UNIVERSIDAD NACIONAL DEL SUR

PROYECTO FINAL DE INGENIERÍA EN COMPUTACIÓN

TESIS DE LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN

Hacia la sistematización del Hospital Penna: sistema de órdenes de trabajo

Alumnos:

Franco Alesis Toniolo - Ingeniería en computación

Matías Morón - Licenciatura en ciencias de la computación

Directora:

Dr. María Laura Cobo

Departamento de Ciencias e Ingeniería de la Computación

Universidad Nacional del Sur – Bahía Blanca - Argentina

Diciembre del 2017

Índice de Contenido

Capítulo 1 - Introducción

El "Policlínico de Bahía Blanca" nació como una imperiosa necesidad de atender a los vecinos de la región que, en procura del alivio de su salud, llegaban al Hospital Municipal de Bahía Blanca, el cual a los pocos años de haber abiertos sus puertas en 1889 vio saturada sus modestas instalaciones. Es por esto, que el 28 de abril de 1928 se habilitó el Policlínico aunque la inauguración oficial se llevó a cabo el 27 de abril de 1930, con la participación de autoridades locales y provinciales. Este constó de veintiún cuerpos de edificios que sumaron más de trece mil metros cubiertos y costaron tres millones trescientos mil pesos de entonces.

En 1957 se lo designó con el nombre actual: Hospital Interzonal General de Agudos "Dr. José Penna", en memoria del médico sanitarista y epidemiólogo que alcanzó renombre mundial por sus investigaciones.

Durante la década del ´70, el crecimiento de Bahía Blanca, Punta Alta y otras ciudades de la zona inmediata hicieron pensar en una ampliación de aquellas instalaciones que ya evidenciaban signos del paso del tiempo. Acertadamente, se optó por construir un nuevo edificio y no modificar o ampliar las existentes.En el primer semestre del año 1973, se inician las obras de un gigantesco complejo sanitario, moderno y de alta complejidad. La inauguración oficial se realizó el 17 de agosto de 1984, con la presencia del ministro de Salud de la Nación, el gobernador provincial y otras altas autoridades.

Hasta el día de hoy se ha convertido en uno de los principales hospitales municipales de la región, sin embargo, como en toda entidad municipal sufre de una escasez de presupuesto que actualmente ronda 35.000.000 millones de pesos anuales destinados tanto para el personal como para la compra y reparación de los bienes, lo cual

es insuficiente como para destinar parte de este a la sistematizaciòn sino que se decide invertir en sus necesidades primordiales.

Dada la limitación presupuestaria que la entidad tiene, con nuestros recursos y ganas de aplicar todo el esfuerzo invertido en aprendizaje de nuevas tecnologías, framework, metodologías, etc. tratamos de devolver un poco todo el aporte que el hospital brinda a la comunidad. Es por esto que se sostuvo una reunión preliminar con los responsables del hospital con el objetivo de poder conocer el estado de sistematización actual en el que se encontraba. En la misma, pudimos observar que anteriormente mucha de la información, como lo era el manejo del personal, equipos existentes en las distintas áreas, y los proveedores externos con los que se trabajaba, eran mantenidos en un sistema de carácter privativo, que debido a su naturaleza, al perder su licencia fue imposibilitando su utilización, causando que con el paso del tiempo el registro comience a realizarse de forma manual, como ya se venían realizando en otras cuestiones como lo era el manejo del stock y las compras de insumos que se realizaban. Otro punto muy relevante en esta cuestión era la administración que se les daba a la órdenes de trabajo, que al igual que en casos mencionados con anterioridad, no se encontraba digitalizado. Esto forma un punto de inflexión importante, ya que como es sabido, el mantenimiento y reparación de la infraestructura y aparatología existente es una pieza fundamental para que el hospital pueda funcionar de forma correcta. Esto producía una gran dificultad para poder realizarles un correcto seguimiento, desde el momento que se creaba una nueva orden , hasta que alcanza su estado definitivo, que en su mayoría, no era el de finalización

Uno de los aspectos que generó mayor preocupación y que ellos creían que requerían una necesidad de cambio fue la dificultad de acceso y manejo de la información, ya que la toma decisiones depende del acceso a la misma, y en el caso del hospital, no se posee un registro organizado. Hay distintos causantes implicados, uno de ellos es el hecho

de no contar con un sistema que mantenga estos registros, y por otra parte por no tener un protocolo estandarizado para su organización, causando que cada área se maneje bajos sus propios criterios.

A partir de todas estas dificultades identificadas se procedió al comienzo de la resolución de las mismas diagramando el diseño de los sistemas que el hospital necesita para poder mitigar los causantes que fueron surgiendo en el proceso.

Unas de las necesidades iniciales con las que se cuenta es la de la realización de un sistema de administración del las entidades que el mismo posee, desde el manejo de la información del personal, áreas con las que éste cuenta, hasta la información referida a los equipos y prestaciones existentes. Todo esto formará una parte fundamental del proceso de sistematización, ya que las entidades son las que conforman las columna vertebral del hospital.

Otra de las inquietudes surgidas en el proceso, fue la necesidad de un sistema dedicado al manejo de órdenes de trabajo, que, como se mencionó con anterioridad, era una parte muy importante para el correcto funcionamiento del hospital. Desde el momento que se produce su creación, ya sea de forma automática por un mantenimiento preventivo o mediante la creación de un usuario del sistema, hasta que se produzca la finalización de la misma.

Será necesario además, un sistema propio para la administración de las órdenes de compras, donde se tenga un registro de los insumos que son requeridos, con información detallada de los mismos, los proveedores a los cuales las compras son realizadas, y una correcta actualización del stock del hospital. Esto además, será un pieza importante en el proceso de resolución de las órdenes de trabajo para poder llevar un registro de los insumos utilizados

Los sistemas mencionados, serán los que iniciarán el proceso, en el cual, el hospital Dr José Penna, de la ciudad de Bahía Blanca, emprenderá hacia una sistematización interna.

Luego de analizar el proceso de sistematización y ver la tamaño del mismo, se decidió acotar su realización de acuerdo a las necesidades primordiales con las que se contaba , como lo es el “sistema de órdenes de trabajo”. Sin embargo, para poder llevarla a cabo, se requirió el desarrollo de etapas adicionales que son necesarias para su utilización , como lo es , el sistema de administración de las entidades del hospital.

Organización de la Tesis

El desarrollo del presente trabajo estará dividido en otros tres capítulos, de los cuales el primero denominado “*Frameworks*” abordará el análisis de las diferentes alternativas entre los frameworks que se pueden utilizar para el desarrollo de los sistemas eligiendo uno para la implementación del mismo, continuando con el capítulo “*Decisiones de diseño del sistema”*, en el cual, como el nombre lo indica se mencionan las decisiones que se analizaron y tomaron para el diseño de los subsistemas que el hospital necesita desde el sistema de personal, pasando por el de órdenes de trabajo y terminando con el de compras. Luego en el capítulo “*Decisiones de implementación del sistema*” se abordaran los detalles de implementación que se tomaron tanto en el backend como en el frontend y se hará la presentación del sistema que se desarrolló y por último se introducirá un breve resumen de todo el trabajo realizado.

Capítulo 2 - Frameworks

Los frameworks para el desarrollo de software surgen de la necesidad de los programadores de estructurar y organizar el código, de manera de estandarizar el proceso para que todos los desarrolladores lo hagan de la misma manera, facilitando y agilizando la creación de nuevas piezas de software, además de proveer herramientas que simplifican tareas que se realizan cotidianamente.

Si bien no existe una definición formal para explicar que es un framework en el ámbito de la computación, podemos utilizar como referencia ciertas aproximaciones a ella. En el año 1994, Nelson Carl integrante de la organización IEEE Computer, desarrolló una aproximación a la definición del significado de un framework:

"Un Framework ayuda a los desarrolladores a proporcionar soluciones para el dominio de un problema, como así también a un mejor mantenimiento de estas soluciones. Brindan una estructura bien pensada y diseñada , de modo que cuando se crean nuevas piezas de software, estas puedan ser sustituidas con un mínimo impacto". Es decir , sirve de base para la organización y desarrollo de software.

En este capítulo se analizarán los diferentes frameworks que se pueden utilizar en el desarrollo del sistema, describiendo las características de cada uno y las ventajas y desventajas de los mismos. Luego, a partir de lo analizado y de las limitaciones que surgieron se procede a explicar cual se eligió y porque.

2.1- Evolución general de Frameworks:

Teniendo el claro concepto general de framework, haremos foco en los frameworks web, estos son básicamente frameworks orientados al desarrollo de aplicaciones web. Esto se debe a que el objetivo de este trabajo es desarrollar una aplicación web. Pueden encontrarse diversas clasificaciones y taxonomías, las más comunes, hacen énfasis en el lenguaje de programación que utilizan y en el tipo de arquitectura subyacente.

Al momento de realizar algún desarrollo web, resulta importante tener en cuenta las ventajas y desventajas que reportaría para el desarrollador el uso de algún framework. La evaluación de los frameworks es de suma importancia ya que su utilización no siempre será la mejor alternativa para el desarrollo de la solución buscada. El tamaño del proyecto y su grado de complejidad, el conocimiento de los desarrolladores acerca del framework a utilizar, y la facilidad para desenvolverse con ellos, pueden ser altamente determinantes para la elección.

El uso de frameworks tiene como ventajas más destacadas la facilidad para desarrollar código de manera rápida y sencilla. Esto se debe al soporte dado por medio de sus herramientas ya implementadas. Estas ayudan a resolver aquellas cuestiones que son básicas en el desarrollo de aplicaciones web, como por ejemplo las conexiones con las bases de datos y el enrutado de los enlaces webs, entre otros, evitando así,escribir código repetitivo pudiéndonos centrar en el desarrollo de la aplicación.

Además, es de importancia destacar que tanto las funcionalidades, como todas las características de estos frameworks, se encuentran bien documentadas y explicadas para que puedan ser utilizadas por los distintos desarrolladores. Existen grandes comunidades que continuamente ayudan a dar soporte a los mismos, y a su vez, implementan nuevas librerías y componentes para extender sus funcionalidades.

Si bien las ventajas son interesantes, también hay factores que podrían determinar que el uso de frameworks no resulta apropiado.

Puede destacarse que la curva de aprendizaje de algunos frameworks sea alta, para algunos desarrolladores puede resultar difícil entender y aprender las funcionalidades brindadas. Por ende, hasta que no se adquiera un conocimiento previo de su funcionamiento y estructura, no se llegará al desarrollo ágil de una pieza de software. Cabe destacar, que el proceso de aprendizaje es independiente del conocimiento del desarrollador en el lenguaje de programación, dado que la complejidad se centra en la estructura general del proyecto y el funcionamiento de cada uno de sus componentes. Por lo tanto, dependiendo del tamaño del proyecto puede resultar ineficiente el uso de un framework dado el tiempo que tomaría aprenderlo.

En cuanto a la elección del mismo, hay que ser conscientes que algunos frameworks se encuentran en desarrollo o cuyo potencial no es lo suficientemente amplio para abarcar los requerimientos del proyecto y muchas veces esta problemática puede no ser detectada en el análisis previo debido a una falta de conocimiento acerca del framework elegido o a un posible apresuramiento en la elección, resultando en un fracaso rotundo o en la necesidad de re implementar el proyecto con algún otro.

Como una de las mayores desventajas, existe la posibilidad de que el framework deje de proporcionar soporte y nuevas herramientas a los desarrolladores. En consecuencia, si se desea expandir el proyecto a futuro, puede que ciertos requerimientos del mismo no puedan ser implementados.

Teniendo en cuenta las ventajas y desventajas analizadas se espera que, en una primera etapa, se evalúe correctamente si es necesario utilizar un framework web para el desarrollo de la aplicación, la popularidad y las constantes actualizaciones que estos brindan a los desarrolladores, como así los numerosos requerimientos que el proyecto

posee. En caso de que esta evaluación no sea lo suficientemente completa, por lo explicado anteriormente, puede resultar perjudicial en el desarrollo, o en el peor caso, puede resultar en un fracaso.

2.2 - Clasificación de los Frameworks Webs:

Existen numerosas características sobre los frameworks webs, entre ellas podemos destacar el tipo de lenguaje que utilizan para el desarrollo, el patrón arquitectónico que toma como estructura principal, una posible división en módulos u orientación a objetos, la licencia con la que se encuentran patentados, entre otros.

Por lo tanto, analizando y teniendo en cuenta el contexto del proyecto a desarrollar, en el marco de este trabajo mostrará una clasificación basándose únicamente en el tipo del lenguaje y el patrón arquitectónico que el framework utiliza.

Teniendo en cuenta los lenguajes más utilizados actualmente y las necesidades de la aplicación, centraremos la atención en aquellos frameworks basados en los lenguajes PHP, JavaScript, Phyton.

Presentaremos las opciones más conocidas de cada uno:

Framework Python:

Son frameworks pensados para el desarrollo de grandes aplicaciones, los cuales se enfocan en la robustez y no tanto en la velocidad y en el consumo de recursos, por este motivo es que no se va a tener en cuenta para el desarrollo del presente proyecto.

Frameworks PHP:

Este tipo de framework utiliza el lenguaje PHP para el desarrollo de nuevas piezas de software. Por lo general, son utilizados en casos donde la sobrecarga es mayor del lado del servidor y no tanto del lado del cliente. Por ende, se responsabiliza al servidor de generar las vistas y todo tipo de acciones y tareas sobre el modelo y el controlador. Son recomendados en aplicaciones donde la cantidad de las tareas es excesiva para el cliente o no es necesario trabajar de forma dinámica en el mismo.

Dentro de esta categoría tres de los frameworks más utilizados en los últimos años. Son “Synfony”, “Cake PHP” y “Laravel”.

Symfony:

Es un framework completo, diseñado para optimizar el desarrollo de las aplicaciones web, gracias a sus características. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web por medio del patrón MVC. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada proyecto.

Symfony está desarrollado completamente en PHP y es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Es importante mencionar que es multiplataforma.

Posee numerosas características de las cuales podemos destacar, su facilidad de instalar y configurar en la mayoría de plataformas, ya que posee un sencillo mecanismo de instalación como lo puede ser “Composer”. Es independiente del sistema gestor de bases de datos, pudiendo elegir entre MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. El mismo brinda facilidades para poder ser utilizado tanto en pequeños como en grandes proyectos, dada la flexibilidad que otorga sus componentes.

Adopta la mayoría de las mejores prácticas y los mejores patrones de diseño para la web, como lo es el patrón MVC. Se encuentra diseñado para aplicaciones empresariales, con la posibilidad de poder ser adaptable a las políticas y arquitecturas propias de cada empresa.

Además, se caracteriza por ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo. Es fácil de extender, lo que permite su integración con librerías desarrolladas por terceros de una manera muy sencilla.

Resulta particularmente útil que siga el patrón MVC dado que es un patrón bien conocido, así como también el uso de Composer como gestor de dependencias lo cual facilita la instalación y la actualización de paquetes. Sin embargo, aunque sea un framework estable, está quedando en desuso por la aparición de otros nuevos frameworks.

CakePHP:

Es un framework libre, de código abierto, para el desarrollo rápido de aplicaciones. El objetivo principal de este framework, es permitir trabajar de forma estructurada y rápida, sin pérdida de flexibilidad. Pone a disposición todas las herramientas que se necesitan para poder desarrollar la lógica específica de la aplicación, haciendo uso del patrón MVC como estructura principal. Se encuentra patentado bajo licencias libres, siendo el caso de la "licencia MIT". Brinda una amplia compatibilidad, tanto con las versiones más actuales, como para las versiones más antiguas de PHP. Hace uso de la herramienta "CRUD" para realizar la interacción de la base de datos, otorgando una mayor facilidad para la comunicación entre ambos componentes. Otra característica que es importante mencionar es que posee mecanismos de generación automática de código, tanto para el modelo como para vista y el controlador, esto genera una mayor rapidez a la hora de desarrollar nuevas piezas de software. Cabe destacar que este framework, utiliza el patrón MVC como estructura principal del framework.

Al igual que con Symfony es de relevancia para el proyecto seguir el patrón MVC. Una cualidad que cabe destacar es el soporte con otras tecnologías como AJAX y JavaScript

Laravel:

Es un framework de código abierto para desarrollar aplicaciones y servicios web. Se encuentra basado en el lenguaje PHP, utilizando en las versiones más recientes PHP 5. Su objetivo es desarrollar aplicaciones de forma elegante y simple. También ofrece las

funcionalidades necesarias para que las aplicaciones desarrolladas sean lo más modernas y seguras. Está equipado con numerosas nuevas tecnologías, en las que se incluye enrutamiento RESTful, PHP nativo, eloquent ORM, entre otros.

Su estructura interna se encuentra basada en la arquitectura MVC, para su construcción se utilizaron varios componentes del framework Synfony. Por lo tanto, esto conlleva a que las aplicaciones web estén basadas en un código base confiable y optimizado, dado por los años de desarrollo que Synfony posee.

