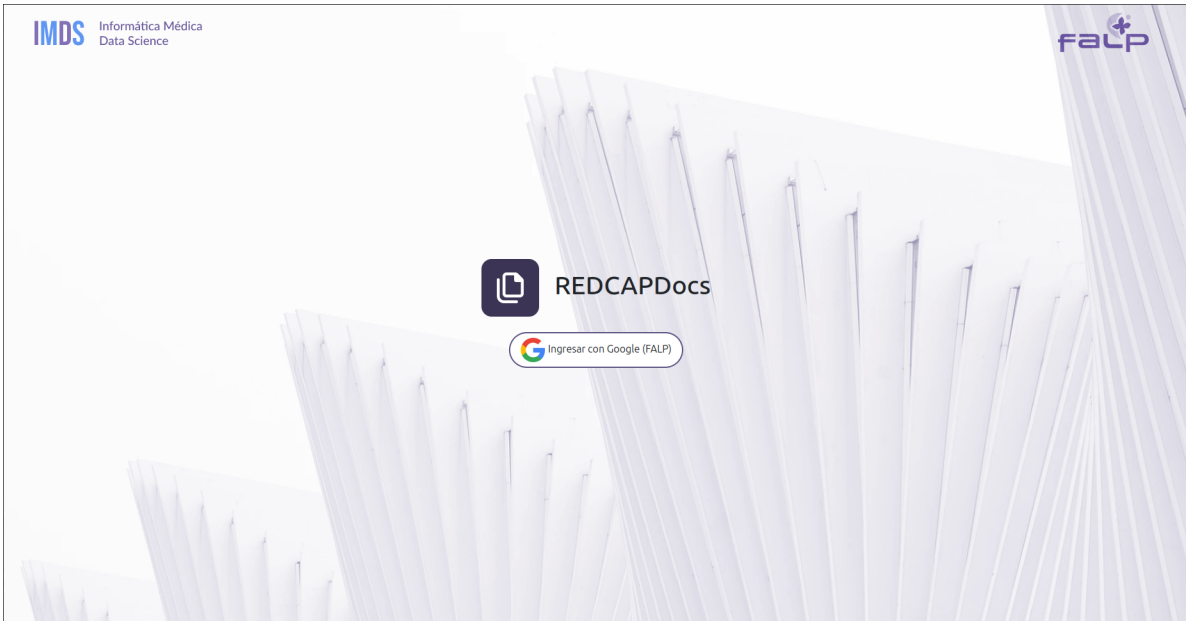

Figura 4.17: Interfaz inicio de sesión

4.10. Barra de navegación y botones de volver atrás

4.10.1. Barra de navegación

Luego de construir todas las interfaces anteriormente descritas, se creó una componente de navegación común a todas las vistas, a excepción de la de inicio de sesión. Esta componente permite ir hacia la vista de proyectos y desconectarse de la aplicación.

A fin de construir la interfaz en el *frontend*, en la vista principal de Vue "App" se creó una barra de navegación que posee el logo de IMDS y FALP a la izquierda, el nombre del sistema en el medio, y dos botones a la derecha que son para ir a la interfaz de proyectos y cerrar la sesión actual.

Para mostrar esta barra de navegación únicamente cuando no se muestra la vista de inicio

de sesión, se consulta una de las propiedades "meta" del enrutador de Vue que es un booleano que indica si se debe esconder la componente. Esta variable se define como falso en caso de no ser establecida, y, por tanto, en el archivo principal de rutas de Vue, cuando se declara la vista de inicio de sesión, se define esta variable como verdadera para esconder la barra de navegación en la interfaz.

Respecto a los botones de la derecha de la componente, por un lado, el elemento que permite ir a la vista de proyectos es una componente "RouterLink" que redirige a tal interfaz, mientras que, el botón para desconectarse elimina las variables guardadas en el almacenamiento local, elimina la conexión previamente establecida con la cuenta de Google y redirecciona el navegador a la vista de inicio de sesión.

