2020

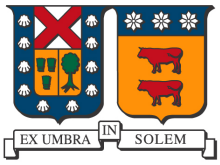
DESARROLLO DE UNA INTERFAZ GRÁFICA PARA MONITOREO Y PREDICCIÓN DE FALLAS POR TEMPERATURA Y LUBRICACIÓN DE UN MOLINO SAG

WEIPPERT MARTÍNEZ, ANÍBAL ARTURO

Universidad Técnica Federico Santa María

Departamento de Electrónica

Valparaíso Chile



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

"Desarrollo de una interfaz gráfica para monitoreo y predicción de fallas por temperatura y lubricación de un molino SAG"

ANÍBAL ARTURO WEIPPERT MARTÍNEZ

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO

PROFESOR GUÍA: JUAN I. YUZ EISSMANN

PROFESOR CORREFERENTE: CRISTÓBAL D. BADILLA CASTILLO

Índice general

Índice de figuras

Resumen

En el presente trabajo de título se diseñó, desarrolló e implementó una interfaz gráfica tipo dashboard para el monitoreo y predicción de fallas de un molino SAG, en particular el SAG-17 de la faena de Chuquicamata, CODELCO. La información utilizada para el monitoreo y predicción de fallas es extraída de archivos con información real del proceso de CODELCO y del molino SAG-17, por lo que el alcance de este trabajo es el monitoreo y predicción de fallas con datos históricos. Este trabajo de título se encuentra en el marco de una colaboración entre CODELCO y el Centro de Avanzado de Ingeniería Eléctrica y Electrónica (AC3E).

Para el desarrollo de la interfaz gráfica fue necesario estudiar y comprender qué se está haciendo y las tecnologías actualmente disponibles para el desarrollo de dashboards. Puntualmente el desarrollo de este dashboard fue dividido en tres secciones principales: back end, front end y base de datos. Por lo que el análisis de las tecnologías fue realizado para cada una de estas secciones.

El diseño de la interfaz gráfica abarcó distintos elementos, incluyendo: el diseño de los modelos de datos a utilizar y como se implementan estos en la base de datos, el diseño de la estructura del back end, los módulos presentes en el back end y el diseño de una vista preliminar de la interfaz gráfica e interacción básica de sus elementos.

Con el diseño realizado, se procedió a desarrollar el dashboard, aplicando lo estipulado en el diseño de este. Se desarrollaron módulos de: visualización y administración de detenciones del molino SAG-17, visualización y administración de señales del proceso del molino SAG-17 y visualización y administración de mantenimiento predictivo. A estos módulos se le realizaron ajustes a medida que eran desarrollados. Estos ajustes fueron realizados gracias a la retroalimentación obtenida de las reuniones periódicas sostenidas entre el AC3E y CODELCO.

Con el software de la interfaz gráfica ya desarrollado se realizó el despliegue de este prototipo (implementación) para realizar pruebas y ver su correcto funcionamiento. El despliegue se llevó a cabo utilizando contenedores de Docker. Con las pruebas de funcionamiento ya realizadas y aprobadas se tiene la implementación terminada y funcional de este prototipo para monitoreo y predicción de fallas. Al ser un prototipo este desarrollo, se exponen posibles mejoras, o caminos a seguir si se desea seguir desarrollando esta interfaz para obtener un producto terminado e implementable en la industria.

Abstract

In this thesis, a dashboard platform was designed, developed and implemented in order to monitor and predict failures of a SAG mill, in particular, the SAG-17 of Chuquicamata, CODELCO. All the information used to monitor and predict failures was real information gathered from the proces of CODELCO. Therefore, the scope of this work is to monitor and predict failures of a SAG mill with historical data. This thesis is part of a collaboration between CODELCO and AC3E (Advanced Center for Electrical and Electronic Engineering).

In order to develop a dashboard, it was necessary to study the state of the art in terms of dashboards and the technologies used to develop them. In this development the dashboard was divided into three main sections: back end, front end and database. Therefore, the analyzis of technologies was done to each one of them.

The design of the dashboard include different elements, including the following: design of the data models to use and their implementation in the database, the design of the structure of the back end, the modules of the back end, the design of a basic view of the dashboard and the interaction between the main elements of the dashboard.

the development of the dashboard was done based on the design of it. The following modules were developed: visualization and managment of stops in the SAG-17 mill, visualization and managment of process signals of the SAG-17 mill and visualization and managment of predictive maintanance.

With the dashboard developed, this prototype was deployed in order to do different tests and to check the right functioning of it in a production environment. The deployment of the dashboard was done with Docker containers..

After all the tests done and passed, the deployment of this prototype for monitor and predict failures of a SAG mill was complete. Because this development is

a prototype, possible paths of work or improvements are mentioned in case of continue this dashboard development in order to have a finished product to be implemented in the industry.

Glosario

Dashboard: Es una representación gráfica de los principales indicadores KPI (*Key Performance Indicator*) que intervienen en la consecución de los objetivos del negocio. Un dashboard debe transformar los datos en información y esta en conocimiento para el negocio.

Back-end: Es donde se procesan los datos provenientes tanto de la base de datos como del front end. En palabras sencillas, es el cerebro del software (dashboard en este caso).

Front-end: Es la parte del software con la que interactúan los usuarios de forma gráfica, siendo el responsable de obtener los datos de entrada que proporcione el usuario y desplegar los datos solicitados por el usuario.

Componente de React: Clase o función que recibe atributos llamados 'props' (proviene de propiedades) y devuelve elementos de React indicando qué debe aparecer y mostrarse en pantalla.

Callback: Función que se ejecuta luego de la finalización de otra función y que es pasada como argumento de entrada.

Props del Componente de React: Atributos de entrada de un componente de React. Estos son solo de lectura (no se pueden editar) y pueden ser de cualquier tipo: arreglos, texto, numérico, objetos y funciones (*callbacks*).