Entre algunas de las características de Laravel se incluyen, la de poseer un poderoso sistema de ruteo propio, con la posibilidad de poder ser alternado con el enrutamiento RESTful.

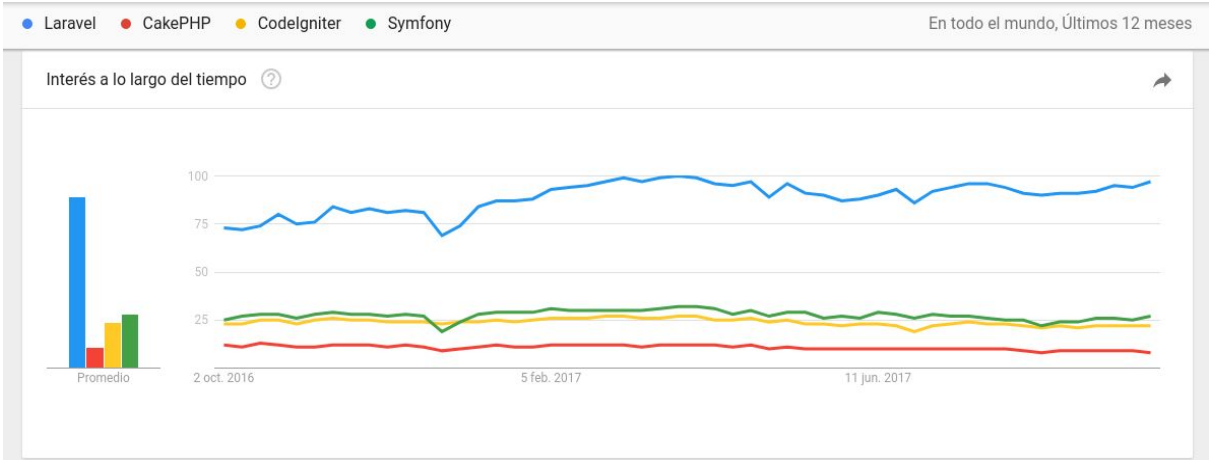
Para simplificar el desarrollo de plantillas, utiliza Blade como motor, brindando mayor facilidad para construir las vistas. Es importante mencionar, que posee una amplia y actualizada documentación, ayudada por el hecho de ser de Código abierto o *open source.* El mismo se encuentra en constante desarrollo, por lo que libera permanentemente nuevas actualizaciones y nuevas funcionalidad que aumenta considerablemente la potencia del mismo. Es por esta razón que en la actualidad llegó a ser el framework PHP más utilizado, por lo cual cuenta con una comunidad extremadamente activa y en continuo crecimiento.

Para la interacción con la base de datos se utiliza Eloquent ORM, otorgando a los desarrolladores la capacidad de evitar escribir extensas consultas SQL y optimizando la construcción de las mismas. Además, cuenta con un sistema de migraciones para gestionar la su estructura.

Cabe destacar que además posee una fácil y rápida instalación haciendo uso de la herramienta “Composer”.

Dentro de las características que posee, vemos interesante que esté pensado para ser una APi RESTful ya que creemos que un buen diseño sería separar la lógica del servidor con la cliente creando webservices. Además, dado que no está diseñada ni creada la base de datos es muy útil que cuente con un sistema de migraciones ya que va a estar en constante cambio su estructura. [1]

Teniendo en cuenta la información recolectada por la empresa Google, y mediante la utilización de la herramienta que proporciona “Google Trends”, podemos ver un análisis comparativo acerca del interés de los usuarios sobre los frameworks PHP utilizados en los últimos 5 años.



[2]

Como se puede apreciar en el gráfico, laravel es el framework más utilizado actualmente, con lo que conlleva, a tener una comunidad más activa brindando soluciones a los inconvenientes que nos pueden surgir.

Framework y librerías JavaScript:

Estos se basan el lenguaje JavaScript como lenguaje principal. A diferencia de los frameworks PHP la sobrecarga es mayor del lado del cliente que por sobre el servidor. De esta forma, se dividen las responsabilidades y las cargas que generan las tareas haciendo un mayor aprovechamiento de los dispositivos de los clientes.

Son muy utilizados en caso de que se requiera mucho dinamismo en los sistemas web que se generan, evitando excesivas peticiones al servidor por cada acción que lleva a cabo los usuarios. Es decir, algunas cuestiones se resuelven rápidamente del lado del cliente sin necesidad de hacer un pedido al servidor. Entre algunas de estas acciones podemos destacar rápidos cambios de porciones de vistas, pre-procesamientos de formularios, requerimientos AJAX, etc.

Dentro de esta categoría podemos encontrar 4 de los frameworks y librerías más utilizados en los últimos años. Ellos son “Jquery”, ”AngularJs”, ”Angular2” y “React”.

Jquery

Es una librería JavaScript de código abierto, que funciona en múltiples navegadores, y que es compatible con CSS3. Su objetivo principal es hacer la programación “scripting” mucho más fácil y rápida del lado del cliente además de su excelente integración con AJAX. Con jQuery se pueden producir páginas dinámicas así como animaciones parecidas a Flash en relativamente corto tiempo.

En su momento esta librería fue la más utilizada y aún sigue siendo usada pero en un número mucho menor dado que salieron al mercado otros frameworks, que se

mencionan posteriormente, que tienen una visión distinta de estructurarse y que son mucho más óptimos .

Angular JS:

Es un framework MVC de código abierto desarrollado por Google y escrito en Javascript, que trabaja del lado del cliente (client-side), permitiendo realizar de forma más dinámica nuestra aplicación web. Hace uso de otras tecnologías como HTML y CSS, así como librerías de terceros.

Este framework permite estructurar, organizar y escribir código de una manera más eficiente y en un tiempo menor, haciendo aplicaciones más rápidas de acuerdo a la manera en la que evolucionan los motores de render de los navegadores. Esto significa que, entre más poderoso y eficiente un navegador web, aplicaciones web más rápidas e intuitivas se necesitan. Trabaja con el patrón MVC (Modelo, Vista, Controlador), lo que permite separar correctamente la lógica, el modelo de datos y la vista en una aplicación web.

AngularJS permite realizar aplicaciones de tipo SPA (Single Page Applications), lo que significa que podemos construir una aplicación web en donde una parte de la misma se cargue dinámicamente, sin que se tenga que recargar todo la página. Esto permite hacer una aplicación web más rápida y fácil.

Es importante nombrar algunos de los objetivos de diseño que posee angular, como lo es el de disociar el lado del cliente de una aplicación del lado del servidor, esto permite que el trabajo de desarrollo avance en paralelo, y permite la reutilización de ambos lados. Además, permite disociar la manipulación del DOM de la lógica de la aplicación, esto mejora la capacidad de prueba del código. Otro punto, es el de considerar a las pruebas de la aplicación como iguales en importancia a la escritura de la aplicación, esto produce que la

dificultad de las pruebas se ve reducida drásticamente por la forma en que el código está estructurado.

Como consecuencia de estos objetivos, se guía a los desarrolladores a través de todo el proceso del desarrollo de una aplicación, es decir, desde el diseño de la interfaz de usuario a través de la escritura de la lógica del negocio, hasta los tests de pruebas.

Entre sus mayores ventajas, importar AngularJS a un proyecto existente es tan simple como hacer uso de una librería externa en formato ".js", ya que se encuentra preparado para adaptarse a cualquier ambiente de trabajo.

Sin embargo tiene una serie de desventajas que ha llevado a la decisión de construir un framework totalmente nuevo, entre las desventajas mencionadas se encuentran: Posee una sobrecarga en el navegador, dado que del lado del cliente se tiene mucha más programación que antes estaba del lado del servidor, para lo cual se puede sobrecargar el navegador haciendo que algunas aplicaciones sean lentas usando Angular 1 como motor. El mismo tiene un impacto negativo en la primera visita, ya que se tiene que descargar todo el código de la aplicación (todas las páginas, todas las vistas, todas las rutas, componentes, etc), que puede llegar a tener un peso de megas. Sin embargo, a partir de la segunda visita no es un problema, porque ya están descargados los scripts y cacheados en el navegador, pero para un visitante ocasional sí que representa un inconveniente grande porque nota que la aplicación tarda en cargar inicialmente. Otro punto, es el impacto negativo en el SEO, esto se produce ya que el renderizado se realiza del lado del cliente. El contenido se inyecta mediante Javascript y aunque se dice que Google ha empezado a tener en cuenta ese tipo de contenido, las posibilidades de posicionamiento de aplicaciones Angular 1 son mucho menores.

Angular 2:

Los problemas que se mencionaron anteriormente que son difíciles de solucionar con la tecnología usada por Angular 1, han sido los que han impulsado a sus creadores a desarrollar desde cero una nueva versión del framework.

Entre las nuevas características se puede mencionar el uso de typescript, este es un lenguaje que agrega las posibilidades de ES6 y el futuro ES7, además de un tipado estático y ayudas durante la escritura del código, el refactoring, etc. pero sin alejarte del propio Javascript (ya que el código de Javascript es código perfectamente válido en TypeScript).

Ahora el inyector de dependencias de Angular no necesita que estén en memoria todas las clases o código de todos los elementos que conforman una aplicación. En resumen, con Lazy SPA el framework puede funcionar sin conocer todo el código de la aplicación, ofreciendo la posibilidad de cargar más adelante aquellas piezas que no necesitan todavía.

Angular nació para hacer web y renderizar en HTML en el navegador, pero ahora el renderizado universal nos permite que no solo se pueda renderizar una vista a HTML. Gracias a ésto, alguien podría programar una aplicación y que el renderizado se haga, por ejemplo, en otro lenguaje nativo para un dispositivo dado.

Otra cosa que permite el renderizado universal es que se use el motor de renderizado de Angular del lado del servidor. Es una de las novedades más interesantes, ya que ahora se puede usar el framework para renderizar vistas del lado del servidor, permitiendo un mejor potencial de posicionamiento en buscadores de los contenidos de una aplicación. Esta misma novedad también permite reducir el impacto de la primera visita, ya que se tiene vistas "precocinadas" en el servidor, que se pueden enviar directamente al cliente.

Uno de los motivos del éxito de Angular 1 fue el data binding, pero éste tenía un coste en tiempo de procesamiento en el navegador, que si bien no penaliza el rendimiento en todas las aplicaciones sí era un problema en aquellas más complejas. El flujo de datos ahora está mucho más controlado y el desarrollador puede direccionar fácilmente, permitiendo optimizar las aplicaciones. El resultado es que en Angular 2 las aplicaciones pueden llegar a ser hasta 5 veces más rápidas.

La arquitectura de una aplicación Angular ahora se realiza mediante componentes. En este caso no se trata de una novedad de la versión 2, ya que en la versión de Angular 1.5 ya se introdujo el desarrollo basado en componentes. Sin embargo, la componentización no es algo opcional como en Angular 1.5, sino es una obligatoriedad. Los componentes son estancos, no se comunican con el padre a no ser que se haga explícitamente por medio de los mecanismos disponibles, etc. Todo esto genera aplicaciones más mantenibles, donde se encapsula mejor la funcionalidad y cuyo funcionamiento es más previsible. Ahora se evita el acceso universal a cualquier cosa desde cualquier parte del código, vía herencia o cosas como el "Root Scope", que permitía en versiones tempranas de Angular modificar cualquier cosa de la aplicación desde cualquier sitio.

De las características mencionadas es relevante para el proyecto el uso de componentes dado que nos brinda una mejor organización y reutilización de código permitiendo que futuros desarrolladores, que continúen con el proyecto, entiendan mejor la estructura y no pierdan tiempo desarrollando las mismas cosas. [3]

React

React es una librería Javascript de código abierto focalizada en el desarrollo de interfaces de usuario, creada por Facebook y utilizado por Instagram, Netflix y Airbnb entre otras.

Maneja un lenguaje llamado JSX, básicamente es una mezcla entre html, xml y javascript, el cual facilita mucho la programación debido a que tiene una sintaxis más corta y fácil de usar. Usar JSX no es obligatorio, pero si es muy recomendable a la hora de usar ésta librería.

Este pretende ayudar a los desarrolladores a construir aplicaciones de gran tamaño usando información que cambia a lo largo del tiempo. Su principal objetivo es en definitiva ser sencillo, declarativo e inequívoco (claro). El objetivo de React es renderizar el contenido del lado del servidor en la primera carga para dar la sensación de una SPA (Single Page Application) al usuario.

ReactJS es, en definitiva, la V del modelo MVC(Model-View-Controller) que tiene las siguientes características que le han llevado a ser una de las principales alternativas a las librerías más populares de JavaScript.

Sirve para desarrollar aplicaciones web de una manera más ordenada y con menos código que si se usa Javascript puro o librerías como jQuery centradas en la manipulación del DOM. Permite que las vistas se asocien con los datos, de modo no necesitamos escribir código para manipular la página cuando los datos cambian. Esta parte en librerías sencillas es muy laboriosa de conseguir y es algo que React hace automáticamente. Además nos permite una arquitectura de desarrollo más avanzada, con diversos beneficios como la encapsulación del código en componentes, que nos ofrecen una serie de ventajas más

importantes que los plugin, como la posibilidad de que estos componentes interactúen entre sí, algo que que sería muy difícil de conseguir con Plugins.

En comparación con Angular y Ember, React es una librería y no un framework puesto que react se ocupa de las interfaces de usuario. Esta posible carencia con respecto a los framework Javascript se soluciona con capas adicionales a React, como es el caso de Redux o Flux que aportan partes que React no se encarga. Éstos se ocupan del flujo de datos en React y lo resuelven de una manera optimizada, elegante, poniendo énfasis en la claridad de las aplicaciones. Lo que podría interpretarse como una desventaja, muchos desarrolladores lo entienden como una ventaja con respecto a frameworks completos ya que se puede aplicar aquellas herramientas y librerías adicionales que se adapten mejor a un proyecto en particular

La principal diferencia con otras librerías que se especializan en el "data-binding" es que React pone más inteligencia a la necesidad de actualizar una vista cuando es necesario y lo consigue mediante el "DOM Virtual". En líneas generales el virtual DOM es una representación del DOM pero en memoria. Cuando se actualiza una vista se actualiza el DOM Virtual, que es mucho más rápido que actualizar el DOM del navegador. Cuando React compara el DOM Virtual con el DOM del navegador sabe perfectamente qué partes de la página debe actualizar y se ahorra la necesidad de actualizar la vista entera. Es algo muy potente y es la base del rendimiento optimizado de React. Esto se hace de manera transparente para el desarrollador, que no necesita intervenir en nada.

Otra característica importante es que el flujo de información a través de los componentes de React es unidireccional. Cada componente pasa la información a sus componentes hijos y no al revés. Estos trabajarán con esos datos y cuando cambian su estado envían los eventos hacia los de orden superior para actualizarse. Esto hace fácil el seguimiento y razonamiento sobre cómo los datos se mueven a través de la aplicación.

Además renderiza HTML tanto en el servidor como en el cliente, rebajando la carga de trabajo necesaria para realizar aplicaciones web amigables para buscadores. El problema de las aplicaciones Javascript es que muchas veces reciben los datos en crudo del servidor, o un API, en formato JSON. Las librerías Javascript y frameworks toman esos datos para producir el HTML que debe representar el navegador. Esto, que es la solución más adecuada, porque nos permite desacoplar el servidor del cliente, pero se convierte en un aspecto negativo de cara al posicionamiento en buscadores como Google, debido a que el cuerpo de la página no tiene contenido.

Al no tener contenido una página que recibe los datos en un JSON, Google no sabe qué palabras clave son interesantes y no otorga ranking para ellas. Con ello la aplicación o página no consigue posicionar. Google está haciendo cambios y ha comenzado a procesar el Javascript para saber los datos de una página, pero aún dista de las ventajas que supone que el contenido esté en el propio HTML que entrega el servidor.

En resumen React permite isomorfismo, algo que le faltaba tradicionalmente a AngularJS 1.x. Ahora se ha resuelto en Angular 2.x. Aunque muchas librerías siguen sin ser capaces de soportar isomorfismo.

Por otra parte, al desarrollar en base a componentes permite que puedas usar el desarrollo de un proyecto en otro. Por lo cual se encuentra una gran comunidad que libera sus propios componentes para que cualquier persona los pueda usar en cualquier proyecto.

Una característica totalmente nueva es React Native y su enfoque Híbrido que es posible desarrollar una app con el rendimiento nativo pero con los beneficios del desarrollo web, es decir, su rendimiento es considerable como una aplicación nativa y simplifica la programación al utilizar una misma plataforma común. Al desarrollar una app en React Native el código que escribimos es el mismo para las versiones de Android e IOS. Sólo nos tenemos que preocupar de la lógica de negocio y de maquetar una única vez.

De las cualidades de React que vemos interesantes para el desarrollo del proyecto, además del uso de componentes, es que se ocupa únicamente de las interfaces y como tenemos pensado hacer un web service para todas las consultas al servidor, vemos oportuno su uso dado que sobrecargaría el sistemas con librerías que no se usan. [4]

Otra característica que nos llama la atención es el uso del DOM Virtual dado que actualiza los componentes únicamente cuando es necesario, haciendo mucho más fluida la aplicación. [5]

A partir de lo descripto y analizado de cada uno de los Framework Webs más utilizados, en la próxima sección del capítulo se explicará cuales se utilizaron y por qué.