La barra de navegación se puede visualizar en la parte superior de las figuras 4.11 4.13 y 4.16

4.10.2. Botones de volver atrás

Si bien, la barra de navegación es suficiente para entrar a todas las vistas de la aplicación, en muchos casos, un usuario requiere volver desde la interfaz formulario e información médica hacia la vista de un proyecto, o bien, desde la interfaz de un proyecto hacia la vista de proyectos. Debido a que la barra de navegación está más alejada de la interfaz y es más útil para los usuarios tener una forma de volver a la vista anterior cerca del área de trabajo, se decidió usar dos botones para volver a las interfaces previas.

Para la confección de ambos elementos, se utilizó un componente "router-link". Respecto al botón que va hacia la vista de un proyecto, se le entrega el nombre de la misma junto al número del estudio, mientras que, para el que va hacia la vista de proyectos, se le proporciona únicamente la dirección de la interfaz. Ambos elementos se encuentran debajo de la barra de navegación en la parte izquierda. El botón para volver a la vista de un proyecto se puede ver en la figura 4.11, mientras que, el elemento para ir hacia la interfaz de proyectos se muestra en la figura 4.13.

Capítulo 5

Evaluación

5.1. Marco evaluativo

Por un lado, para evaluar el trabajo respecto a la eficiencia del trabajo, se realizará una única prueba de usuario con un investigador y un proyecto que contiene 1000 pacientes. No se puede probar la aplicación con más de un investigador debido a la baja disponibilidad de examinadores para comprobar el sistema y tampoco se probará más de un proyecto, ya que, los estudios contienen cerca de 1000 pacientes o más y una cantidad variable de formularios, lo que requiere más de un día para ser completado.

La evaluación consistirá en realizar la estructuración de datos de un proyecto utilizando la plataforma REMI y la aplicación de REDCap, y luego, ejecutar el mismo procedimiento con el mismo estudio con el sistema "REDCAPDocs" junto a otras plataformas que pueden ser accedidas con REMI pero no con la aplicación creada en la presente memoria.

En ambas formas de estructuración de datos, se realizará el procedimiento en distintas sesiones en las que el investigador será responsable de medir el tiempo usando un cronómetro que iniciará al comenzar a revisar informes médicos y llenar formularios, y detendrá al pausar al finalizar la sesión. Luego de terminar el proceso de estructuración de datos, la suma del tiempo de todas las sesiones corresponderá al tiempo final de uno de los dos procedimientos.

Finalmente, se comparará el tiempo que le tomó al examinador estructurar los datos a través del uso del nuevo sistema, con el tiempo que le tomó procesar la información sin utilizar la plataforma construida, es decir, usando REDCap y REMI.
La eficiencia con el *software* implementado debería de aumentar en al menos un 50 % respecto a la cantidad de tiempo que le toma a los investigadores llenar los formularios de REDCap y revisar las fichas clínicas sin la aplicación. Es decir, que si T_{cp} es el tiempo que tomó el proceso con la plataforma y T_{sp} es el tiempo que tardó el investigador en realizar la estructuración de datos sin la plataforma, se debe cumplir que:

$$T_{cp} \leq T_{sp} \cdot 0{,}5 \tag{5.1}$$

Por otro lado, para comprobar si realmente el sistema es restringido, y que permita cumplir con las normativas legales y éticas vigentes, se debe realizar una demostración del flujo de la aplicación al cliente para que este confirme que la anonimización de los pacientes es efectiva y que se cumplirá con la normativa legal con el *software* creado.

Para ello, el cliente ingresará a la plataforma como usuario investigador para, luego, ingresar a la vista de proyecto de cada uno de sus estudios para verificar que no se muestre el RUT de ninguno de los pacientes existentes. Finalmente, ingresará a todos los instrumentos de un único registro para un proyecto no longitudinal y un estudio longitudinal, para verificar que no se muestre en ningún momento el RUT de un paciente dentro de la vista de formulario e información médica.