Estado del Componente de React: El estado es donde se almacenan variables que son propias del componente y que al ser actualizadas se renderiza nuevamente ese componente, actualizando lo que se ve en pantalla.

API REST (*Application Programming Interface - Representational State Transfer*)**:** Interfaz entre sistemas que utiliza HTTP para obtener datos o indicar operaciones sobre estos datos. Cada mensaje HTTP contiene toda la información necesaria para comprender una petición, por lo que no es necesario recordar el estado del servicio por ninguno de los participantes.

HTTP (*Hypertext Transfer Protocol*) **:** Protocolo de comunicación que permite la transferencia de información en internet.

JWT (*Json Web Token*) **:** Forma de transmitir información entre distintos puntos mediante un objeto JSON codificado y firmado con una llave única para verificar su veracidad. Poseen duración definida y no es necesario almacenar su estado.

JSON (JavaScript Object Notation) **:** Froma de transmitir información utilizada para intercambiar información. Es fácil de leer y entender para los humanos y fácil de generar y procesar para las máquinas.

Contenedor de Docker: Unidad estándar de software que contiene y encapsula todo el código y dependencias necesarias para que un programa funcione de forma segura e independiente respecto a otros contenedores de Docker.

Capítulo 1

Introducción

La capacidad de poder analizar y visualizar datos tales como: señales, KPIs, registros históricos, etc. es fundamental en la industria moderna si se desea tener un proceso exitoso. De esta necesidad se origina esta colaboración entre **CODELCO** y el Centro de Avanzado de Ingeniería Eléctrica y Electrónica (**AC3E**) en la que se busca el desarrollo de una interfaz para monitoreo y predicción de fallas en molinos SAG que incluye un modelo de degradación para mantenimiento predictivo en molinos SAG. En este marco se han realizado reuniones mensuales con José Allende, ingeniero de la Gerencia Digital y Analítica Avanzada de CODELCO.

Es necesario mencionar que si bien el titulo de esta memoria es "Desarrollo de una interfaz gráfica para monitoreo y predicción de fallas por temperatura y lubricación de un molino SAG", posterior a la inscripción del trabajo, se decidió orientarlo hacia una solución mas general titulada: "Desarrollo de una interfaz gráfica para monitoreo y predicción de fallas de un molino SAG". La segunda parte del trabajo mencionado previamente (modelo de degradación) está siendo desarrollado en paralelo como un trabajo de titulo, titulado "Desarrollo de un modelo de degradación de un molino SAG para mantenimiento predictivo"[1], en el que se busca encontrar parámetros, señales relevantes y regímenes de operación

del proceso de un molino SAG para poder construir un modelo de degradación y así poder realizar mantenimiento predictivo.

A raíz de las reuniones sostenidas con CODELCO a lo largo del proceso de este trabajo, se vio la posibilidad de utilizar archivos Excel generados en la faena misma de CODELCO que contienen registros diarios de detenciones de maquinaria y otras informaciones. El uso de estos registros permitió el desarrollo de uno de los módulos de la interfaz que mas información aporta hacia el proceso: módulo de detenciones.

Junto a los archivos previamente mencionados, se trabajó con registros históricos de múltiples sensores registrados en formato csv (*Comma Separated Values*), lo que permitió el desarrollo de un módulo de visualización de estos datos para mayor claridad sobre que está pasando en el proceso: módulo de señales.

El tercer módulo relevante es el relacionado a mantenimiento predictivo. En el marco de esta memoria se desarrolló un indicador de desgaste basado en el procesamiento básico de señales disponibles. En este se desarrollaron toda las bases para implementar a futuro el trabajo que busca encontrar un modelo de degradación del molino, Este es el módulo de PHM (*Prognostics and health management*).

Los 3 objetivos principales del presente trabajo de memoria son:

1. Desarrollar una plataforma tipo dashboard (panel de control interactivo para el usuario y de fácil manejo) para monitorear las señales de un molino SAG.

2. Integrar a dicha plataforma un módulo de predicción de fallas para mantenimiento predictivo (a ser desarrollado por otro memorista) en base a datos historicos o en operación normal.

3. Realizar pruebas funcionales en terreno del prototipo desarrollado para ajustes en base a necesidades del usuario

Es por lo anteriormente mencionado que en este proyecto se busca establecer los cimientos de esta colaboración entre la academia y la industria desarrollando un prototipo funcional que integre de forma exitosa ambos trabajos de título y validado por CODELCO.

Capítulo 2

Estado del Arte

El objetivo principal de esta memoria es el desarrollo de una interfaz gráfica tipo dashboard. Para el desarrollo se consideró la siguiente estructura del software: front end, back end y base de datos. Esto se ilustra en la Figura 2.1. En este capítulo se analizan alternativas existentes tanto para dashboards como para el desarrollo de front end, back end y base de datos.

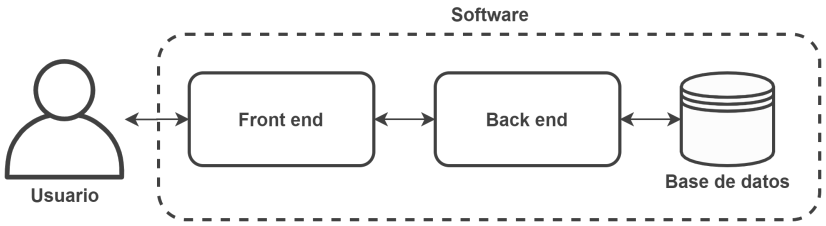


Figura 2.1: Módulos del sistema y sus principales interacciones

2.1. Dashboards Existentes

Para poder desarrollar una plataforma funcional fue necesario revisar que es lo que se está haciendo hoy en día en interfaces gráficas tipo *dashboard*. A continuación se exponen 4 proyectos *open source* con una breve descripción y sus vistas respectivas:

- Mozaïk [2]: Herramienta escalable, modular y personalizable. Está desarrollada en NodeJS y ReactJS y posee una comunicación optimizada con el back end. Visualmente se ve de muy buena calidad el trabajo realizado por el equipo de Mozaïk.