2.4 - Elección del framework:

Para tomar la decisión de qué framework utilizar, además de haber analizado características y ventajas de cada uno se tuvo una serie de condicionamientos teniendo en cuenta la naturaleza del proyecto a desarrollar

Entre estos condicionamientos se pueden mencionar que el proyecto se presenta para un ente público por lo cual requería que la elección del framework a utilizar se basará en uno cuya licencia fuese libre y no de carácter privativo. Además, para una mayor extensibilidad y adaptabilidad tanto a desarrollos realizados y futuros, se requería que el framework elegido para la realización del back-end sea de la familia de los frameworks PHP.

Este lenguaje había sido seleccionado para desarrollos pasados y presentaba una facilidad en la mantención del mismo debido a no requerir presentar una capacitación

exhaustiva en un ambiente totalmente nuevo por parte del grupo de sistemas del hospital. Su elección debía contemplar que el mismo presente gran documentación existente al igual

que una considerable comunidad activa para asegurar una fácil extensibilidad del mismo debido a posteriores desarrollos y encontrarse en crecimiento activo para asegurar el potenciamiento del sistema a medida de que este vaya incorporando futuros requerimientos.

Era de suma importancia una compatibilidad con Mysql, debido a que es el sistema de gestión de bases de datos relacional utilizado por el hospital en sistemas previos.

A partir de lo analizado de cada uno de los frameworks y teniendo en cuenta las limitaciones que se tenía se decidió utilizar un framework para el “Front-end” y otro para el “Back-end”. A continuación se explicará cual y porque se utilizó cada uno separando su explicación.

Back-end:

Como mencionamos anteriormente, una de las grandes limitaciones era la utilización de un Framework web de la familia de PHP. Debido a la naturaleza del desarrollo de la aplicación y su necesidad en la división en etapas incrementales en el cual el grupo de desarrollo cambiará en el transcurso de las mismas, era necesario una herramienta estable, con una comunidad activa y en crecimiento constante, que brinde el soporte necesario para su fácil utilización. Otro cuestión a tener en cuenta, era la necesidad de la implementación de un Rest Api para abstraernos completamente del desarrollo Front-end , para lo cual Laravel nos brinda facilidades en dicha cuestión.

Luego de analizar las limitaciones y las necesidades que se requerían, se optó por utilizar Laravel como framework web.

Front-end:

Optamos por usar React gracias a la gran cantidad de ventajas que posee respecto a otras librerías y frameworks como Jquery,Angular,etc. y la enorme y muy activa comunidad que tiene, la cual hace que la librería está en constantemente crecimiento, dando múltiple soporte al mundo de desarrolladores.

Como mencionamos anteriormente entre las ventajas y una de las principales razones por las cual elegimos React es que permite actualizar los elementos que componen un interfaz actualizando el componente que ha cambiado su estado, en lugar de actualizar toda la página haciendo el sistema realmente ágil. Además el uso de componentes hace que el código sea mucho más limpio y fácil de mantener mejorando notablemente la reutilización.

Otra razón por la cual lo elegimos React y no Angular es que solo se encarga de la vista y al al trabajar con una API para obtener los datos , no es necesario un framework completo, como es el caso de este último,dado que realizar cualquier cambio no es tarea fácil ya que los componentes tienen una fuerte dependencia para que funcione correctamente.

También tiene la ventaja que si en un futuro se quiere realizar una app para móviles con React Native es mucho más sencillo y se puede reutilizar el código.

Resumen

En este capítulo presentamos las distintas alternativas de frameworks que se tuvieron en cuenta para el desarrollo de acuerdo a su popularidad. Luego, a partir de sus características y de las limitaciones que se tenían se llegó a la conclusión de utilizar Laravel como framework de backend y React para el frontend.

En el próximo capítulo se tratarán las decisiones de diseño que se tomaron para el desarrollo del sistema.

Capítulo 3 - Decisiones de diseño del sistema

Una orden de trabajo es un documento que una persona o grupo le entrega a una entidad y contiene una descripción pormenorizada del trabajo que se debe llevar a cabo.

Además de indicarse el lugar geográfico preciso y algunos datos personales de quien solicitó la realización del trabajo, luego de resolverse, se indica el tiempo que duró el trabajo realizado por parte del técnico, los materiales que se necesitaron para llevarlo a cabo, los costos producidos y cualquier otro tipo de contingencia que sea relevante de ser mencionada.

Es posible encontrarse con dos tipos de órdenes de trabajo, la correctiva, que nos informa especialmente sobre el problema a solucionar que fue oportunamente reportado.

En tanto, la preventiva es aquella que se emite de modo automático y que está vinculada con el mantenimiento preventivo que demandan algunos bienes.

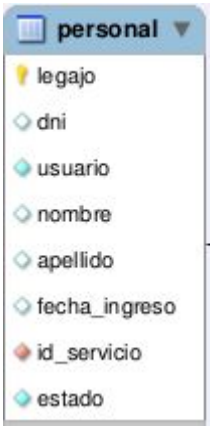
En este capítulo se explicarán las decisiones de diseño que se llevaron a cabo para poder realizar el sistema de **órdenes de trabajo**. Comenzando por la descripción de los subsistemas que requieren ser desarrollados previamente, como lo son el de **personal**, **técnicos** , **sectores** y **servicios**, así como también, con el manejo de los distintos tipos de bienes identificados, como lo son los **equipos** y **prestaciones**. Luego se mencionan las decisiones que se tomaron para desarrollar el manejo de **permisos**, y así, poder controlar el acceso de los usuarios a los distintos subsistemas mencionados. Posteriormente se expondrán las decisiones tomadas para la realización del sistema de órdenes de trabajo, exponiendo los distintos tipos de usuarios que serán implicados en el proceso de realización

de la orden. Finalmente, se explicarán las decisiones que se tuvieron en cuenta para la realización de los futuros subsistemas planificados, que como se comentó en el capítulo uno, no serán desarrollados en esta etapa inicial.

3.1 - Componentes previos:

Como se ha mencionado en ocasiones anteriores, el proyecto forma parte de una serie de proyectos a largo plazo, en el cual el desarrollo del mismo se realizará a partir de etapas incrementales. Es por esto, que la genericidad pasó a ser un punto de total necesidad en el diseño de la aplicación debido al carácter del mismo. Para lograr esto, React es esencial para su desarrollo. En el capítulo anterior fue descrita su arquitectura, la cual se diferencia de muchas otras ya que permite la encapsulación de código en componentes, posibilitando que estos interactúen fácilmente entre sí. Logrando esto, permite que el desarrollo de nuevas etapas en la aplicación favorezca en la integración con las ya constituidas. El el capítulo siguiente se hará mayor hincapié en esta forma de desarrollo, la cual fué uno de los ejes fundamentales en el diseño de la aplicación.

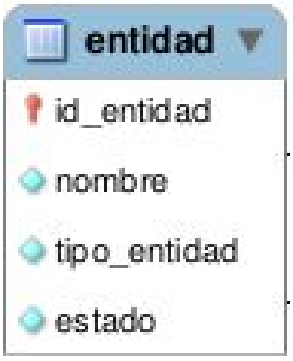
Como se hizo referencia en el primer capítulo, luego de analizar y ver el tamaño del proyecto solicitado, se decidió acotar su realización de acuerdo a las necesidades primordiales con las que se contaba, como lo es el “sistema de ordenes de trabajo”. Sin embargo, para poder llevarla a cabo, se requiere el diseño y desarrollo de etapas adicionales.



Dentro de estos subsistemas adicionales se encuentra el de personal, este es necesario para mantener un registro de todos los empleados con los que cuenta el hospital, además de poder registrar quienes realizaron los pedidos de mantenimiento de los distintos bienes. Como cuestiones relevantes se puede resaltar el uso del legajo para identificar unívocamente a cada empleado, además de un campo usuario que se utilizará para loguearse en el futuro sistema. En cuanto al id_servicio es el tenido en cuenta para mantener el registro del sector a la cual pertenece cada empleado dentro del hospital. Una de las decisiones tomadas para representar si se encuentra activo en el hospital, es la de llevar un campo representando su estado actual, esta medida es tomada en vez de eliminar el registro de una persona cuando ya no se encuentre activo, para permitir seguir manteniendo registro de todas las acciones relacionadas a dicho empleado durante su actividad en el hospital.

Otro agente relevante dentro del sistema son los técnicos, estos son los empleados que pertenecen a un área y tienen la capacidad de resolver las órdenes de trabajo.

Para revolverlo se optó por crear una tabla que relacione el personal con los sectores. Aquí los dos campos son llaves primarias dado que un empleado puede ser técnico de varias áreas. Otra decisión que se toma, es la de no llevar un estado como fue requerido en personal, sino que se elimine directamente la relación dado que no es perjudicial para el sistema perder esta información, ya que en todas las acciones que podrán realizarse, el campo registrado será el legajo. Las mismas irán describiéndose posteriormente.



Cabe destacar que un técnico solo podrá ver las órdenes asignadas a las áreas que pertenece, por este motivo, es necesario llevar registro de todos las áreas con las que cuenta el hospital. Como cuestión relevante se puede apreciar el campo tipo_entidad que se utiliza para diferenciar entre los distintos tipos existentes. Uno de ellos es el interno, utilizado para identificar las distintas áreas a las que los técnicos se encuentran asignados dentro del hospital, dentro de este grupo se encuentran Informática, Ingeniería y Electromedicina. Luego podemos identificar el tipo externo, utilizado para aquellas entidades externas al hospital encargadas de resolver las reparaciones en caso de que las áreas internas no puedan hacerlo. Por cuestiones de extensibilidad, el mismo fue diseñado con posibilidad de poder incorporar nuevos tipos de entidades a medida a que sea necesario, como lo serán en un futuro cercano, los proveedores, los cuales serán necesarios para el sistema de compras.

Como fue determinado en situaciones anteriores, se utilizará un estado para determinar si una entidad se encuentra activa o no.

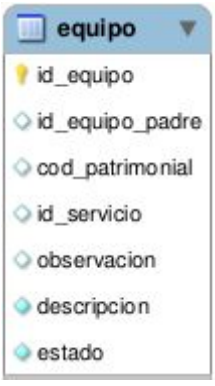
Otro dato que es importante almacenar son los sectores que componen el hospital, como son lo son por ejemplo Pediatría, Oftalmología, Traumatología, etc. y a las cuales pertenecen un empleado y los equipos. A estos sectores a partir de ahora los llamaremos servicios.

La misma será útil para poder conocer el servicio al cual pertenece un empleado o un equipo. Al igual que en el caso de los técnicos, no se mantendrá un estado dado que no es un elemento crítico dentro del sistema.

Estas son las entidades que se requieren, sin embargo también es necesario para el sistema mantener registro de todos los bienes con los que cuenta el hospital para poder identificarlos y así asignarlos a una orden de trabajo para luego tener información, como por ejemplo, de la cantidad de veces que estuvo en reparación, que partes se rompieron, si es necesario comprar uno nuevo en vez de arreglarlo,etc.

Dado que los bienes del hospital pueden tener caracteristicas muy distintas es que se los decidió separar en equipos y prestaciones.

Los equipos son los bienes tangibles con los que cuenta el hospital como por ejemplo gammágrafos, camillas, etc. Entre las características que posee se destaca el número patrimonial que es un código que identifica unívocamente a cada equipo, identificado en el sistema como cod_patrimonial. A su vez otra cualidad de estos bienes es que un equipo puede formar parte de otros equipos. El campo id_equipo_padre representa el equipo que lo contiene.



Las prestaciones como lo son por ejemplo una pared, una cañería de agua, etc., a diferencia de los equipos, no poseen un número patrimonial, por lo cual no se los puede identificar unívocamente. Tanto en los equipos como en las prestaciones se decidió que tengan un estado y no eliminarlos cuando los mismos dado de baja, ya que sino se perdería todo registro vinculado a ellos.

Otra necesidad primordial que se tiene, es limitar el acceso de cualquier persona al

sistema para no exponer información sensible como lo son los datos de los empleados, de los bienes que se disponen,etc. Para esto se decidió restringir el acceso de los usuarios mediante una autenticación al inicio utilizando un nombre de usuario y una contraseña, los cuales se le otorgarán al momento de dar el alta al personal, teniendo la contraseña un valor por defecto para facilitar la carga por parte del administrador, que luego la misma deberá ser modificada por el usuario.

Cada usuario va a estar asociado a un tipo de perfil, esto es necesario dado que los

mismos cuentan con distintas responsabilidades según su puesto, es por esto que se decidió crear un sistema de permisos para limitar el acceso dependiendo el perfil asignado. Contará inicialmente con tres tipos: Administrador, Técnico y Básico.

Cada perfil tiene asociado un conjunto de subsistemas a los que puede acceder, a esto lo llamaremos menú, dentro de cada uno hay acciones que lo componen, a la cual

denominaremos opciones, por lo tanto solamente va a poder acceder a aquellos menús y opciones que tiene habilitados.

Para darle más flexibilidad se decidió agregar un campo legajo, dado que en casos especiales se puede necesitar que un usuario acceda a ciertas acciones que no tiene asignada el perfil y que no se desea agregar al mismo ya que sino todos los usuarios con ese perfil dispondrán de ese permiso.

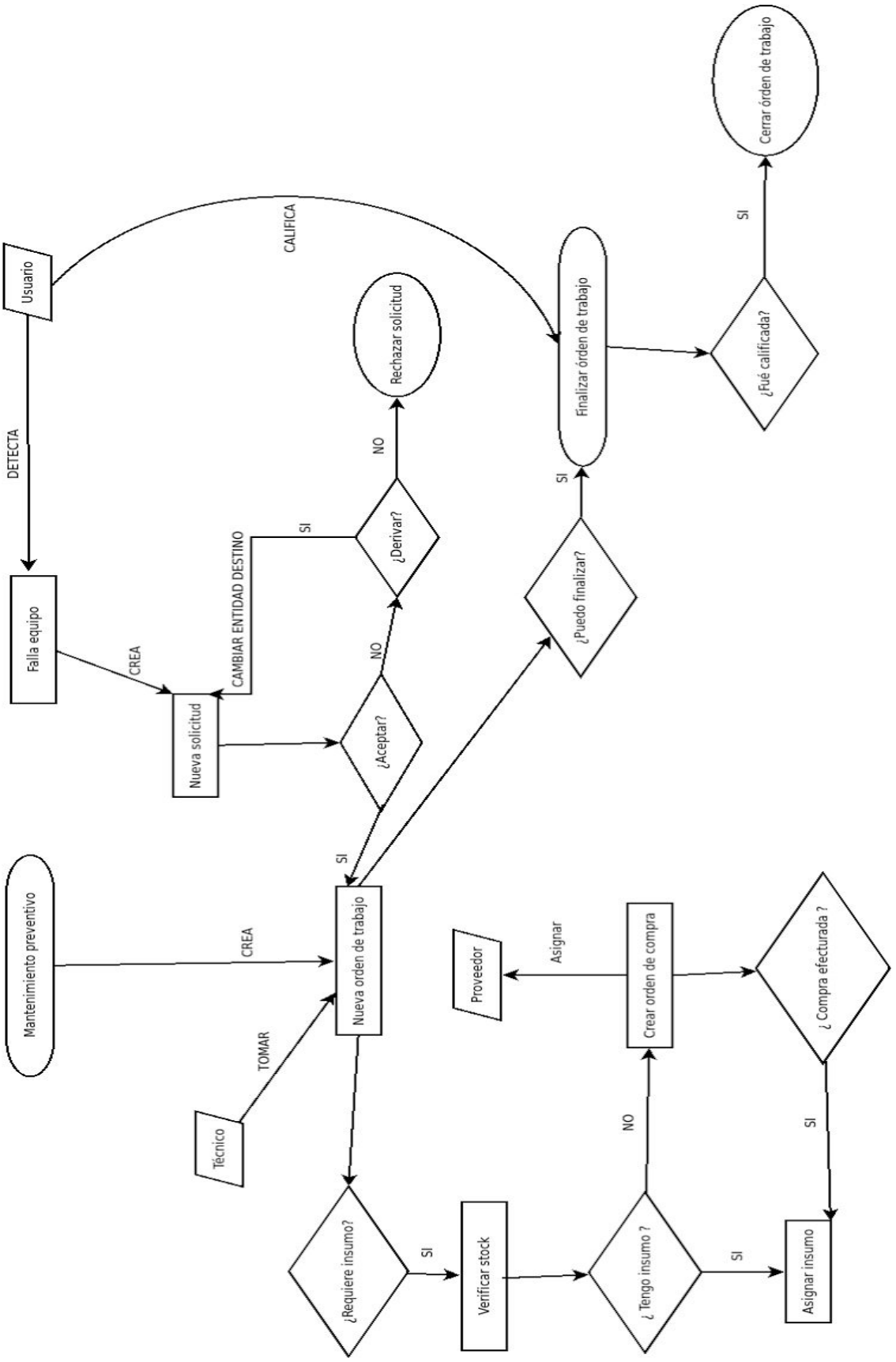
Estos son los subsistemas que son necesarios tener antes de empezar a diseñar el sistemas de órdenes de trabajo, el cual, en la próxima sección de capítulo se explicará las decisiones de diseño que se tomaron para llevar a cabo dicho sistema.