5.2. Resultados

Por una parte, respecto a la restricción del sistema, el cliente verificó que en el caso de los usuarios investigadores, estos últimos no pueden ver en ningún momento en la aplicación el RUT del paciente y tampoco tienen alguna forma de obtenerlo mediante la misma. Además, ningún usuario puede buscar los datos de un paciente por RUT a través del sistema debido a que en ninguna vista del *frontend* se maneja directamente el RUT. Por tanto, usando la aplicación ya no se corre riesgo de incurrir en problemas legales como sí ocurría con la plataforma REMI.

Por otra parte, en cuanto a la eficiencia del trabajo, esta no se pudo terminar de comprobar usando la aplicación, debido a que el servidor de REDCap de FALP bloqueó el acceso del servidor en el cual se desplegó la aplicación. Esto último sucedió debido a que el servidor de REDCap posee un límite de solicitudes por dirección IP que en un punto final del *backend* de la aplicación podía llegar a ser sobrepasado, ya que, se mandaba una petición por cada instrumento de un estudio para obtener los registros de un proyecto.

Esta limitante era desconocida por parte del autor de esta memoria, y no se corrigió anteriormente debido a que se probó la aplicación con usuarios investigadores de forma tardía. Si bien, aquel *endpoint* actualmente realiza una única petición, no se logró volver a desplegar el sistema en el servidor debido a la disponibilidad del personal de FALP. Por tanto, no se pudo evaluar la eficiencia del sistema implementado.

Capítulo 6

Conclusión

6.1. Trabajo realizado y reflexión

En el presente informe, se mostró el trabajo realizado para elaborar un sistema de captura y análisis de datos médicos no estructurados a través de una única plataforma que integra REDCap y una base de datos de informes clínicos. Para realizar la labor mencionada, se mantuvieron reuniones semanales con el cliente a fin de identificar el contexto, el problema y sus limitaciones, se generaron requisitos de usuario y de *software* a través de la solución planteada por el equipo de *data science* de FALP, se planteó una metodología de *software* a seguir, se diseñó la arquitectura y las interfaces de la aplicación, y finalmente se implementó el sistema tal como se describió previamente en este mismo documento.

Respecto a los objetivos, el general se resolvió de manera parcial, ya que, si bien, se logró desarrollar el sistema planteado, al no ser evaluada la aplicación, no se puede afirmar que el proceso de revisión de fichas clínicas y llenado de formularios es eficiente. En cuanto a los específicos, los primeros tres objetivos que son acerca del desarrollo propio de la aplicación fueron logrados, no así el cuarto, pues, a pesar de que se cumple correctamente la anonimización de los pacientes, no se pudo verificar la mejora de al menos un 50 % de la eficiencia del proceso de estructuración de datos.

La aplicación obtenida finalmente, si bien cumple con permitir procesar los datos no estructurados y anonimizar los datos de los pacientes para los usuarios investigadores, no pudo ser desplegada en producción debido al problema de limitación de peticiones al servidor de REDCap de FALP. Esto último limita considerablemente el impacto y relevancia de la aplicación realizada, ya que, no podrá ser utilizada por usuarios hasta que se vuelva a desplegar el sistema con los últimos cambios realizados para prevenir el límite de peticiones de REDCap.

Tras el desarrollo del proyecto, se destaca la importancia de acordar y realizar pruebas de manera temprana en la aplicación lo antes posible, para detectar problemas que incapaciten a la misma de poder ser utilizada, además de la retroalimentación que pueda existir sobre el diseño y funcionamiento de la aplicación de parte de los usuarios. De esta manera, se hubiese entregado una aplicación, quizás incompleta, pero funcional y que entregue valor a la organización.

6.2. Trabajo futuro

En primer lugar, como se mencionó en la primera sección del capítulo de concepción de la solución, el proceso de estructuración de datos en la aplicación es manual. La automatización de este procedimiento a través de algoritmos de aprendizaje de máquinas y procesamiento de lenguaje natural podría mejorar notablemente el tiempo que toman los investigadores en estructurar los datos a partir de los informes clínicos consolidados.

Una manera en la que se podría complementar el proceso actual de estructuración es obtener una lista de términos frecuentes identificados a partir de todos los informes de un paciente. Luego, de forma manual, se podría revisar dentro de los informes, de ser necesario, si la clasificación de los términos es correcta.