Figura 2.2: Vista de Mozaïk

- Dashbuilder [3]: Esta es una herramienta de código abierto desarrollada en Java y de un uso muy simple mediante *drag and drop*. Contiene múltiples tipos de gráficos y conectividad tanto con bases de datos como con archivos de texto plano.

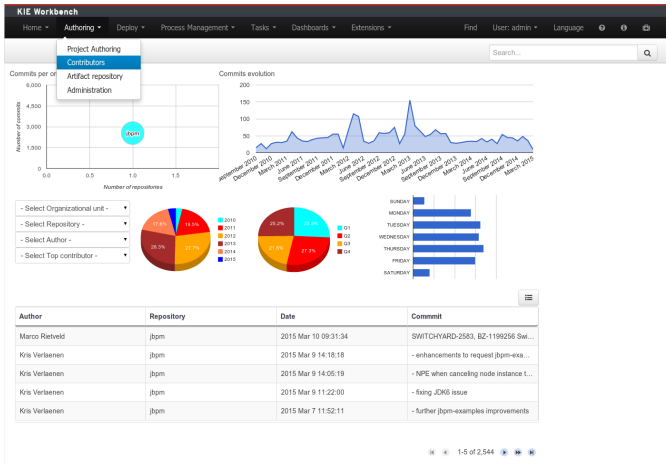


Figura 2.3: Vista de Dashbuilder

- Grafana [4]: Popular herramienta para visualizar datos en tiempo real, orientado a cualquier tipo de base de datos. Es bastante fácil su manejo y no requiere ningún conocimiento extra. Es altamente personalizable y puede medir cualquier tipo de dato y desplegarlo con múltiples tipos de gráficos.

Figura 2.4: Vista de Grafana

- Metabase [5]: Visualizador de datos de fácil manejo, orientado a todo tipo de usuarios y gráficos minimalistas. Está desarrollado en Clojure y corre sobre JVM (Java Virtual Machine).

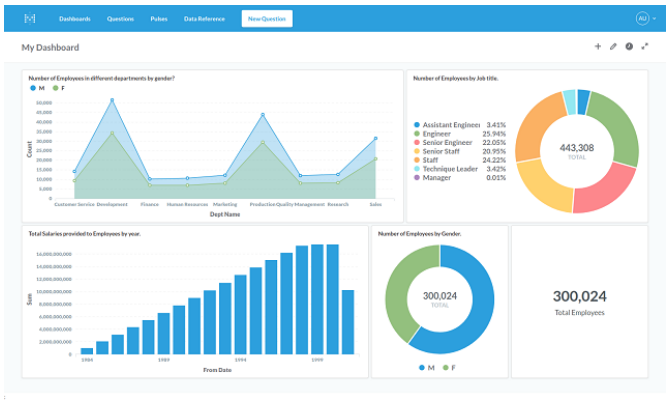


Figura 2.5: Vista de Metabase

Es importante destacar que las cuatro alternativas mostradas no corresponden a librerías de código ni *frameworks*, estas son soluciones integrales que utilizan múltiples herramientas, lenguajes y *frameworks*

2.2. Alternativas de Solución

A continuación se describen las alternativas disponibles para las 3 secciones del software a desarrollar (Ver Figura 2.1) y se incluye una cuarta categoría correspondiente al entorno en donde funcionará este software. En cada alternativa se exponen tanto sus fortalezas como sus debilidades.

2.2.1. Base de Datos

MySql [6]

Es un sistema de gestión de bases de datos relacional, es de código abierto (solo para servicios no comerciales) y desarrollada por Oracle.

- Fortalezas
  - Es la base de datos mas popular, o la mas popular dentro de los servicios web, por lo que cuenta con basta documentación y comunidad activa
  - La curva de aprendizaje no es difícil, ya que posee herramientas que facilitan el manejo y configuración de la base de datos
  - Al ser simple, permite gran velocidad en el manejo de datos
- Debilidades
  - Su licencia no es completamente de código abierto. Hay casos en los que se debe pagar por su uso.
  - Al ser desarrollada por una empresa, y no por la comunidad, la solución de problemas es levemente mas lenta.
  - Tiene algunos problemas de estabilidad respecto a sus competidores.

PostgreSQL [7]

Sistema de gestión de bases de datos relacional orientado a objetos y de código libre desarrollado por The PostgreSQL Global Development Group.

- Fortalezas
  - Código abierto, haciendo sus costos muy bajos respecto a los competidores.
  - Posee una herramienta gráfica para para el diseño y administración de bases de datos.
  - Buena escalabilidad tanto en el uso del hardware como la estructura misma de esta, pudiéndose ajustar parámetros o el hardware mismo para mejorar el rendimiento.
- Debilidades
  - Al no tener un canal de soporte oficial, la ayuda no es obligatoria por parte de la comunidad.
  - La sintaxis no es intuitiva, haciendo difícil aprender ciertos comandos.
  - El uso de memoria crece bastante según la cantidad de peticiones hacia la base de datos.

Microsoft SQL Server [8]

Sistema de gestión de bases de datos de manera relacional desarrollada por Microsoft, utiliza lenguaje Transact-SQL, debido a sus desarrolladores este solo ha estado disponible para sistemas Windows.

- Fortalezas
  - Posee un potente entorno gráfico de administración.

- Es un producto con gran comunidad y mucho tiempo en la industria, haciendo mas fácil la solución de problemas.
- Altos niveles de seguridad, haciéndola poco vulnerable a ataques informáticos.

- Debilidades
  - Consume grandes cantidades de RAM respecto a sus competidores.
  - Mala implementación de los tipos de datos y variables
  - Limite de conexiones simultáneas para una misma página

MongoDB [9]

Gestor de bases de datos no relacional desarrollada por Mongo DB. Esta almacena sus archivos en formato JSON, haciendo fácil su lectura y aumentando la flexibilidad de los datos almacenados.