3.2 - Sistema de órdenes de trabajo:

Como fue mencionado anteriormente, el sistema de órdenes de trabajo, era una de las necesidades primordiales del hospital para poder ayudar en el correcto funcionamiento del mismo. En esta sección se pasará a detallar el proceso completo que implica el manejo de órdenes de trabajo.

Dentro del hospital, el proceso de una orden de trabajo no difiere demasiado del concepto brindado anteriormente . Sin embargo, como en toda organización, se tiene que adaptar a las necesidades propias del ambiente.

Por este motivo, se va a explicar el proceso cotidiano de una orden de trabajo dentro de la estructura propia del hospital mediante un diagrama de flujo.


Usuario
CALIFICA
Cerrar órden de trabajo
DETECTA
Rechazar solicitud
Finalizar órden de trabajo
¿Fué calificada?
SI
NO
Falla equipo
CREA
CAMBIAR ENTIDAD DESTINO
SI
¿Derivar?
NO
SI
Nueva solicitud
¿Aceptar?
¿Puedo finalizar?
SI
Mantenimiento preventivo
CREA
Nueva orden de trabajo
Proveedor
Asignar
Crear orden de compra
¿ Compra efecturada ?
TOMAR
Técnico
¿Requiere insumo?
SI
Verificar stock
¿ Tengo insumo ?
NO
SI
Asignar insumo
SI

Como podemos ver en el diagrama y de acuerdo a la definición dada, una orden de trabajo puede ser creada de distintas maneras según su tipo. Es decir, de forma preventiva, la cual se realiza automática y periódicamente mediante un técnico del hospital o de modo correctivo que es cuando, personal del hospital detecta una falla en un bien existente y procede a su creación, identificando dicho bien y asignándola a la entidad interna correspondiente de su reparación.

La entidad, al recibir una solicitud, analiza si le corresponde la tarea y de no ser así, un técnico tiene la capacidad de derivarla a otra. En caso contrario, la orden es tomada por uno de los técnicos para comenzar con el proceso de reparación.

Durante la reparación del mismo, se hace un reporte de los insumos requeridos, y en caso de no tenerlos en stock, se procederá a realizar una nueva orden de compra al proveedor correspondiente.

Al momento de terminar la reparación, para poder dar por finalizada la orden de trabajo, se requiere que el usuario que creó la solicitud de su conformidad del trabajo realizado.

Este es el circuito completo de la orden de trabajo en el hospital y es el que el futuro sistema intentará imitar, sin embargo es necesario, como primer medida, realizar un análisis de cómo conformar el diseño del mismo.

Como se mencionó anteriormente una orden de trabajo tiene como datos sobresalientes el bien que hay que reparar, la persona que crea la solicitud, la persona encargada de hacer el trabajo y una observación del problema encontrado.

Como existían casos en el que las órdenes de trabajo no eran solo para equipos, sino que también para ciertas prestaciones, fué que se decidió unificar estos con un id_bien, para luego tipificar los mismos con un tipo bien, el cual permitirá demarcar la diferencia para

poder identificar si corresponde a un equipo, o a una prestación, permitiendo además poder extenderlo a otro tipo, si así se requiere en un futuro.

A su vez, luego de crear la solicitud hay que asignarla a la entidad encargada de la reparación, es por esto, que se optó por poner el campo entidad_destino. Como forma de optimización, dado que en el hospital hay muchas órdenes de trabajo que se rechazan antes de comenzar a trabajar en ella, es que se decidió apartar la registración de todo el trabajo realizado por el técnico para agregarla solo cuando sea necesario. Para tales casos, se registrará la fecha de inicio y fin del trabajo indicando el tiempo dedicado en esa tarea. Además se incorporará un índice de prioridad de reparación dada por algún técnico, al finalizar con la labor se esperará por la conformidad del usuario creador de la orden para poder darla por cerrada. Otro campo que difiere al resto es el estado, aquí ya no es solamente alta/baja o activo/inactivo, sino que el mismo representará en qué parte del proceso se encuentra la orden de trabajo, es decir, al momento de ser

creada quedará en “Pendiente” hasta que un técnico comience a trabajar en ella, en el cual pasará a un estado “En Curso”, luego una vez concluido estará en “Finalizada” hasta que el usuario que creó la solicitud dé su conformidad y pase a estar “Cerrada”. Sin embargo la orden de trabajo puede ser cancelada por el usuario ya que cometió un error al crearla o rechazada por el técnico si no acepta el trabajo por diversos motivos, lo cual queda registrado con un estado "Cancelada por usuario" o "Rechazada por técnico".

Estas fueron las decisiones que se analizaron y tomaron antes de comenzar con la implementación del sistema de Órdenes de Trabajo, sin embargo , con las reuniones que se

tuvieron con los miembros del hospital y de acuerdo a sus necesidades, que no se llegan a satisfacer en el presente proyecto, se decidió dejar plasmado todo lo hablado en el diseño del sistema dejándolo abierto para futuras ampliaciones, es por esto que en la siguiente sección del capítulo se desarrollaran las decisiones de diseño de los subsistemas que quedaron afuera de la presente implementación.

3.3 - Subsistemas relacionados

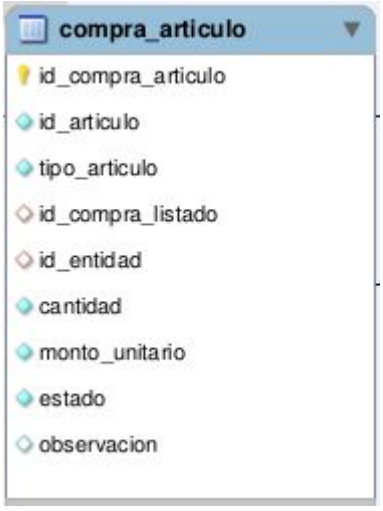
Como se comentó anteriormente, por cuestiones de alcance, ciertos subsistemas quedaron fuera del desarrollo de esta primer etapa. Debido a esto, era fundamental diseñar el sistema de forma que el mismo fuese fácilmente extensible al momento de la realización de las etapas que se irán incorporando.

Dentro de los subsistemas que quedaron fuera del alcance de la primera etapa, se encuentra el de **Compras**; ideado para mantener información de las todas las compras que se realizan en el hospital. El mismo fue inicialmente asociado a los equipos e insumos que serán requeridos para la refacción y mantenimiento de los mismos, sin embargo, fué diseñada de forma tal que pueda ser extensible para poderse asociar a cualquier otro artículo que requiera una compra.

Era necesario llevar un registro del stock de insumos con los que cuenta el hospital, en el cual es importante destacar de que cada uno tendrá definido una cantidad mínima con la que deberá contar, la cual

al momento de alcanzar un monto inferior al mínimo, se incorporará a una lista de insumos con necesidad de compra para realizar un restablecimiento del stock necesario.

Para llevar una auditoría de los insumos y equipos que se adquieran, es que se analizó llevar un registro de las compras que se llevan a cabo. Para esto será necesario poder diferenciar si la adquisición se trata de un insumo o de un equipo, para los cuales , por cuestión de encapsulamiento es que ambos serán registrados como “artículos”. Para poder realizar esto es que fué necesario contar con una tipificación al momento del registro, que permita distinguirlos posteriormente, la cual llamaremos tipo_articulo.



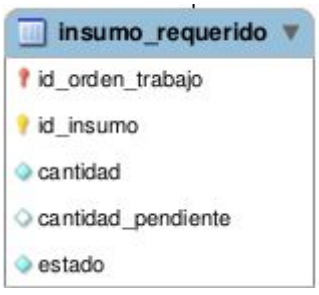
La creación de un nuevo registro de compras podrá ser realizado de distintas formas. Para el caso de los insumos, al momento de que cuente con un stock inferior al mínimo definido en su configuración, es que se creará de forma automática un registro de una nueva “compra de insumo”, que inicialmente contará con un estado “pendiente”, el cual irá cambiando al momento que la compra vaya tomando su curso. Cabe mencionar, que la creación de un registro de una compra de un insumo podrá realizarse igualmente de forma manual si esto es requerido. Para el caso particular de los equipos, la creación de una nueva compra será solamente de forma manual, al momento que sea necesario.

Es relevante destacar que además de un precio unitario y una cantidad a comprar, cada registro tendrá asociado un id_proveedor que será el que mantendrá la asociación al proveedor al que se le realizará la compra.

Al momento del diseño, se determinó la creación de un módulo encargado de la

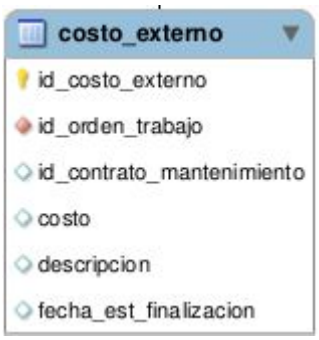
creación de un listado de artículos a comprar, el cuál asociará automáticamente mediante el proveedor seleccionado, todos los artículos pendientes de compra y además permitirá incorporar nuevos al listado, actualizando el proveedor al momento de la incorporación a la lista. El listado creado contará con un estado, que al igual que en el caso de cada artículo propio del listado, se irá actualizando al cambiar el curso de la compra.

Uno de los requerimientos surgidos de las necesidades planteadas por los miembros del hospital, fué la de poder registrar aquellos insumos utilizados en la reparación de un



equipo y poder asociarlos a la orden de trabajo previamente creada para dicha reparación. Es por esto que se diseñó llevar un registro de cada insumo utilizado en una orden, utilizando un id_orden_trabajo y un id_insumo, los cuales serán los encargados de asociar el insumo utilizado con la orden de trabajo creada. El mismo contará con un estado, que si el stock disponible del insumo alcanza a satisfacer la cantidad requerida, pasará a una estado finalizado, descontando automáticamente la cantidad del stock disponible, el cual en caso de no contar con stock suficiente permanecerá en un estado pendiente, hasta poder conseguirlo.

Durante el análisis de los requerimientos, surge el hecho de que ciertas órdenes de

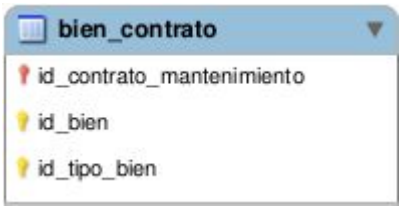


trabajo solicitadas no podían ser resueltas por técnicos internos del hospital, para lo cual, se requería tercerizar técnicos externos para su atención. Es aquí cuando surgió la necesidad de la registración

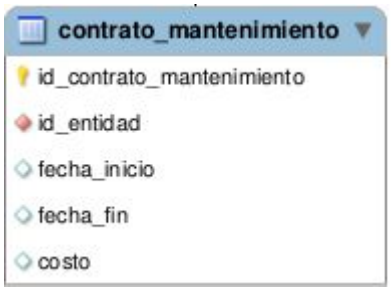
de datos adicionales para estos casos, que por la frecuencia con la que se dan, los llamaremos excepcionales. Es por eso, que para este tipo de casos se decidió adherir el registro del costo monetario que le llevará al hospital esta reparación.

Otro punto en cuestión, era que el hospital cuenta con contratos de mantenimiento con entidades externas para ciertos bienes, es por eso, que para cubrir los casos que así los fuera, se decidió llevar un registro del id_contrato_mantenimiento asociado a este bien, que será el que cubrirá la reparación, ya sea de forma total o parcial, según sea el caso.

Como fué mencionado anteriormente, ciertos bienes del hospital cuentan con contratos de mantenimiento, es por eso que se necesitó almacenar dicha información para poder emular la realidad. Para llevar esto a cabo, se requirió asociar el id_contrato_mantenimiento con el id_bien.



Para el manejo de contratos fué necesario registrar además de un periodo de duración y un costo, el id_entidad asociado al mismo, el cuál permitirá poder identificar a la entidad encargada de dicho contrato.



Resumen

Durante el desarrollo del capítulo se plasman las decisiones de diseño que se tomaron para poder sistematizar las órdenes de trabajo comenzando con los componentes previos que se necesitaban, para luego sí, centrarnos en las órdenes de trabajo

Además se tiene en cuenta el diseño de los subsistemas que implementarán en un futuro cercano. Todo esto pensado para facilitar el desarrollo de las etapas que vendrán.

En el próximo capítulo se desarrollarán las decisiones que se tomaron al momento de realizar la implementación del sistema, haciendo la presentación del mismo.

Capítulo 4 - Decisiones de implementación del sistema

Como se hizo mención en capítulos anteriores, se tomó la decisión de separar el desarrollo del backend con el frontend utilizando Laravel y React respectivamente con el objetivo de poder dividir la implementación de la interfaz del cliente con el servidor para lograr la mayor independencia posible entre las dos partes del sistema. Tanto Laravel cómo React tienen sus ventajas para poder desarrollar el rol que le asignamos. En el caso de Laravel presenta ciertas características que son beneficiosas para la implementación de una API RESTfull, como por ejemplo su propio manejo de ruteo, además de sus diversas interacciones con la base de datos, entre otras cosas. React, en cambio, está pensado para llevar a cabo el desarrollo de las interfaces del usuario con una separación en componentes que lo hace altamente extensible.

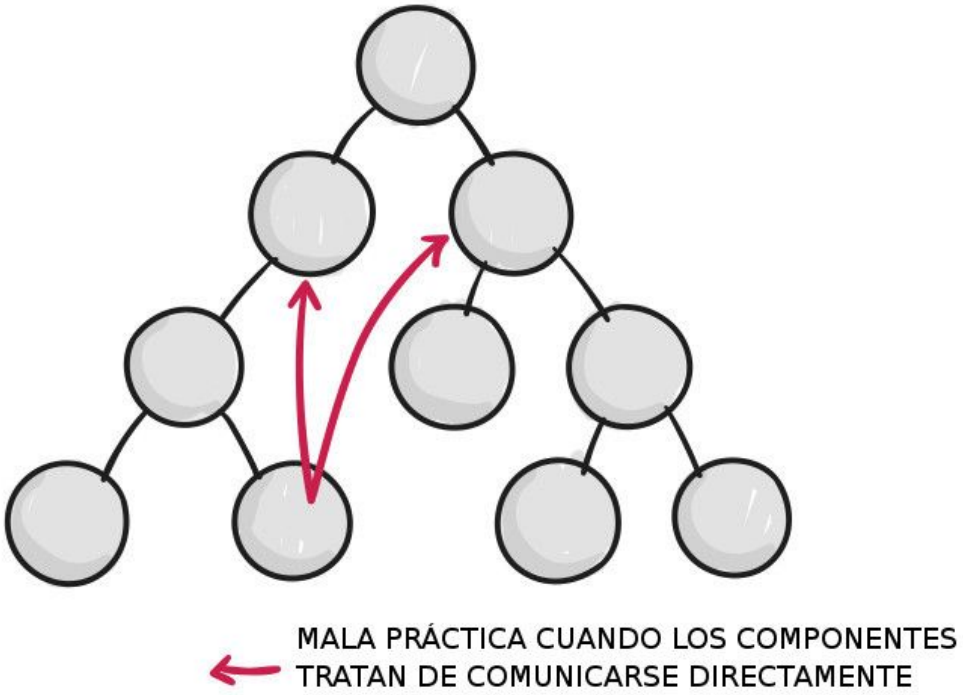
Durante el capítulo se explicarán las decisiones de implementación que se llevaron a cabo para poder realizar el sistema de **órdenes de trabajo**. Comenzando con la explicación de los componentes genéricos que se realizaron en React, como así también, la forma en que estos se comunican entre sí mediante el uso de Redux continuando por describir el motivo de implementar una API RESTfull además de la forma en que se realizan las consultas a la base de datos a través del framework Laravel. Luego se hará la presentación del sistema describiendo los distintos tipos de usuarios y la tarea que tienen que desarrollar cada uno de estos en el mismo.

4.1 - Decisiones de implementación - Frontend

Uno de los principales objetivos a los que se apuntaba en el diseño propio de la generación de código, era la de la modularización y genericidad de aquellos elementos propios de HTML que serían utilizados a lo largo de toda la aplicación. Este propósito propuesto, presentaba una gran disyuntiva, ya que la curva de desarrollo inicial sería demasiado alta. Sin embargo, por las características de la aplicación, se presentaba un fuerte condicionante que producía que esto fuese un requerimiento fundamental, la de el desarrollo en etapas, las cuales no solo iban a ser incrementales, sino que además serían realizadas por equipos de desarrollo distintos. No lograr este propósito, no sólo complicaría notablemente la extensibilidad del sistema, el cual, como fué mencionado, era necesario para poder culminar las etapas planificadas, así como también para la incorporación de aquellas nuevas que surjan con el avance del tiempo, y el aumento de las necesidades que se presenten en el hospital, sino también el de lograr unicidad a lo largo de los distintos subsistemas desarrollados, consiguiendo un equilibrio, el cual permita al usuario encontrarse con un sistema estándar, e intuitivo, para que la utilización e incorporación al uso de nuevas funcionalidades no solo sea fácil sino agradable.