En segundo lugar, la aplicación desarrollada tanto en el *frontend* como *backend* carecen de pruebas del código realizado (conocido como *testing*). Estas últimas al ser implementadas otorgarían tanto robustez al sistema al detectar errores cuando se modifica el código, como documentación del proyecto, pues, se define el comportamiento del mismo con cada función de *testing*.

En tercer lugar, se deben agregar características que fueron planteadas por usuarios posteriormente al primer uso con la aplicación desplegada inicialmente, las cuales son:

- Agregar botones que redirijan a aplicaciones de desarrollo propio de FALP, como lo es Oncotext y Topomorfo
- Añadir nuevas fuentes de datos y permitir elegir entre estas últimas desde cuáles se extraen los informes clínicos de cada paciente
- Selección de variables enlazadas, es decir, que al elegir una variable de un formulario, se muestren únicamente los informes en los que puede aparecer el dato buscado

Estas implementaciones quedaron fuera del alcance de esta memoria debido al tiempo que tomaría agregarlas. Sin embargo, desarrollarlas podría impactar en la cohesión de otras aplicaciones existentes en FALP con el sistema creado y posiblemente acelerar aún más el proceso de estructuración de datos si se utiliza la aplicación construida.

Por último, es necesario realizar el despliegue de la aplicación en producción para finalmente probar el sistema con investigadores, y de esta forma, verificar que se haya cumplido con la mejora de eficiencia explicitada en el capítulo de evaluación.

Bibliografía

[1] Fundación Arturo López Perez, *Misión y visión FALP*, https://www.falp.org/fundacion/mision-vision-falp/, Accedido: 26-04-2023.

[2] Fundación Arturo López Perez, *Unidad de Investigación Epidemiológica y Clínica (UIEC)*, https://www.institutoncologicofalp.cl/fundacion/epidemiologia/, Accedido: 26-04-2023.

[3] REDCap, *REDCap*, https://www.project-redcap.org/, Accedido: 26-04-2023.

[4] NTT Data, *EhCOS by NTT Data. we're here to help*, https://www.ehcos.com/, Accedido: 28-04-2023.

[5] I. Müggenburg Rodríguez V. María Cristina Pérez Cabrera, "Tipos de estudio en el enfoque de investigación cuantitativa," *Enfermería Universitaria*, 2007, ISSN: 1665-7063. dirección: https://www.redalyc.org/articulo.oa?id=358741821004.

[6] Biblioteca del Congreso Nacional, *Ley 20584 — Regula los derechos y deberes que tienen las personas en relación con acciones vinculadas a su atención en salud*, https://www.bcn.cl/leychile/navegar?idNorma=1039348, Accedido: 28-04-2023.

[7] C. Zúñiga y J. Zuñiga-Hernández, "Excepciones al uso del consentimiento informado en investigación: ¿cuándo es esto posible en Chile?" *Revista médica de Chile*, vol. 147, págs. 1029-1035, ago. de 2019, ISSN: 0034-9887. dirección: http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0034-98872019000801029&nrm=iso.

[8] Nimbo, *La solución todo-en-uno para clínicas*, https://www.nimbo-x.com/, Accedido: 24-12-2023.

[9] Reservo, *Impulsa tu negocio junto a Reservo*, https://reservo.cl/cl/, Accedido: 24-12-2023.

[10] Nextgen, *An award-winning EHR system from a partner you can count on*, https://www.nextgen.com/, Accedido: 24-12-2023.

[11] OpenClinica, *Driving the future of clinical trials*, https://www.openclinica.com/, Accedido: 24-12-2023.

[12] Castor, *Finally, a modern clinical trials platform that is both powerful and easy to use*, https://www.castoredc.com/, Accedido: 24-12-2023.

[13] REDCap, *Clinical Data Interoperability Services (CDIS)*, https://projectredcap.org/software/cdis/, Accedido: 24-12-2023.