- Fortalezas
  - Gran escalabilidad en su servicio
  - Al ser no relacional, presenta flexibilidad en los datos a almacenar
  - Buena capacidad de recuperarse ante fallas en términos de tiempo.

- Debilidades
  - No se recomienda para datos ordenados y relacionales.
  - No se pueden realizar transacciones entre distintos documentos
  - Aun es una tecnología joven, por ende algunas soluciones no están desarrolladas de la mejor forma aun.
  - No es recomendada para aplicaciones con transacciones complejas.

2.2.2. Back End

En esta sección se exponen los posibles lenguajes a utilizar en la sección del back end y adicionalmente se menciona de forma breve algunos de los *frameworks* disponibles para el desarrollo de servicios web.

C# [10]

Lenguaje de programación multiparadigma desarrollado por Microsoft, Utiliza el modelo de objetos de la plataforma .NET, siendo bastante similar al de Java. El *framework* mas popular para desarrollo de aplicaciones web es ASP.NET, que incluye manejo de peticiones HTTP y HTTPS, autenticación de usuarios y uso de JSON. Destaca por su facilidad para implementar y escalar servicios.

- Fortalezas
  - Gran potencia y versatilidad en la resolución de problemas.
  - El rango de tipos de datos soportados es mas amplio y definido que en sus competidores.
  - Es un lenguaje muy popular y fácil de aprender debido a que toma elementos de múltiples otros lenguajes.
- Debilidades
  - Principalmente orientado a plataformas Windows, lo que lo hace poco compatible.
  - Cada vez que se quiere ejecutar nuevamente el código, independiente de la cantidad de cambios, este debe ser compilado en su totalidad.

Java [11]

Lenguaje de programación muy popular para aplicaciones del tipo cliente-servidor. Es un lenguaje orientado a objetos e imperativo. Un *framework* muy utilizado y completo para este lenguaje es Spring Boot, que permite desplegar aplicaciones web de forma rápida. Este framework es altamente escalable, maneja JSON y posee una buena autenticación de usuarios y seguridad. Además permite recibir configuraciones externas para distintos ambientes de ejecución tales como producción o desarrollo.

- Fortalezas
  - Permite la ejecución del código en cualquier tipo de sistema operativo, ejecutando el mismo programa en distintos ambientes.
  - Java se encarga automáticamente de ir liberando memoria durante la ejecución del programa, sin intervención del usuario (programador).
  - Buena gestión de errores durante la ejecución del software programado.
- Debilidades
  - Para un buen rendimiento, necesita de un equipo mas poderoso respecto a otros lenguajes.
  - Si bien funciona en múltiples plataformas, esto solo es posible si se tiene instalada la maquina virtual de java, que es en donde se ejecuta el programa. En caso de no tenerla el programa no puede ser ejecutado.
  - Si no se está familiarizado con la programación orientada a objetos, puede parecer engorroso en una primera etapa.

Node.JS [12]

Entorno de ejecución de JavaScript construido con el motor de JavaScript de Chrome. Este entorno está pensando para ejecutarse en el lado del servidor

y para construir aplicaciones en red escalables. El *framework* mas popular para desarrollo web es *Express*, que al igual que sus competidores permite el manejo de peticiones HTTP y HTTPS, autenticación de usuarios y una alta escalabilidad. La implementación de este es muy sencilla por lo que agiliza procesos de desarrollo y posee una gran velocidad de respuesta ante múltiples clientes.

- Fortalezas
    - Al ser de código abierto, el aporte de la comunidad es importante. En este caso se tiene el NPM (*Node Package Manager*) que es un repositorio con mas de 50.000 paquetes para desarrollo que cuenta con gran soporte y documentación.
    - Al estar basado en JavaScript, se puede desarrollar tanto el lado del servidor como el del cliente en el lenguaje.
    - Se puede acceder a los datos mientras estos se cargan, facilitando la codificación de datos en tiempo real.
    - Alta velocidad en el procesado de peticiones HTTP y muy escalable.
- Debilidades
    - Al tener un único hilo de ejecución, si se le encargan tareas muy demandantes de tiempo, detiene la velocidad de respuesta debido a los múltiples cambios entre la tarea demandante y las demás.
    - Al tener tipos de datos dinámicos, en ocasiones los objetos no tienen una definición clara de un tipo, haciendo difícil su reconocimiento.
    - Al estar basado gran parte en los *callbacks*, estos se anidan en el código y puede ser difícil su seguimiento y depuración al leerlos.
    - Si bien se tienen muchos paquetes en el NPM, al no ser oficiales debido a su naturaleza de código abierto, puede ser difícil elegir entre múltiples opciones para cumplir una misma tarea.

Python [13]

Lenguaje de *scripting* de código abierto administrado por la *Python Software Foundation*. Es un lenguaje multiparadigma, fácil de aprender y, por tanto, muy popular. Últimamente es de los lenguajes mas populares, si no el mas popular. Para desarrollo web existen dos *frameworks* que son muy utilizados: Flask y Django. Ambos poseen manejo de peticiones HTTP, autenticación de usuarios y buena escalabilidad, sin embargo es necesario mencionar que Flask es menos demandante en término de recursos y mas fácil de implementar para usuarios no expertos.

- Fortalezas
  - Fue pensado desde su origen, como un lenguaje simple, rápido de leer y de aprender, haciendo fácil su aprendizaje y su uso.
  - La comunidad Python es muy comprometida, con una gran cantidad de librerías desarrolladas y a disposición del usuario.
  - El código es multiplafatorma, la única condición es tener el interprete de Python, por lo que no es necesario compilar el código nuevamente para cada plataforma.
- Debilidades
  - Al ser programas interpretados, su ejecutción es mas lenta que los compilados (C y C++ como ejemplo).
  - Si la tarea a ejecutar es muy demandante de memoria, Python no es una buena elección.
  - En términos de acceso a bases de datos es bastante inseguro y limitaciones. Esto se debe a que el desarrollo de la capa de bases de datos es aun prematuro.