Como pudimos observar anteriormente, la moderna arquitectura que nos ofrece React posee grandes ventajas para realizar una encapsulación de código en componentes, logrando una rápida extensibilidad del sistema para la incorporación de las nuevas etapas y funcionalidades planificadas. Esta característica propia de React, fué la que impulsó la decisión de realizar el desarrollo del sistema mediante la utilización de estos componentes, a los que catalogaremos de genéricos, vistas desde el entorno del desarrollo de la aplicación, que serán los que luego se utilizarán para la generación del código del sistema.

Una de las características de React es la forma de “pasar” datos a través de sus componentes, la cual se produce siempre de una sola manera, de los padres a los hijos, esto se denomina “flujo de datos unidireccional”. Como se puede entrever, esta característica no deja claro cómo lograr la comunicación entre dos componentes que no sean padre e hijo.

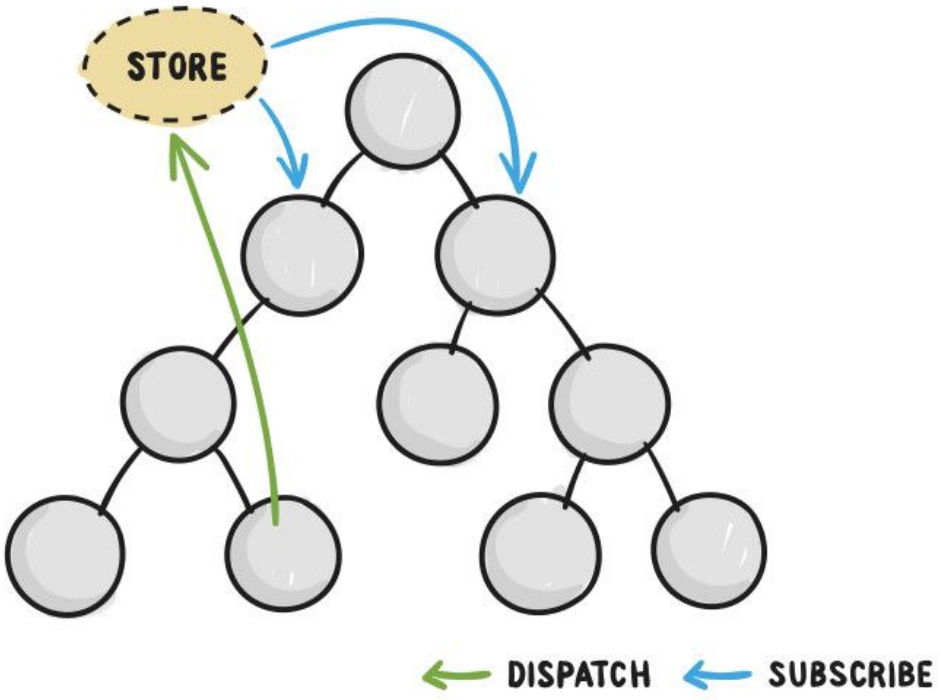


React no recomienda la comunicación directa componente a componente de esta manera. Incluso si tuviera características para respaldar este enfoque, muchos lo consideran una mala práctica porque la comunicación directa de componente a componente es propensa a errores y conduce al “código de spaguetti”, en inglés conocido como “spaghetti code”, un término antiguo para el código que es difícil de seguir.

React ofrece una sugerencia, pero esperan que la implemente uno mismo. Aquí hay una sección de la propia documentación de React :

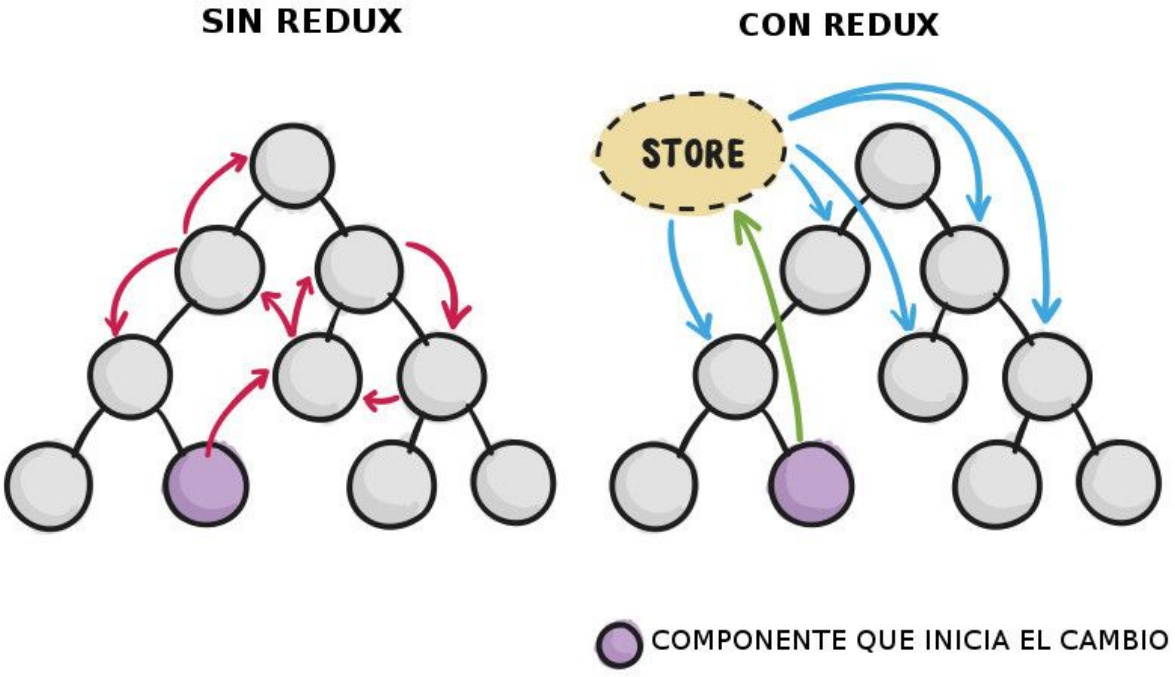
“Para la comunicación entre dos componentes que no tienen una relación padre-hijo, puede configurar su propio sistema de eventos global. ... El patrón de flujo (“ Flux pattern ”) es una de las formas posibles de organizar esto.”

Acá es donde Redux es útil. Redux ofrece una solución para almacenar todo el estado de su aplicación en un solo lugar, llamado "store". Los componentes luego "despachan" (dispatch) los cambios de estado al store, no directamente a otros componentes. Así mismo, los componentes que deben conocer los cambios de estado pueden "suscribirse" (subscribe) al store en cualquier momento.



Se puede pensar en el store como un "intermediario" para todos los cambios de estado en la aplicación. Con Redux involucrado, los componentes no se comunican directamente entre sí, sino que todos los cambios de estado deben pasar por la única fuente de verdad o como es conocido en inglés “single source of truth” , el store.

Esto es muy diferente de otras estrategias donde partes de la aplicación se comunican directamente entre sí. A veces, se argumenta que esas estrategias son propensas a errores y confusas para poder razonar con ellas. A continuación veremos cómo se comportaría el sistema con y sin la incorporación de esta estrategia de implementación.



Con Redux, está claro que todos los componentes obtienen su estado del store. También está claro a dónde deben enviar los componentes los cambios de estado, también

al store. El componente que inicia el cambio solo se ocupa de enviarlo al store y no tiene que preocuparse por una lista de otros componentes que necesiten el cambio de estado. Así es como Redux hace que el flujo de datos sea más fácil de razonar.

El concepto general de usar store (s) para coordinar el estado de la aplicación es un patrón conocido como patrón Flux. Es un patrón de diseño que complementa arquitecturas de flujo de datos unidireccionales como React.

A continuación pasaremos a describir las características más importantes que posee cada uno de los componentes “genéricos” que fueron implementados, las cuales nos darán un panorama más completo de lo que se intentará lograr para el desarrollo del sistema del hospital.

Una de las características en común en gran parte de los componentes, son los atributos que poseen para la validación: “validator” y “validation”, es decir la que se realiza para determinar si un elemento es requerido, deba ser de un tipo de dato específico, o posea un longitud mínima/máxima necesaria.

Es importante mencionar que cada componente es encargado de realizar su propia validación por lo cual esa característica quedó plasmada en la representación propia de los componentes que la requieran, como es el caso del Input, TextArea, SelectChosen, DatePicker.

Cabe destacar, que además fue necesario la incorporación de atributos adicionales en cada componente para poder generalizar su implementación, como lo son las *clases*, utilizadas para brindar estilos adicionales a los propios del componente. Así como también el *valor,* utilizado para mantener la representación del valor propio que tendrá dicho componente.

A continuación presentaremos la estructura de cada uno de los componentes mencionados.

```
<Input
        label           = {input-label}
        valor           = {input-value}
        clases          = {input-classes}
        validator       = {this.state.validator.input}
        validation      = {input-validation}
/>

<TextArea
        cols            = {column-number}
        rows            = {rows-number}
        valor           = {textarea-value}
        validator       = {this.state.validator.textarea}
        validation      = {textarea-validation}
/>

<SelectChosen
        label           = {select-label}
        llave           = {select-selectorKey}
        valor           = {select-value}
        descripcion     = {select-description}
        clases          = {select-classes}
        data            = {select-data}
        multi           = {singular/multi selection}
        validator       = {this.state.validator.select}
        validation      = {select-validation}
/>

<DatePicker
        label           = {DatePicker-label}
        valor           = {DatePicker-value}
        clases          = {DatePicker-classes}
        validator       = {DatePicker-validator}
        validation      = {DatePicker-validation}
/>
```

Además de los ya mencionados, es importante destacar otros, que a pesar de no poseer atributos particulares en su representación, serán generalizados para poder conseguir, como fue mencionado anteriormente, la unicidad a lo largo de los distintos subsistemas desarrollados. Los mismos son: Formulario, InputFake, Boton, GenericModal, Card.

```
<Formulario
        titulo          = {form-tittle}
        id              = {form-id}
        submit          = {function-on-submit}
>
{form-content}
/>


<InputFake
        label           = {label-input}
        value           = {input-value}
        clases          = {input-classes}
/>


<Boton
        label           = {button-label}
        icon            = {button-icon}
        clases          = {button-classes}
/>


<GenericModal
        show            = {visibility-status-modal}
        onHide          = {close-function}
        body            = {modal-body}
        accion          = {callback-confirm-function}
/>
```

```
<Card
        key             = {card-key-selector}
        url             = {card-url}
        nombre          = {card-name}
        logo            = {card-logo}
/>
```

En el caso particular de las tablas, estas forman una parte importante del sistema, por la gran muestra de información que es requerida. Para lo cual se planea utilizar una librería previamente desarrollada, llamada "*react-bootstrap-table"*. Por este motivo, se analizó el desarrollo de un nuevo componente que logre tanto encapsular su representación, como los atributos que serán agregados para poder cubrir los requerimientos previstos, y de esta forma poder abstraerse de ciertos detalles de la tabla y así facilitar su uso.

```
<BootstrapTable
        height          = {table-height}
        search          = {search-field-option}
        data            = {table-content}
        deleteRow       = {delete-row-option}
        selectRow       = {BsTable.selectFila}
        cellEdit        = {editar}
        options         = {table-configuration}
        hover
        striped
        pagination
        >
        {table-header-column-structure}
</BootstrapTable>
```

```
<TableHeaderColumn
        isKey                           = {key-option}
        dataField                       = {table-data-column}
        editable                        = {edit-column-option}
        hidden                          = {hidden-option}
        invalidEditColumnClassName = {classes-on-invalid-edition}
</TableHeaderColumn>
```

Estos componentes mencionados, son lo que serán requeridos para la implementación propia del frontend que poseerá el sistema, lo cuales permitirán alcanzar tanto la genericidad y extensibilidad buscada, como así también la estandarización de los distintos subsistemas que poseerá el hospital.

A continuación se pasará a describir las características que poseerá la implementación de la otra parte faltante de la aplicación, el “Backend”, así como también las decisiones que se llevaron a cabo.

4.2 - Decisiones de implementación - Backend

Como se hizo hincapié en la introducción del capítulo , una vez elegido Laravel como framework, el próximo paso es tomar la decisión de cómo estructurar la implementación del backend y cómo realizar la comunicación con el frontend.

Hasta hace poco tiempo, los protocolos de intercambio de datos en modo estándar ofrecían gran capacidad, pero también resultaban complejos a la hora de manejar. SOAP (Simple Object Access Protocol) era el perfecto ejemplo de ello. Sin embargo, el uso de una alternativa más sencilla como REST (Representational State Transfer o Transferencia de Estado Representacional), ha crecido exponencialmente ya que por ejemplo, entre otras cualidades, consume menos ancho de banda, lo cual lo hace más adecuado para el uso en internet o en redes lentas.

En sí, una API consigue que los desarrolladores interactúen con los datos de la aplicación a través de una serie de servicios, de un modo planificado y ordenado, siguiendo unas reglas, y sin necesidad de conocer exactamente cómo están programados dichos

servicios.

El concepto REST engloba una serie de especificaciones que inicialmente definieron la arquitectura de software para sistemas distribuidos, los cuales debían de alguna manera intercambiar información. En la actualidad, y de forma general, se usa para describir cualquier interfaz entre sistemas que utilice directamente el protocolo HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (en el caso de la API REST de WordPress, será JSON).

Al estar basada en HTTP, todas las operaciones sobre la información se realizará a través de operaciones POST, GET, PUT y DELETE, que permiten enviar, obtener, reemplazar y eliminar en su respectivo orden. El hecho de emplear HTTP, establece un protocolo común (no específico de ninguna tecnología o lenguaje de programación), lo que permite que el acceso a la API se pueda realizar desde cualquier medio que soporte el intercambio de datos mediante HTTP y únicamente mediante una URL.

Este diseño presenta grandes ventajas, una de ellas es la separación entre la interfaz de usuario del servidor y el almacenamiento de los datos, mejorando la portabilidad de la interfaz a otro tipo de plataformas, logrando aumentar la escalabilidad de los proyectos y permitiendo que los distintos componentes de los desarrollos puedan evolucionar de forma independiente. Con esto se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta convirtiendo a las aplicaciones en productos más flexibles a la hora de trabajar. Otra ventaja que se da gracias a la separación entre el cliente y el servidor es la libertad a la hora de cambiar o probar nuevos entornos de desarrollo, con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es

indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Es por esto que se decidió realizar la implementación de una API RESTfull con Laravel para aprovechar entre otras cosas la separación entre el cliente y el servidor con el objetivo principal de que los dos desarrollos sean independientes y puedan ser modificados sin afectar gran parte del sistema.

Otra decisión de implementación que se tuvo en cuenta fue como trabajar con las base de datos y en Laravel tenemos varios caminos válidos según su nivel de abstracción, es decir, de más o menos alto, para cubrir todas las necesidades de proyectos, situaciones o perfiles de desarrolladores. Entre estos se encuentra **Eloquent ORM, Fluent Query Builder** y **Raw SQL.**

Con Eloquent, y en general con cualquier ORM (Object-Relational mapping) tenemos la posibilidad de trabajar con los datos que hay en las bases de datos por medio de objetos, es decir, los datos de las tablas se mapean a objetos, evitando todo el trabajo de escribir las consultas SQL . El acceso a sus relaciones también se realiza por medio de propiedades de objetos, lo que facilita mucho las dinámicas de acceso a la información, siempre que haya una relación de dependencia declarada entre las distintas entidades de nuestro modelo de datos.

Eloquent implementa el patrón "Active Record", el cual, permite almacenar en bases de datos relacionales el contenido de los objetos que se tiene en memoria. Esto se hace por medio de métodos como save(), update() o delete(), provocando internamente la escritura en la base de datos, pero sin que nosotros tengamos que componer las propias sentencias.

En Active Record una tabla está directamente relacionada con una clase. En ese caso se dice que la clase es una envoltura de la tabla. La clase en sí es lo que conocemos

en el framework como "modelo", por ejemplo cuando nosotros creamos un nuevo objeto de ese modelo y decidimos guardarlo, se produce la creación de un registro de la tabla.

ORM sería entonces la herramienta de persistencia y Active Record el patrón de arquitectura que se sigue para su construcción. Eloquent es el nombre con el que se conoce en Laravel esta parte del framework, el cual, nos agiliza la mayoría de las operaciones habituales del acceso a bases de datos. [6]

Por otra parte, se encuentra Query Builder cuyo nivel no es tan bajo como escribir las consultas en SQL crudo, pero sí más cercano al sistema gestor de base de datos de lo que sería el ORM. Query Builder contiene una serie de funciones listas para realizar las operaciones más comunes con una base de datos, pero sin usar el lenguaje SQL directamente, sino el API de Laravel.