[14] A. Cheng, S. Duda, R. Taylor et al., "REDCap on FHIR: Clinical Data Interoperability Services," *Journal of Biomedical Informatics*, vol. 121, pág. 103 871, 2021, ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2021.103871. dirección: https://www.sciencedirect.com/science/article/pii/S1532046421002008.

[15] E. Gabriel Maida y J. Pacienzia, "Metodologías de desarrollo de software," Disponible en https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf. Accedido: 24-12-2023, Tesis de Licenciatura en Sistemas y Computación, Universidad Católica Argentina, Facultad de Química e Ingeniería "Fray Rogelio Bacon", 2015.

[16] K. Khakhkhar, *A Guide to Choosing the Right Software Development Methodologies for Your Next Project*, https://www.peerbits.com/blog/how-to-choose-software-development-methodology.html, Accedido: 24-12-2023.

[17] Melissa Alvarez Martell, Mariana Marcelino-Aranda, Alberto Macías Alciba y José Carlos Novoa Sandoval, *Metodología tradicional vs ágil para la gestión de proyectos de software*, https://www.boletin.upiita.ipn.mx/index.php/ciencia/925-cyt-numero-83/1901-metodologia-tradicional-vs-agil-para-la-gestion-de-proyectos-de-software, Accedido: 24-12-2023, Marzo de 2021.

[18] S. Laoyan, *Agile Manifesto: la guía para entender la metodología Agile*, https://asana.com/es/resources/agile-methodology, Accedido: 24-12-2023, Agosto de 2022.

[19] J. Martins, *¿Qué es la metodología Kanban y cómo funciona?* https://asana.com/es/resources/what-is-kanban, Accedido: 24-12-2023, Octubre de 2022.

[20] K. Beck, M. Beedle, A. van Bennekum et al., *Manifesto for Agile Software Development*, https://agilemanifesto.org/, Accedido: 24-12-2023, 2001.

[21] Wikipedia, *Waterfall model — Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=Waterfall%20model&oldid=1190305707, Accedido: 24-12-2023, 2023.

[22] D. Hughey, *Comparing Traditional Systems Analysis and Design with Agile Methodologies*, https://www.umsl.edu/~hugheyd/is6840/waterfall.html, Accedido: 24-12-2023, 2009.

[23] B. Lutkevich, *Waterfall model*, https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model, Accedido: 24-12-2023, Noviembre de 2022.

[24] Wikipedia, *Flask (web framework)*, https://en.wikipedia.org/wiki/Flask_(web_framework), Accedido: 24-12-2023, Noviembre de 2023.

[25] Django, *The web framework for perfectionists with deadlines.* https://www.djangoproject.com/, Accedido: 24-12-2023.

[26] DB-Engines, *DB-Engines Ranking - popularity ranking of database management systems*, https://db-engines.com/en/ranking, Accedido: 24-12-2023, Diciembre de 2023.

[27] A. Robledano, *Qué es MySQL: Características y ventajas*, https://openwebinars.net/blog/que-es-mysql/, Accedido: 24-12-2023, Septiembre de 2019.

[28] Wikipedia, *PostgreSQL*, https://es.wikipedia.org/wiki/PostgreSQL, Accedido: 24-12-2023, Noviembre de 2023.

[29] Amazon Web Services, *¿Cuál es la diferencia entre MySQL y PostgreSQL?* https://aws.amazon.com/es/compare/the-difference-between-mysql-vs-postgresql/, Accedido: 24-12-2023.

[30] Meta Open Source, *React*, https://es.react.dev/, Accedido: 24-12-2023.

[31] E. You, *Vue.js - The Progressive JavaScript Framework*, https://vuejs.org/, Accedido: 24-12-2023.

[32] Gunicorn, *Gunicorn - Python WSGI HTTP Server for UNIX*, https://gunicorn.org/, Accedido: 24-12-2023.

[33] Docker Inc., *Docker overview*, https://docs.docker.com/get-started/overview/, Accedido: 24-12-2023.