2.2.3. Front End

En el caso de alternativas para el desarrollo del front end, el análisis que se presenta es levemente diferente. Se utilizarán las siguientes tres herramientas para el desarrollo de un servicio funcional y visualmente atractivo:

- **HTML** (*HyperText Markup Language*) **:** Lenguaje de marcado utilizado para el desarrollo de páginas web mediante la declaración de *tags*.
- **CSS** (*Cascading Style Sheets*) **:** Lenguaje de diseño gráfico utilizado para darle la presentación y el diseño a un documento escrito en lenguaje de marcado como HTML.
- **JavaScript:** Lenguaje de programación interpretado, que es utilizado principalmente desde el lado del cliente en servicios web, mejorando así la experiencia del usuario al hacer mas dinámico el funcionamiento, permitiendo la interacción con servidores, procesamiento de datos y modificación de los elementos HTML de forma dinámica.

Para ejemplos de forma practica de los tres mencionados previamente, dirigirse a [14] o al sitio web de la World Wide Web Consortium [15].

La diferencia del análisis para el desarrollo de front end se debe a que es común utilizar librerías o *frameworks* que mejoran las capacidades de estos lenguajes. La diferencia fundamental entre un *framework* y una librería, es que la librería es un conjunto de herramientas pertenecientes al lenguaje y que uno ve cuando utiliza y controla en su totalidad, mientras que el *framework* decide el flujo del software y como este interactúa, teniendo uno que adaptarse a sus reglas de funcionamiento. Si bien no existe documentación oficial respecto a este asunto, recomiendo leer este hilo en stackoverflow ante cualquier duda.

Angular [16]

Angular es un *framework* para aplicaciones web y está desarrollado en Typescript (super-conjunto de JavaScript). Este framework es utilizado para crear aplicaciones web de una sola página y altamente adaptables a los distintos tipos de plataformas (smartphones, tablets, laptops, etc.).

- Fortalezas
  - Angular recibe soporte de Google, generando una documentación de alta calidad y buen soporte a los usuarios en los foros
  - Su estructura es modular, haciendo altamente escalable el desarrollo de la interfaz.
  - Al tener una arquitectura vista-controlador, logra separar la lógica de la aplicación con la interfaz de usuario.
- Debilidades
  - El estilo de Angular puede ser engorroso de leer debido a la gran cantidad de palabras utilizadas en el código.
  - El *debugging* del código puede ser engorroso y tener la capacidad de hacerlo sin problemas es crucial en el desarrollo de aplicaciones complejas y funcionales.
  - La curva de aprendizaje puede ser lenta en comparación con otras herramientas.

ReactJS [17]

React.Js es una librería de código abierto de JavaScript para construir interfaces de usuario dinámicas y rápidas. React.Js está basado en componentes reutilizables en los que se mezcla tanto la vista como el controlador. Para esta

finalidad es útil trabajar con JSX, una extensión de la sintaxis de JavaScript. A Continuación se muestra un ejemplo de esta sintaxis para ReactJS.

```
class Example extends React.Component {
  render() {
    var helloWorld = <h1>Hello, world!</h1>
    return (
      <div>
        {helloWorld}
        <h2>This is an Example</h2>
      </div>
    );
  }
}
```

Figura 2.6: Ejemplo sintaxis JSX en ReactJS

- Fortalezas
  - Fue diseñado para realizar aplicaciones altamente escalables y dinámicas.
  - Facebook es el encargado del desarrollo de la librería, teniendo así un gran soporte y una gran comunidad.
  - Al actualizar alguna parte de la interfaz, solo es actualizada esa, no afectando así el renderizado de toda la aplicación.
- Debilidades
  - React no separa la vista de controlador, por lo que algunos desarrolladores lo encuentran poco cómodo.
  - La librería de React es muy amplia, haciendo difícil cubrir todas sus herramientas.

- Al ser una librería, no tiene un modelo estandarizado para la estructura de la aplicación como el de Angular (*framework*), siendo responsabilidad del usuario la administración total de su aplicación.

Elementos de Apoyo

Adicionalmente a las librerías ya nombradas, a continuación se muestran dos más extras, que son capaces de trabajar en conjunto con las anteriores para facilitar la implementación de ciertas funcionalidades y elementos:

- Bootstrap [18] : Biblioteca multiplataforma de código abierto para diseños de aplicaciones web. Contiene una infinidad de elementos, desde íconos individuales a menús enteros de navegación. Estos estn basados en HTML,CSS y JavaScript.

- Material [19]: Librería multiplataforma de código abierto que ayuda en el diseño del sitio web, generando una experiencia digital de alta calidad en el usuario, este es implementable en múltiples lenguajes e incluye al igual que Bootstrap una infinidad de componentes y elementos a disposición del usuario.

2.2.4. Despliegue del Sistema

El despliegue, en palabras simples, es el proceso de configurar y dejar el software funcionando de forma estable y continua tal que los usuarios del producto puedan acceder y utilizarlo.
En esta sección se evaluaran las opciones para desplegar el software. Estas se dividen en dos grandes sub-grupos: máquinas virtuales o contenedores. Una breve explicación de ambos a continuación:

- Máquinas virtuales: Software que emula una máquina dentro de otra utilizando hardware dedicado. Para el caso del desarrollo de servicios se

tiene una máquina virtual por cada uno y este tiene todas las dependencias y configuraciones correspondientes para su correcto funcionamiento. A diferencia de los contenedores, cuando es necesario utilizar grandes recursos en la aplicación, es mejor utilizar una máquina virtual para aprovechar completamente las capacidades del hardware y software. Un ejemplo de máquina virtual es VirtualBox [20]

- Contenedores: Abstracción dentro de una gran máquina, en la que virtualmente se aíslan todas las aplicaciones. En cada aplicación se tienen los servicios necesarios para funcionar, como bases de datos o intérpretes de código. Todos los contenedores de aplicaciones corren sobre un único sistema operativo. La ventaja del uso de contenedores es que se elimina el uso de recursos específicos de cada máquina virtual para cada servicio, y permite un levantamiento mas rápido de cada aplicación y sus servicios correspondientes respecto a las máquinas virtuales. Dos servicios de contenedores muy utilizados son: Heroku [21] y Docker [22]