Entre las características se destaca que permite abstraerse del motor de base de datos que estemos usando , y a su vez, usa internamente los mecanismos de PDO, incluido el bindeo de parámetros en las consultas. Por tanto, no es necesario filtrar los datos que vamos a usar en las sentencias, ya que éstos serán filtrados automáticamente para protegernos de ataques por inyección SQL. [7]

En comparación con Eloquent, este último simplifica todavía más el trabajo con las bases de datos, sin embargo Query Builder consigue completar las funciones en una menor cantidad de tiempo. Para un juego de datos o número de operaciones pequeñas ese tiempo será imperceptible, pero a medida que sube la carga se podrá ver qué Query Builder es un poco más rápido.

De entre todos los mecanismos que nos propone Laravel para acceso a datos, el de más bajo nivel es "Raw SQL", que sería como escribir las consultas SQL "en crudo". En este tipo de acceso realizamos directamente el SQL de las consultas, que ejecutamos vía la clase BD que es uno de los “Facades” que nos ofrece Laravel.

Las consultas que se escriben tendrán SQL puro, por lo que se usará el SQL propio del sistema gestor que estés utilizando en este momento a diferencia de Query Builder o el propio ORM Eloquent que son independientes de la base de datos que se use.

Al igual que Query Builder, este tipo también tiene bindeo de parametros en las consultas mejorando la seguridad en el trabajo con bases de datos.

El motivo de la existencia de Raw SQL es que hay ocasiones en las que las operaciones con las bases de datos son complejas o hay consultas que no se pueden hacer a través de otros mecanismos, o bien porque se desea que se ejecuten más rápido. En ese caso podemos bajar hasta el nivel de la base de datos y ejecutar las consultas directamente contra el sistema gestor. [8]

Para la implementación del presente proyecto se decidió utilizar Raw SQL ya que uno de los requisitos previos era poder extenderlo utilizando otro grupo de desarrolladores, por lo que el SQL plano es la mejor opción ya que es lo universal y es lo que se aprende durante la carrera además de facilitar el debug en caso de errores. Otro motivo por el cual se optó por este método es por la necesidad de optimizar el sistema lo mayor posible dado que los equipos con los que disponible el hospital no son los más actualizados y en comparación con los otros dos métodos los cuales buscan la facilidad de uso y no la velocidad . Otra razón y quizá la más importante por la cual se optó por Raw SQL es por la decisión de realizar una API RESTfull que se separe del desarrollo de la vista pudiendo cambiar cualquiera de las dos partes sin modificar la otra, con lo cual, si se decide en un futuro cambiar el backend por alguna otra tecnología no se tenga que realizar nuevamente el desarrollo de todas las consultas sino que las mismas se puedan reutilizar.

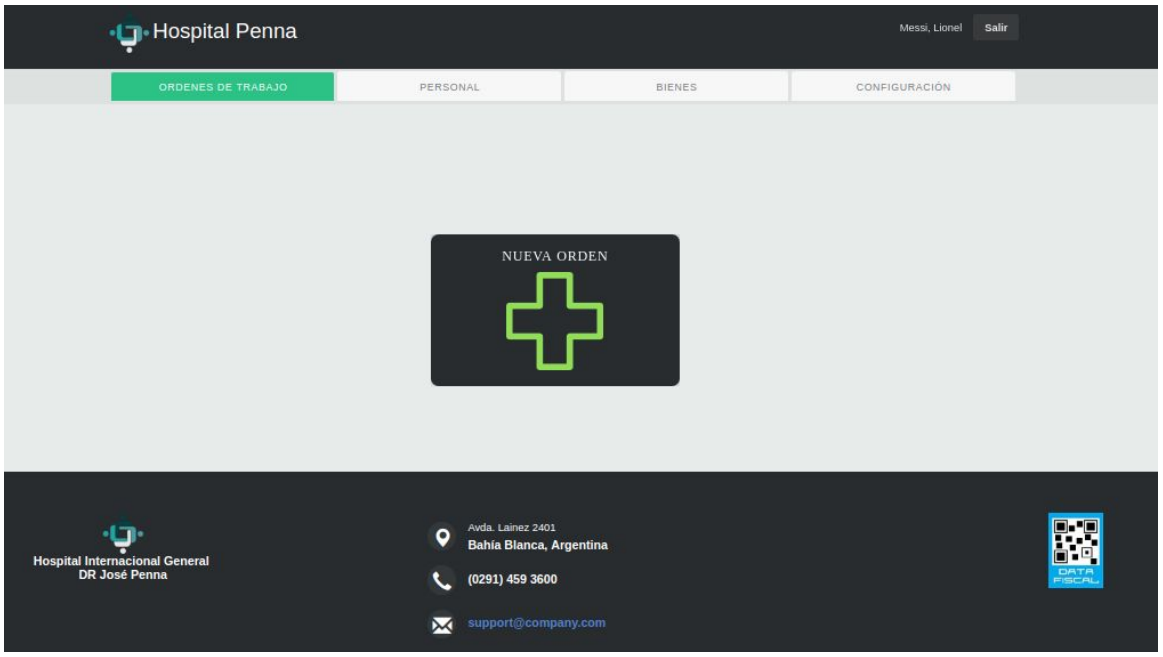
Cómo fue mencionado con anterioridad, para poder satisfacer cuestiones de seguridad que eran requeridas en el sistema, surgió la necesidad de la división para su

utilización, en diferentes tipos de usuarios, lo cuales, dependiendo de sus características, tendrán determinada función a lo largo del sistema. Las mismas serán detalladas a continuación en la próxima sección del capítulo.

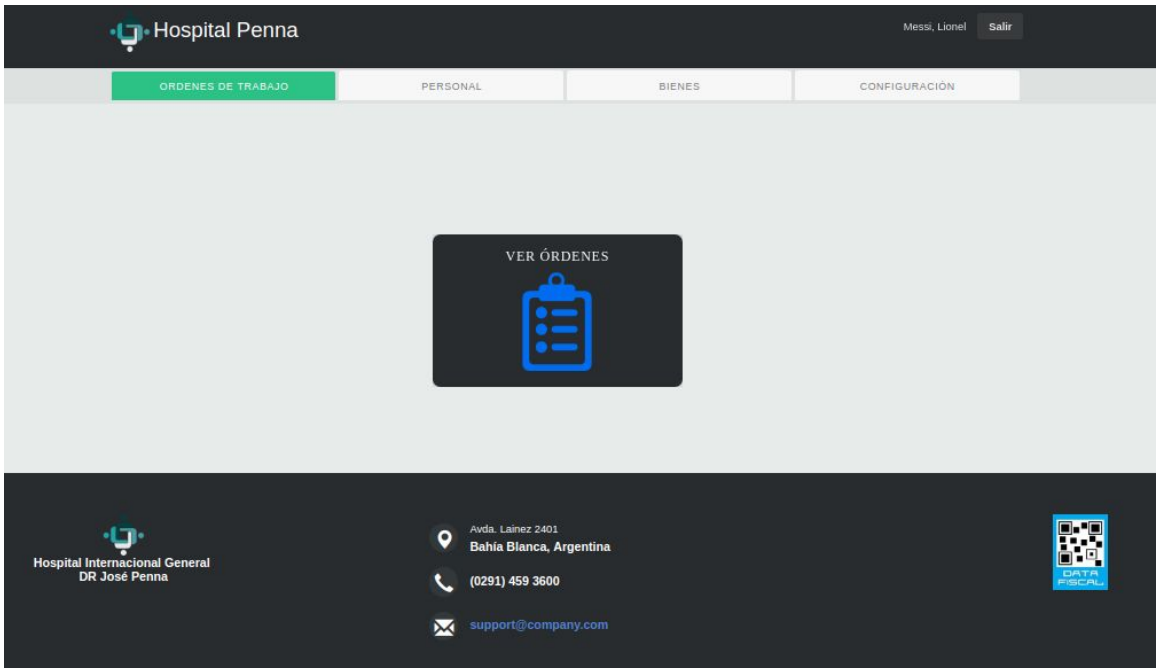
4.3 - Usuarios finales del sistema

En el ciclo de una orden de trabajo hay varios tipos de personas que intervienen dependiendo en qué parte del proceso se encuentre, es por esto, que en el sistema se decidió crear tres diferentes tipos de usuarios según el rol que cumplan en el mismo.

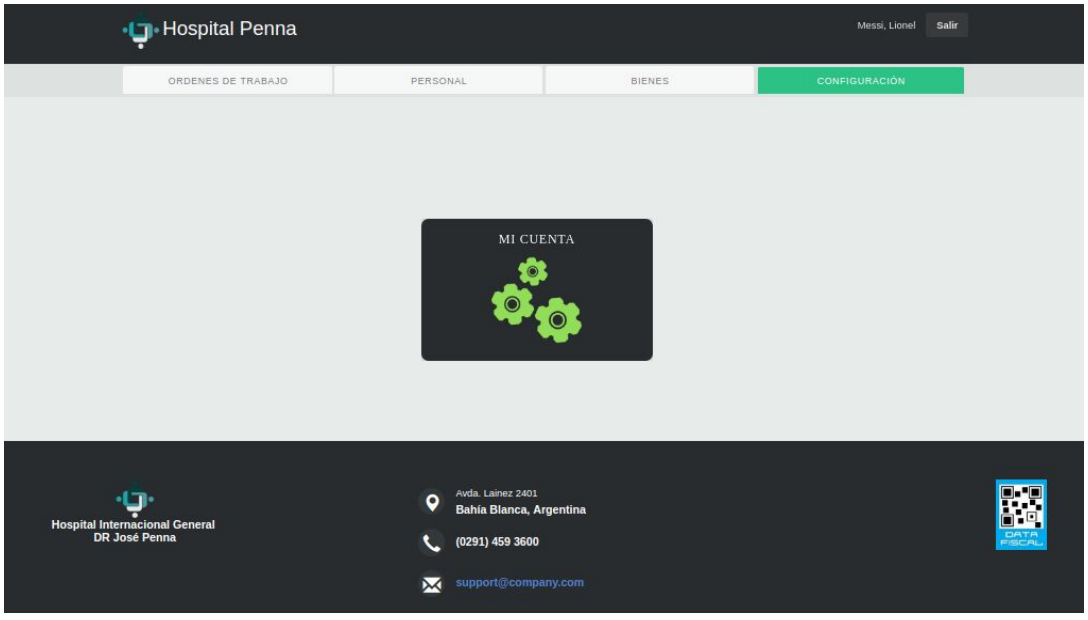
El primero de ellos es el "*Básico*" , cuyo rol principal en el sistema es crear las órdenes de trabajo identificando el bien a reparar, los problemas que este tiene y asignarla a la entidad correspondiente para su reparación. Para esto se le habilitó la opción "Nueva Orden" dentro del menú Órdenes de Trabajo.



Además de poder crear las órdenes este tipo de usuario tiene la capacidad de ver todas las que estén creadas, y así poder determinar si ya existe alguna para un dado bien y en qué estado se encuentra. Una vez que una orden fue resuelta por parte de un técnico el usuario que la creó tiene que dar su conformidad del trabajo realizado, por todo esto es que se le habilitó la opción “Ver Órdenes”

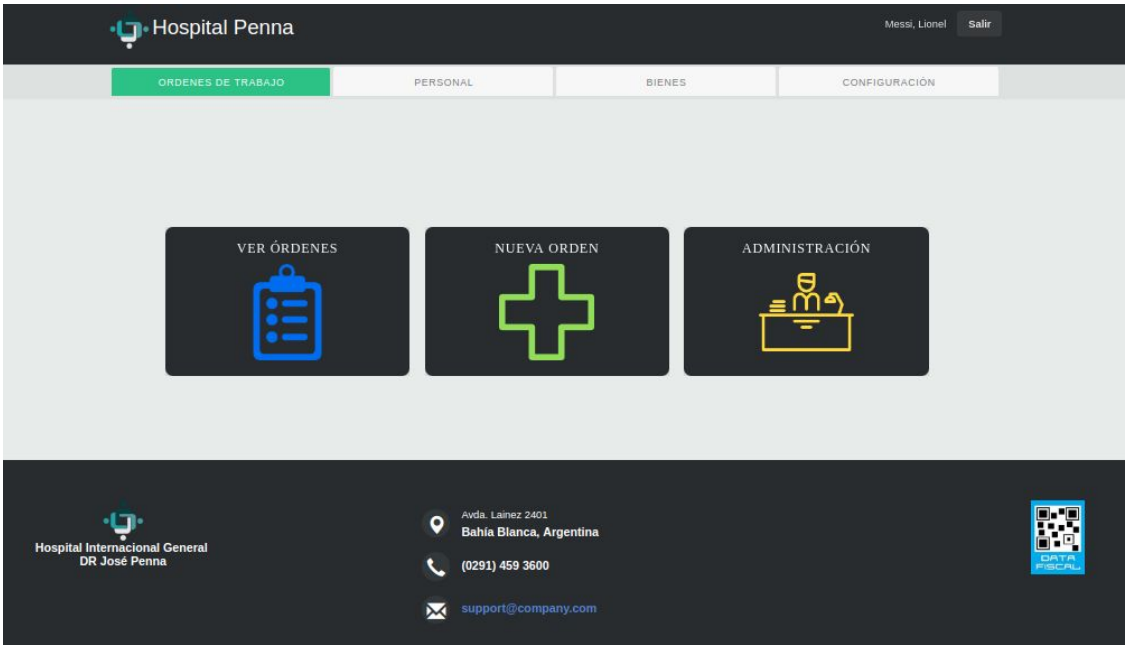


Todos los usuarios del sistema van a disponer de una sección “Configuración” en la cual van a poder realizar el cambio de su contraseña.



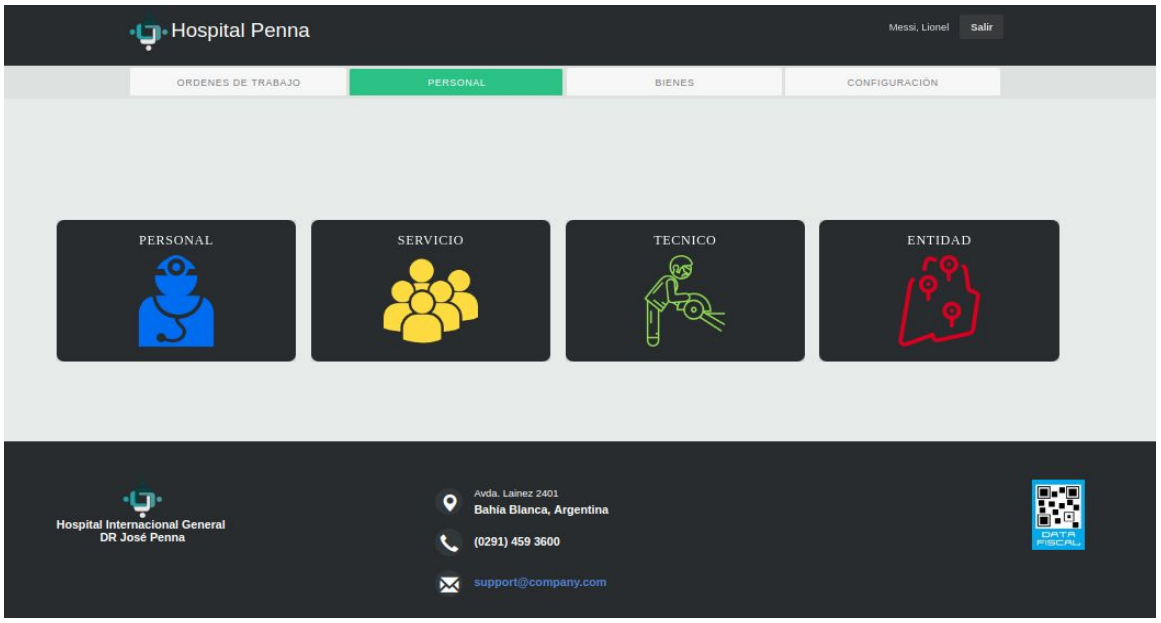
El segundo tipo es el “*Técnico”*, que, además de poder crear órdenes de trabajo como todo usuario, tiene la capacidad de resolverlas, es decir, una vez que un usuario

común crea una orden de trabajo y la asigna a una entidad, tiene la responsabilidad de solucionarla en caso de corresponder a su entidad y sino deberá derivarla a otra que pueda resolverla, es por esto que se le habilitó la opción “Administración”.