[34] P. A. Harris, R. Taylor, R. Thielke, J. Payne, N. Gonzalez y J. G. Conde, "Research electronic data capture (REDCap)—A metadata-driven methodology and workflow process for providing translational research informatics support," *Journal of Biomedical Informatics*, vol. 42, n.º 2, págs. 377-381, 2009, ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2008.08.010. dirección: https://www.sciencedirect.com/science/article/pii/S1532046408001226.

[35] UNMC - Research IT Office, *A Comprehensive Guide to REDCap*, https://www.unmc.edu/vcr/_documents/unmc_redcap_usage.pdf, Accedido: 24-12-2023, 2019.

[36] Red Hat, *REST API*, https://www.redhat.com/es/topics/api/what-is-a-rest-api, Accedido: 24-12-2023, Julio de 2023.

[37] University Information Technology Services, *Get started with the REDCap API*, https://kb.iu.edu/d/anxm, Accedido: 24-12-2023, Febrero de 2023.

[38] I. Li, J. Pan, J. Goldwasser et al., "Neural Natural Language Processing for unstructured data in electronic health records: A review," *Computer Science Review*, vol. 46, pág. 100 511, 2022, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2022.100511. dirección: https://www.sciencedirect.com/science/article/pii/S1574013722000454.

[39] C.-C. Chiu, C.-M. Wu, T.-N. Chien, L.-J. Kao, C. Li y C.-M. Chu, "Integrating Structured and Unstructured EHR Data for Predicting Mortality by Machine Learning and Latent Dirichlet Allocation Method," *International Journal of Environmental Research and Public Health*, vol. 20, n.º 5, pág. 4340, Febrero de 2023, ISSN: 1660-4601. DOI: 10.3390/ijerph20054340. dirección: http://dx.doi.org/10.3390/ijerph20054340.

[40] A. Wulff, M. Mast, M. Hassler, S. Montag, M. Marschollek y T. Jack, "Designing an openEHR-Based Pipeline for Extracting and Standardizing Unstructured Clinical Data Using Natural Language Processing," *Methods of Information in Medicine*, vol. 59, n.º S 02, e64-e78, Octubre de 2020, ISSN: 2511-705X. DOI: 10.1055/s-0040-1716403. dirección: http://dx.doi.org/10.1055/s-0040-1716403.

[41] H. Kharrazi, L. J. Anzaldi, L. Hernandez et al., "The Value of Unstructured Electronic Health Record Data in Geriatric Syndrome Case Identification," *Journal of the American Geriatrics Society*, vol. 66, n.º 8, págs. 1499-1507, Julio de 2018, ISSN: 1532-5415. DOI: 10.1111/jgs.15411. dirección: http://dx.doi.org/10.1111/jgs.15411.

[42] IBM, *¿Qué es el procesamiento del lenguaje natural (NLP)?* https://www.ibm.com/es-es/topics/natural-language-processing, Accedido: 24-12-2023.

[43] E. Ford, J. A. Carroll, H. E. Smith, D. Scott y J. A. Cassell, "Extracting information from the text of electronic medical records to improve case detection: a systematic review," *Journal of the American Medical Informatics Association*, vol. 23, n.º 5, págs. 1007-1015, feb. de 2016, ISSN: 1067-5027. DOI: 10.1093/jamia/ocv180. dirección: http://dx.doi.org/10.1093/jamia/ocv180.

[44] Ming, Frost, *Introduction - PDM*, https://pdm-project.org/latest/, Accedido: 24-12-2023, 2019.

[45] Encode OSS, *Home - Django REST framework*, https://www.django-rest-framework.org/, Accedido: 24-12-2023.

[46] NPM, *Build amazing things*, https://www.npmjs.com/, Accedido: 24-12-2023.

[47] Bootstrap, *Build fast, responsive sites with Bootstrap*, https://getbootstrap.com/, Accedido: 24-12-2023.

[48] Vite, *Why Vite*, https://vitejs.dev/guide/why.html, Accedido: 24-12-2023.

[49] Amazon Web Services, *¿Cuál es la diferencia entre el front end y back end en el desarrollo de aplicaciones?* https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/, Accedido: 24-12-2023.