2.3. Alternativa Seleccionada

Para la elección de que tecnología ocupar en cada sección de la plataforma desarrollada se analizaron las fortalezas y debilidades de cada una de las alternativas y cómo estas se ajustan a este trabajo de memoria. Un criterio relevante al momento de hacer el análisis es que CODELCO posee una batería de tecnologías ya definida. Por lo que se optó por apegarse a este *stack* tecnológico formado por:

- Front end - **ReactJS**: Destaca por su escalabilidad, dinamismo y velocidad. Es importante tener como consideración que en ReactJS se mezcla controlador como vista, por lo que hay que acostumbrarse a esa forma de programar y diseñar. El manejo y flujo de la aplicación recae completamente en el

programador al ser una librería y no un *framework*, por lo que es un factor importante a tener en cuenta al programar.

- Back end - **NodeJS**: Gran apoyo de librerías y comparte lenguaje con el front end (JavaScript), haciendo más fácil el desarrollo de ambos. Es un lenguaje muy rápido y escalable, pero hay que tener cuidado con el tipo de tareas a programar debido a que NodeJS posee un único hilo de ejecución y, si se bloquea, puede ser bastante lento el tiempo de respuesta de las peticiones.

- Base de datos - **PostgreSQL**: Base de datos relacional con buen rendimiento, escalabilidad, de código abierto y con entorno gráfico de administración. Al ser de código abierto carece de soporte oficial y el uso de memoria ante múltiples peticiones no es el mas óptimo. Una ventaja es que la librería utilizada para NodeJS (node-postgres) es muy completa y fácil de implementar.

- Deployment del sistema - **Docker**: Muy fácil de implementar y altamente escalable. Se dispone de herramientas como docker-compose para el manejo en simultaneo de todos los contenedores, permitiendo configurar cada uno de estos por separado y todas sus dependencias. Docker es compatible con todo el resto de los lenguajes y tecnologías a utilizar.

Capítulo 3

Diseño de la Interfaz Gráfica

En el presente capitulo se describen los requisitos que debe cumplir tanto el back end como el front end, las consideraciones al momento de diseñar la estructura de ambos y cómo estos interactúan entre si y con la base de datos.

3.1. Requisitos de Diseño Back End

A continuación se enumeran los requisitos de diseño considerados para el back end de la interfaz desarrollada:

1. Al ser un prototipo, se trabajará con bases de datos históricos de forma local. Estos datos son entregados por CODELCO y contienen señales de los molinos SAG 16 y SAG 17 de Chuquicamata.

2. Debe ser capaz de conectarse con el front end para poder recibir consultas y entregar información almacenada en la base de datos.

3. Debe ser capaz de conectarse con la base de datos para almacenar y solicitar datos de forma dinámica (no programada).

4. Debe ser capaz de procesar archivos Excel relacionados a detenciones de los equipos de la faena (inclusive los molinos SAG) con un formato definido, también provistos por CODELCO

5. Debe ser capaz de procesar archivos CSV (*Comma Separated Values*) relacionados a registros historicos de señales, provistos por CODELCO para poblar la base de datos local

6. El código debe ser fácil de mantener y escalable.

3.2. Modelo de Datos a Almacenar

Se describe la estructura de la información que será almacenada en la base de datos. Los datos de la Figuras 3.1 y 3.2 corresponden a información extraída de archivos Excel y CSV, respectivamente. La Figura 3.3 corresponde a constantes e información utilizada para el cálculo de un índice de degradación básico del molino SAG que corresponde al módulo de PHM de la interfaz.

En las tres figuras que se presentan a continuación se describe brevemente cada campo y se muestra su nombre.

```
{
    "equipment": "Maquinaria o equipo correspondiente al evento de detención",
    "description": "Descripción breve sobre la detención registrada en el archivo",
    "init_t": "Fecha y hora de inicio de la detención",
    "end_t": "Fecha y hora de termino de la detención",
    "stopped_time": "Cantidad de horas que estuvo detenida esa maquinaria o equipo",
    "Date": "Fecha del archivo de detenciones",
    "Type": "Tipo de detención: programada o no programada",
    "Reason": "Listado de los equipos encargados de solucionar el evento de detención"
}
```

Figura 3.1: Modelo de datos relacionado a detenciones de equipos en Chuquicamata

```
{
    "sensor_var": "TAG identificador del sensor o señal",
    "engunits": "Unidades en las que se mide esa variable",
    "descriptor": "Breve descripción de la variable y a donde pertenece",
}
```

```
{
    "sensor_var_id": "ID identificadora de ese sensor o señal",
    "time_stamp": "Instante de tiempo en el que se tiene ese valor",
    "value": "Valor del sensor o señal para un determinado instante de tiempo",
}
```

Figura 3.2: Modelos de datos relacionado a señales de un molino SAG

Notar que se tienen dos modelos de datos en el caso de las señales de un molino SAG (ver Figura 3.2) debido a que de un único archivo CSV se extrajo tanto información de las señales disponibles como de los valores de estas mismas para determinados instantes de tiempo, esta separación se hizo para un mejor manejo de la información e implementación en la base de datos (ver Figura 3.5).

```
{
    "mod_date" : "Fecha de registro de esta entrada en la tabla",
    "init_t":"Fecha de inicio del calculo del índice de
    ↪ degradación",
    "end_t":"Fecha de término del calculo del índice de
    ↪ degradación",
    "initial_state":"Estado inicial para el calculo del índice",
    "alpha":"Valor del coeficiente alfa del modelo",
    "beta":"JSON que contiene tanto la ID como el valor (peso) de
    ↪ las señales involucradas en este índice",
    "mov_average": "Tamaño de la ventana del promedio móvil a
    ↪ utilizar para filtrar"
}
```

Figura 3.3: Modelo de datos relacionado a valores de índice de degradación

En la Figura 3.3 se menciona una ID en el campo 'beta', esta ID es la llave primaria (identificador único de una entrada en la base de datos) de las señales almacenadas en la tabla 'kpi_variables' (ver Figura 3.5). La estructura y explicación del modelo básico utilizado para el cálculo del índice de degradación será descrita en la sección 3.4

3.3. Estructura de la Base de Datos

Para poder desarrollar de forma correcta la base de datos local de la interfaz gráfica, se estimó necesario la utilización de cinco tablas, dos para detenciones, dos para señales y una para índice de degradación. A continuación se describen de forma breve su estructura.