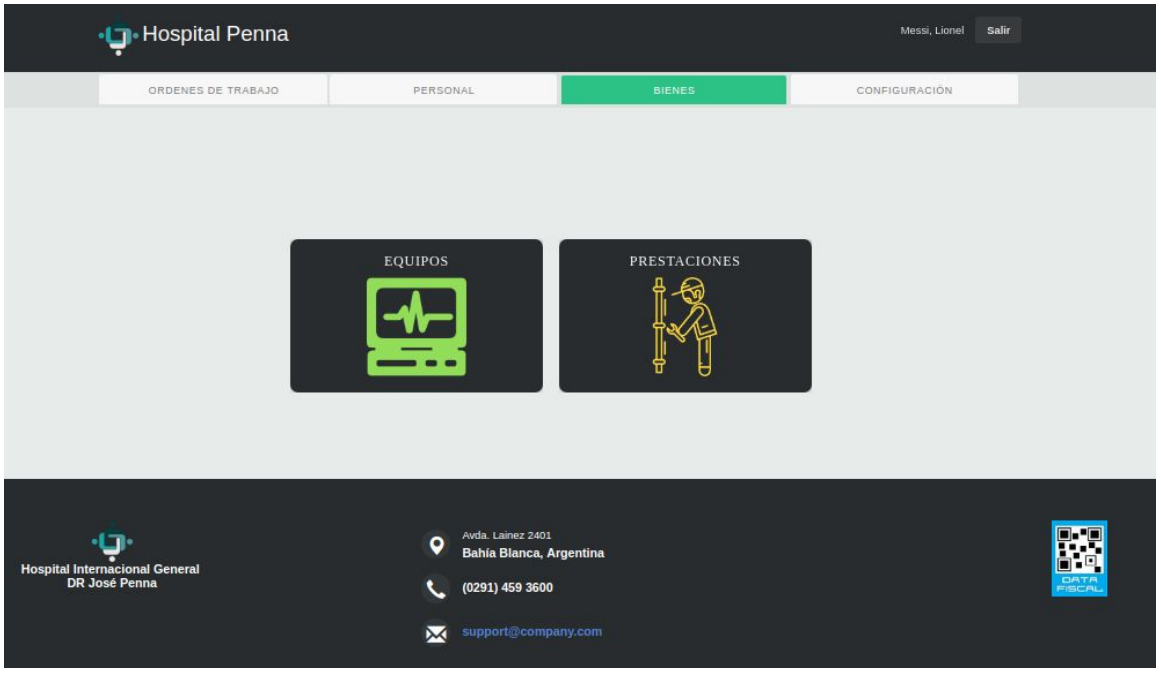


El último de ellos es el “*Administrador*” que además de poseer los niveles de permisos de todos los usuarios anteriores, tiene la tarea principal de mantener actualizado los datos básicos que el sistema necesita para funcionar, como pueden ser los datos del personal, entidades, tecnicos y servicios, además de los equipos y prestaciones. Esta es información sensible que el resto de los usuarios no debería tener acceso a ellos, es por esto que el único que podrá ingresar a estas secciones será el *Administrador*.

Para una mejor gestión se decidió dividir sus tareas en dos secciones: Personal



La otra sección corresponde a la representación de los Bienes del hospital.



Como se mencionó anteriormente, además de tener el trabajo de mantener actualizados los datos, será el encargado de crear los otros tipos de perfiles. Es importante

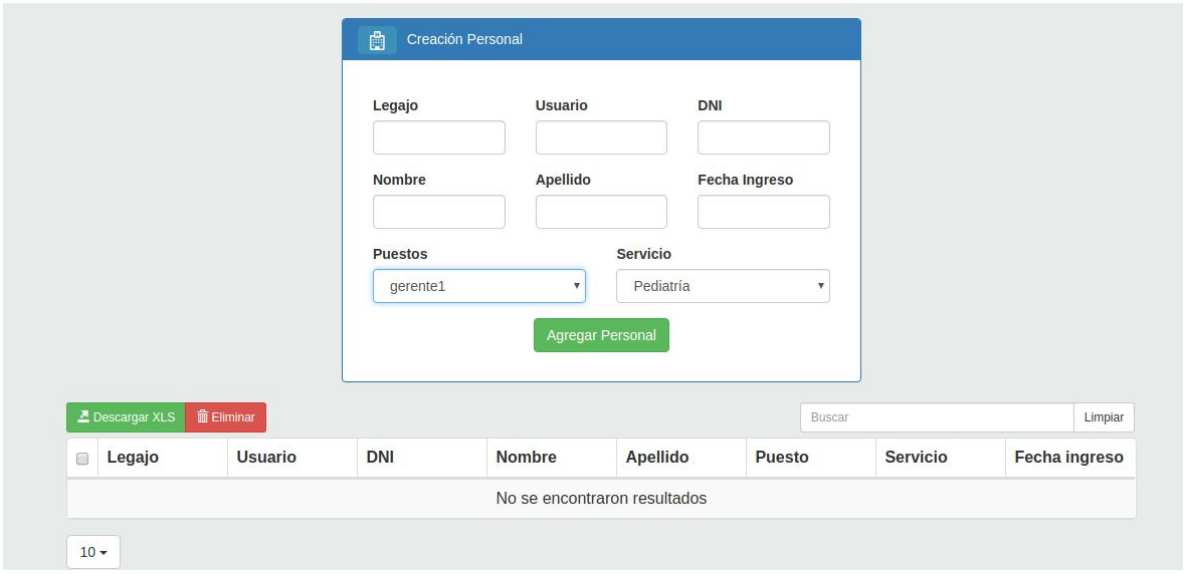
mencionar, que como decisión de implementación, se determinó que al momento de dar de alta una nueva “persona” en el sistema, se le estará asignando el perfil *Básico*. A su vez, al asignar a un usuario, previamente creado, a una entidad en la opción “Técnico” estará actualizando el perfil de dicho usuario al perfil de *Técnico.*

En la próxima sección del capítulo se hará una breve descripción del funcionamiento que cada una de las opciones.

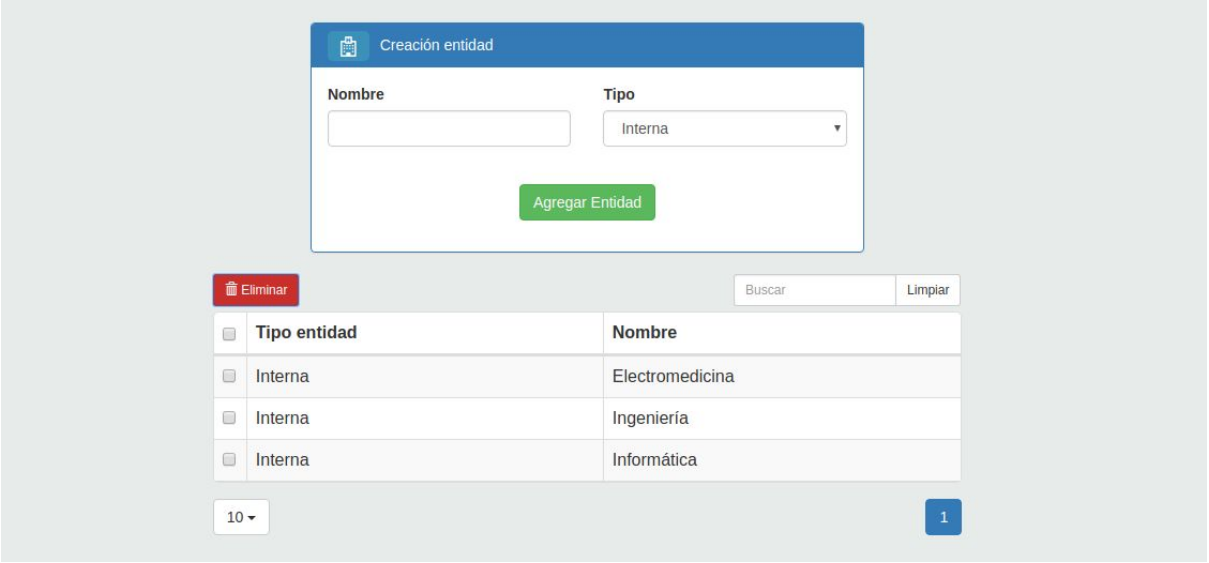
4.4 - Usabilidad del sistema

Como el sistema cuenta con varios subsistemas que lo componen, en esta sección se hará una breve explicación de cómo funciona cada uno de ellos, comenzando con el subsistema *Personal*, cuya primera opción se utiliza para llevar un registro actualizado de todos los empleados del hospital. Aquí el administrador puede visualizar y modificar cada uno de los perfiles de los empleados.

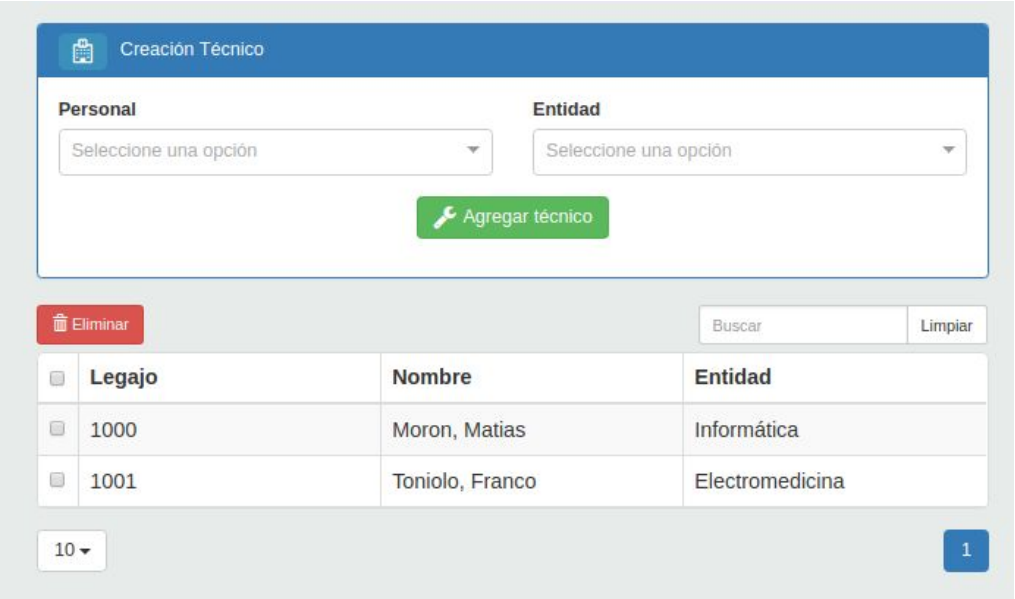
Entre los datos relevantes se encuentran el nombre, apellido, servicio al que pertenece y nombre de usuario con el que va a ingresar al sistema.



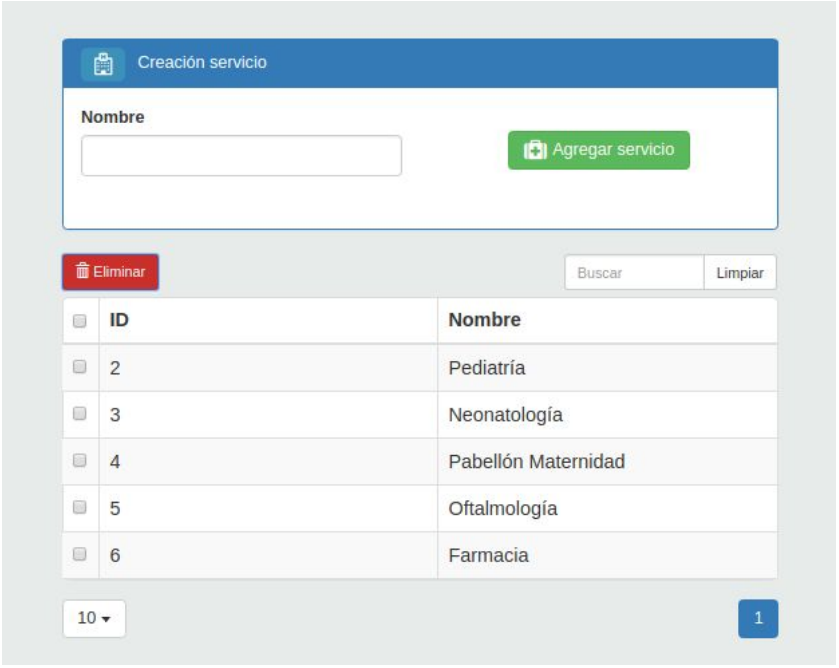
Otra opción dentro del subsistema de Personal que interviene, son las *Entidades*, que al igual que en el caso del *Personal* sirve para mantener actualizado todas las entidades que se encuentran en el hospital.



La opción *Técnico* se utiliza para mantener la relación de los usuarios que tienen la capacidad de resolver las ordenes de trabajo, es decir, los técnicos. Para lograr esto se debe elegir un usuario y asignarlo a la entidad que corresponda.

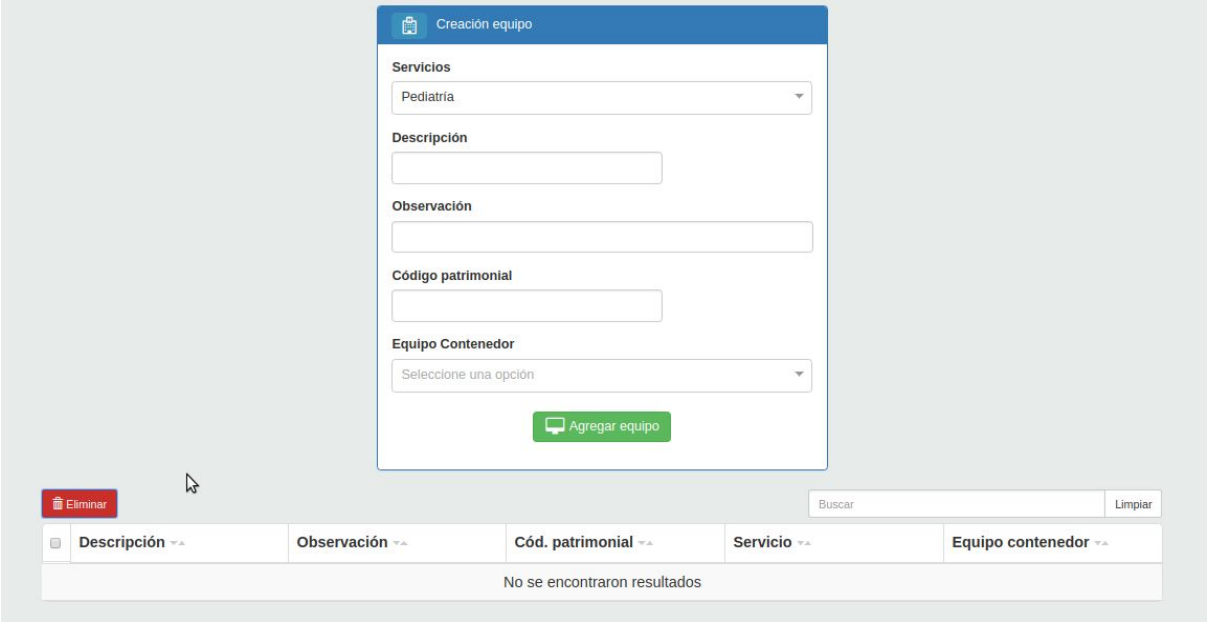


Y por último, dentro del subsistema de *Personal* se encuentra la opción “Servicio”, la cual servirá para administrar todas las áreas que componen el hospital.

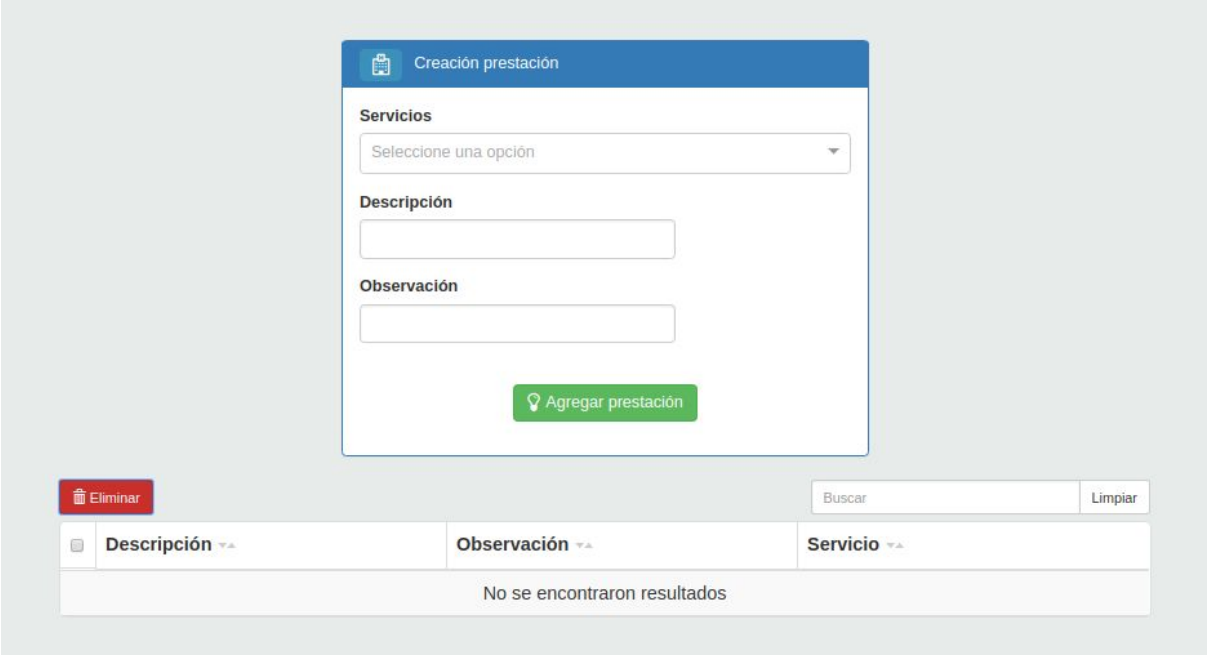


Luego se encuentra el subsistema de *Bienes* que como se mencionó en capítulos anteriores se los dividió en *Equipos* y *Prestaciones*.