3.3.1. Tablas de Detenciones

- **Dates:** Tabla en la que se almacenan todas las fechas de archivos de detención ya registrados para evitar duplicados. Estos son archivos Excel utilizados y provistos por CODELCO. La tabla Posee una llave primaria numérica.

- **Detenciones:** Tabla en la que se almacenan la información relacionada con cada detención. Posee una llave foránea (campo que hace referencia a otra tabla) correspondiente a la fecha registrada en la tabla "Dates"(ver Figura 3.4).

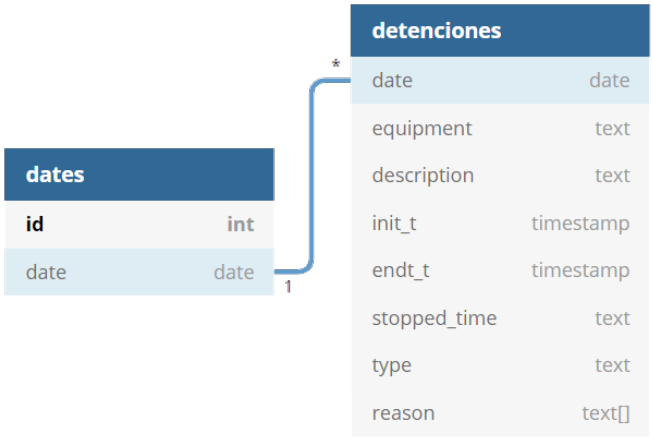


Figura 3.4: Relación entre tablas de detenciones y sus campos

3.3.2. Tablas de Señales

- **kpi_variables:** Tabla en la que se almacenan todos los tags con su descripción y unidad.

- **kpi_data:** Tabla en la que se almacena el valor de una señal para un determinado instante de tiempo. Su llave foránea es el campo “id” de la tabla “kpi_variables” (ver Figura 3.5). Para mejor rendimiento al momento

de guardar y buscar en la base de datos se añadió un índice por fecha en esta tabla (columna 'time_stamp')

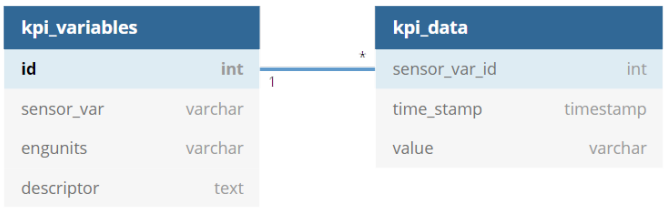


Figura 3.5: Relación entre tablas de señales y sus campos

3.3.3. Tablas de Índice de Degradación

- **phm_index_values:** Tabla en la que se almacena toda la información necesaria para calcular el índice de degradación (ver Figura 3.6) que se explica en más detalle en la sección 3.4 a continuación.

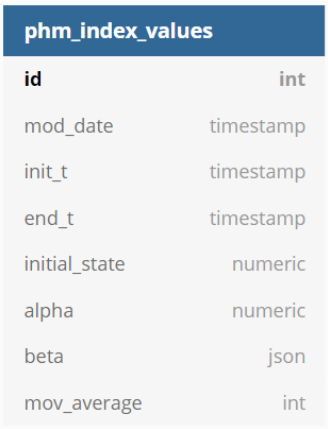


Figura 3.6: Tabla de valores para el cálculo del índice de degradación y sus campos.

3.4. Modelo Básico del Índice de Degradación

Uno de los objetivos de esta memoria es integrar a la interfaz gráfica un modelo de degradación desarrollado por otro estudiante en paralelo [1]. Para definir la

estructura básica de cálculo a partir de las señales disponibles, se desarrolló un modelo básico de este índice. El modelo básico del índice de degradación considera una parte auto-regresiva, es decir, toma valores calculados previamente (salida del sistema) y los utiliza como entrada para la nueva salida, que se combina con una media móvil de ventana variable de largo N para así filtrar los datos de mejor manera. El modelo básico del índice de degradación propuesto está dado por la expresión:

$$I_{[k]} = I_{[k-1]}\alpha + \beta \cdot S_{[k]} \tag{3.4.1}$$

en que $I_{[k]}$ es el índice de degradación, el valor α es una constante que le asigna mayor o menor peso al valor anterior, el vector $\beta \in \mathbb{R}^{1\times m}$ contiene los pesos asociados de cada señal y el vector $S_{[k]} \in \mathbb{R}^m$ contiene la media móvil de las M señales elegidas en una ventana de largo N:

$$S_{[k]} = \frac{1}{N} \sum_{i=KN-N}^{KN-1} s_i \tag{3.4.2}$$

Note que si los datos disponibles están medidos con frecuencia de un minuto, el índice de degradación se actualiza cada N minutos

3.5. Estructura Preliminar Back End

Se describe a continuación la estructura preliminar y una breve descripción de cada uno de sus módulos.