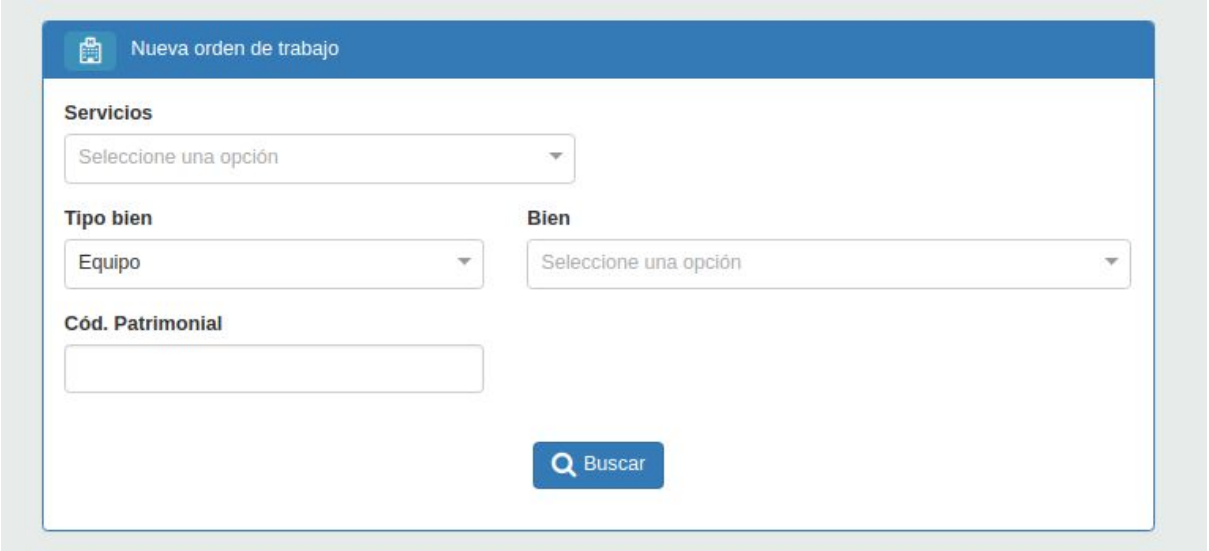
Los equipos son los bienes tangibles con los que cuenta el hospital como por ejemplo gammágrafos,camillas,etc. Entre las características que posee, se destaca el número patrimonial que es un código que identifica unívocamente a cada equipo. A su vez otra cualidad que estos tienen, es que uno equipo puede estar compuesto por otros equipos previamente creados. Como decisión de implementación, se determinó que no será posible eliminar un equipo “padre” si es que este contiene “hijos” dado de alta.



Por su parte en la sección “*Prestación*”, se administrarán todos los bienes que no tienen un código patrimonial y que pueden no pertenecer a un servicio. Un ejemplo de prestación, son los servicios como el agua, electricidad, gas, entre otros.



Por último se encuentra el subsistema de “Órdenes de Trabajo”, el cual contiene tres opciones: La primera de ellas es “Nueva Orden” donde cualquier usuario del hospital puede crear una orden de trabajo. Primeramente debe, utilizando los filtros, buscar el bien sobre el cual quiere generar la orden de trabajo. Para esto tiene los campos: servicios, tipo de bien, bien, número patrimonial.

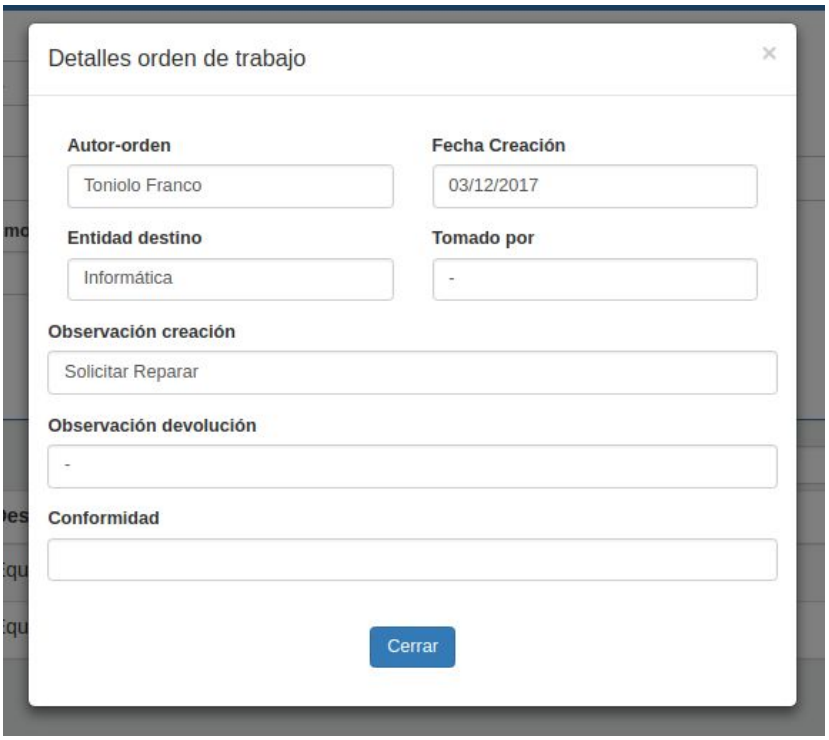


Una vez que se aplicaron los filtros, el sistema muestra una lista de bienes con los que concuerda con la búsqueda realizada.

En este paso únicamente se puede crear la orden de trabajo si el estado es DISPONIBLE, es decir, si ya no existe otra, previamente creada para dicho bien que todavía no está resuelta.

Si el estado es Disponible se puede proceder a crear una orden de trabajo, para esto debe completar algunos datos como una descripción de cuál es el problema que tiene el bien y luego debe seleccionar una entidad de destino, es decir, quien será la responsable de atender la petición.

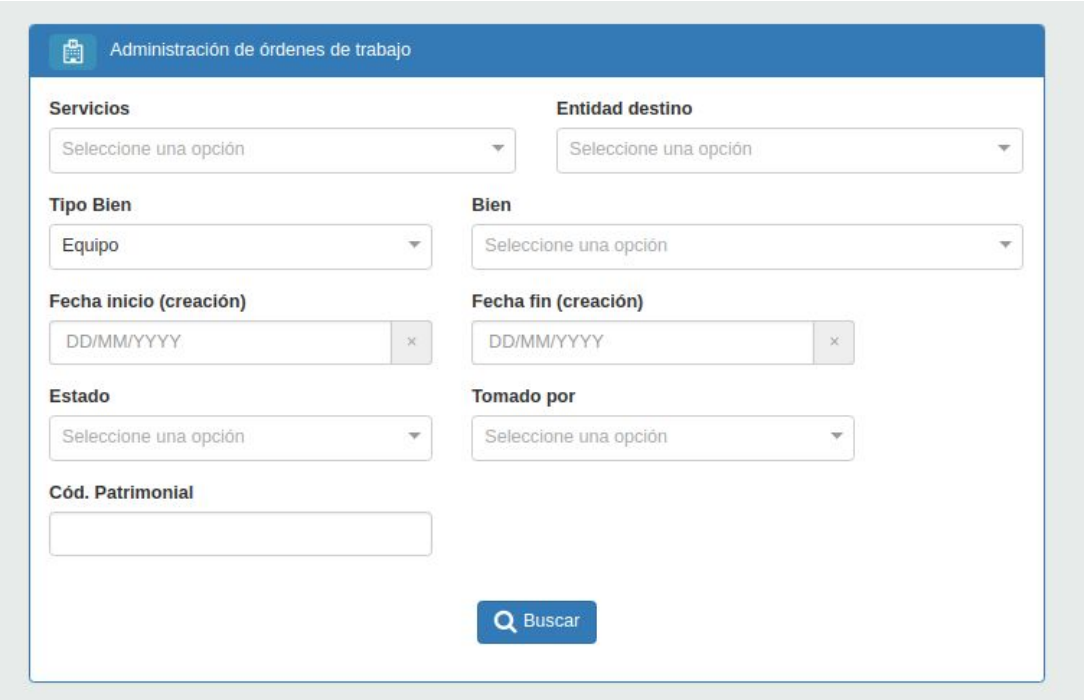
En el caso que el estado sea distinto de *Disponible* el usuario podrá visualizar una breve descripción de la orden de trabajo.



Luego está la opción “Administración” que es únicamente para los técnicos. Aquí tendrán las responsabilidad de administrar cada una de las órdenes de trabajo que estén en su entidad.

Inicialmente tienen la posibilidad de ver todas las órdenes de trabajo que estén en su entidad y cuyo estado sea *Disponible, Pendiente y en Curso*, opcionalmente pueden buscar otras aplicando algunos de los siguientes filtros:

- Servicio que pertenece el bien de la orden de trabajo
- Tipo bien
- Bien
- Número patrimonial en caso de ser un equipo
- Fecha de inicio y fin
- Estado de la orden de trabajo
- Por que técnico fue tomada



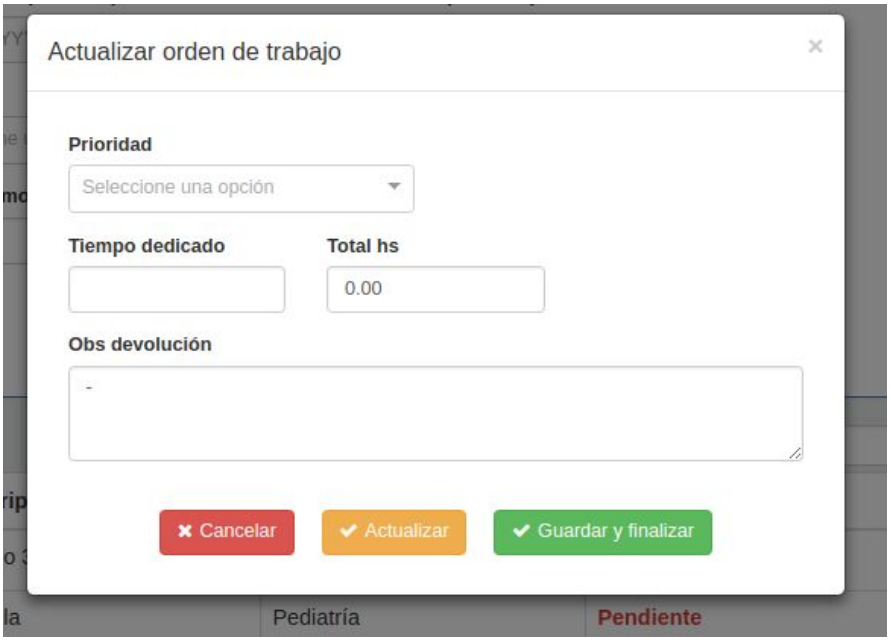
Los técnicos tienen distintas acciones que pueden realizar dependiendo de qué estado se encuentren las órdenes de trabajo.

Si se encuentra en estado pendiente puede:

- **Derivar** la orden a otra entidad para que la resuelva en caso de que el usuario se haya equivocado al momento de crearla.
- **Asignar** la orden a un técnico, describiendo que ya se comenzó a trabajar en ella
- **Ver mas** detalles de la orden de trabajo en cuestión.

Si se encuentra en *Curso* puede:

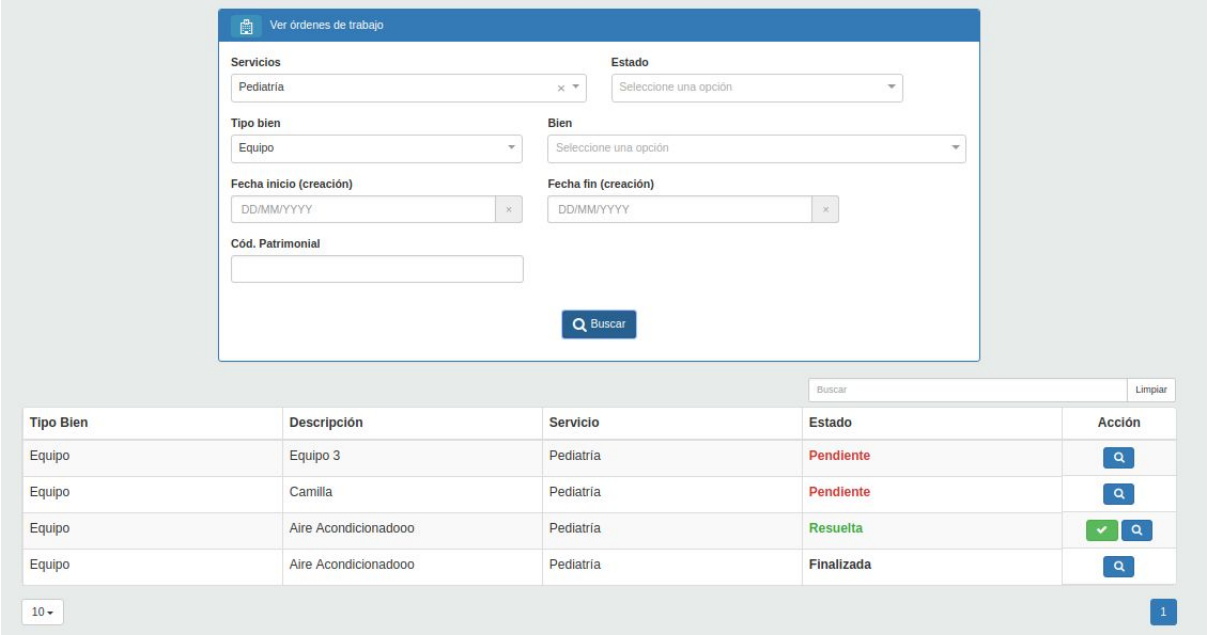
- **Actualizar** datos del trabajo realizado y de la orden de trabajo como son la prioridad, las horas insumidas y una observación de devolución para brindarle al usuario que la creó.
- **Dar por resuelta** la orden de trabajo completando los datos que se mencionaron anteriormente.
- **Ver mas** detalles de la orden de trabajo en cuestión



Si se encuentra en estado *Resuelta, Finalizada, Cancelada por el usuario* o *Rechazada* por el técnico puede:

- **Ver mas** detalles de la orden de trabajo en cuestión.

Y por último se encuentra "Ver Órdenes" , en esta sección se puede ver cada orden de trabajo de una forma más detallada. Es decir aplicando los filtros correspondientes se puede ver el historial completo de todas las órdenes de trabajo creadas en el sistema con sus respectivos detalles.



En el caso de que de haya una orden de trabajo que esté en estado *Resuelta*, el usuario que la creo va a poder dar su conformidad y así calificar el desempeño del trabajo realizado.

Esta sería la funcionalidad del sistema de intranet del Hospital Dr. José Penna correspondiente a la primera etapa de desarrollo planificada.

Resumen

Durante el desarrollo del capítulo se abordan las decisiones de implementación que se tomaron para el desarrollo del sistema, describiendo el porqué de la separación que se realizó en el manejo del frontend con el backend, para luego poder enfatizar en las características particulares de cada uno.

Se hará hincapié en los distintos tipos de usuarios con los que contará el sistema, dependiendo de las responsabilidades que estos posean. Como así también en una descripción de cada subsistema desarrollado para la aplicación.

Conclusión

Durante el desarrollo del informe se evaluaron y fundamentaron las elecciones de las diferentes alternativas acerca de las tecnologías y aquellas decisiones de diseño e implementación para poder desarrollar un sistema ágil, intuitivo y sobre todo fácil de extender con el objetivo de que en un futuro no muy lejano dicho sistema se siga ampliando con un nuevo grupo de desarrolladores para darle al hospital una herramienta para organizar parte de su información, que hasta el día de hoy se manejaban en su gran mayoría de forma manual.

Bibliografía

Información sobre el framework laravel y sus funcionalidades

1. https://laravel.com/

Comparación de uso entre los distintos framework

2. https://trends.google.com.ar

Manual de uso y características del framework Angular

3. https://angular.io/

Característica de la librería React y sus funcionalidades

4. https://reactjs.org/docs/components-and-props.html
5. https://reactjs.org/

Característica de Fluent Query Builder

6. https://desarrolloweb.com/articulos/query-builder-laravel5.html

Característica de Eloquent ORM

7. https://desarrolloweb.com/articulos/laravel-eloquent.html

Característica de Raw SQL

8. https://desarrolloweb.com/articulos/raw-sql-laravel.html