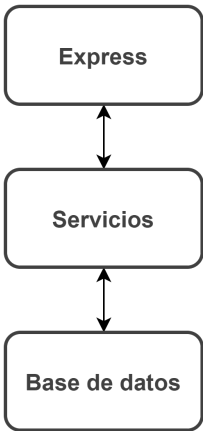


Figura 3.7: Estructura preliminar de interacción en back end

- Express: Módulo encargado de manejar las peticiones HTTP entrantes y salientes. Es el punto de acceso entre el front end y el back End. Para esta tarea se eligió Express debido a su buen rendimiento en nodeJS y a que sus características permiten desarrollar de forma rápida e incremental en caso de necesitar mas funcionalidades. Para mas información dirigirse a su sitio web [23].

- Servicios: Módulo encargado de realizar todas las operaciones necesarias tales como: procesamiento, filtrado, validaciones de datos, etc. Dentro de este se encuentran múltiples archivos con distintos objetivos cada uno.

- Base de Datos: Módulo encargado de realizar la conexión con la base de datos.

Es importante tener una separación entre la lógica del software ('Servicios' en figura 3.7) y las demás etapas, para que el código sea escalable y mas fácil de mantener. Si se dejara todo junto, cada pequeña modificación o ajuste implicaría un trabajo extenso e innecesario.

3.6. Consideraciones de Diseño del Front End

Para un diseño correcto del front end, es necesario entender los principios de la librería o *framework* con que se va a trabajar, en este caso: ReactJS [17]. De su documentación se rescatan los siguientes principios fundamentales para un desarrollo de calidad.

- **Dividir y separar cuanto sea necesario:** Para el desarrollo de un componente de React es necesario tener en cuenta que si este hace mas de una tarea, se sugiere dividirlo en mas componentes independientes con tareas únicas.
- **Reutilización de componentes:** El desarrollo de los componentes debe ir orientado a que estos sean reutilizables en la medida de lo posible, para que escalar el servicio sea más fácil y ordenado.
- **Componentes independientes:** Cualquier cambio dentro de la estructura interna del componente no debe generar cambios en los componentes con los que interactúa.

Los principios antes mencionados se ilustran en las Figuras 3.8 y 3.9. En la primera de estas se ve como todo se desarrolla dentro de un único componente, que si bien es funcional, es poco escalable y extenso. En la segunda de estas se ve como se dividió en 3 componentes: A, B y C, para así disminuir la cantidad de código, para aumentar la escalabilidad y reutilización de componentes y para hacer mas fácil el mantenimiento de estos mismos al ser independientes entre si.

Figura 3.8: Ejemplo de componente poco escalable

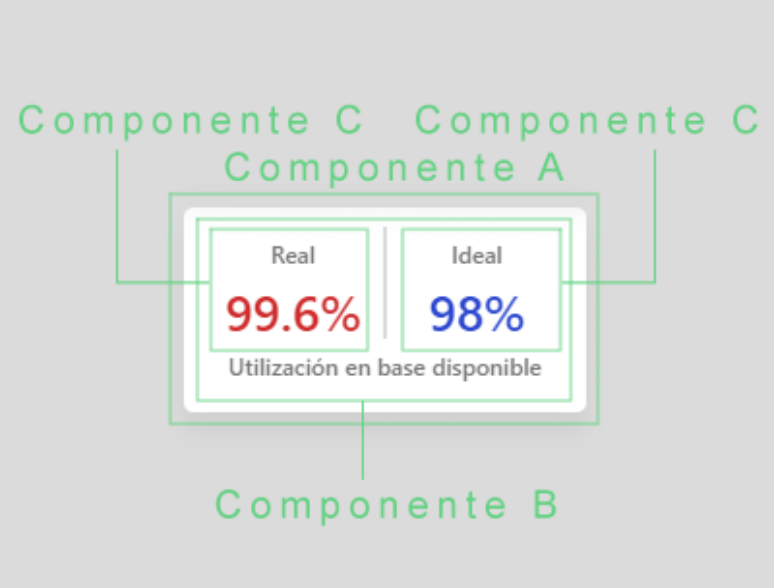


Figura 3.9: Ejemplo de componente escalable y bien dividido

En la Figura 3.10 se muestra un diagrama de como interactúan los componentes padres e hijos, entendiéndose como que uno está arriba del otro en orden jerárquico. Se puede ver cómo en el flujo de datos se tienen las *props*, que podrían contener los siguientes elementos: *callbacks*, datos de distinta índole o el estado actual del componente padre.

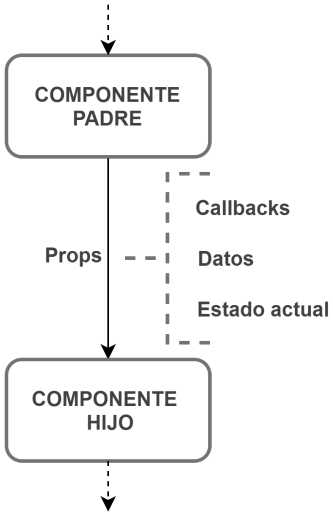


Figura 3.10: Estructura y flujo básico de datos de componentes de React

3.7. Estructura Visual Preliminar del Dashboard

Para el desarrollo del front end, es necesario tener una estructura visual o maqueta preliminar sobre cómo funcionará el dashboard y sus principales componentes. A continuación describen de forma breve los elementos del front end que se muestra en la Figura 3.11.

- Dashboard: Elemento principal que contiene y encapsula a todos los demás.
- Navegación: Barra lateral que permite navegar entre los distintos ítems del servicio. Estos se muestran en 'Contenido a mostrar'. Es posible entregar información del usuario u otra información breve en este elemento.
- Header: Elemento pequeño que se ubica en la parte superior y que incluye pocas funcionalidades, pero estas juegan un papel importante en la estética y en la experiencia del usuario.

- Contenido a Mostrar: Elemento en donde se van mostrando los distintos módulos que entregan información relevante según selección del usuario en 'Navegación'. Los módulos son descritos en la sección 4.7

Notar que los elementos mencionados previamente pueden convertirse directamente en componentes de React, donde 'Dashboard' vendría siendo el componente padre de los otros 3 ya mencionados (ver Figura 3.10): Navegación, Header y Contenido a Mostrar .

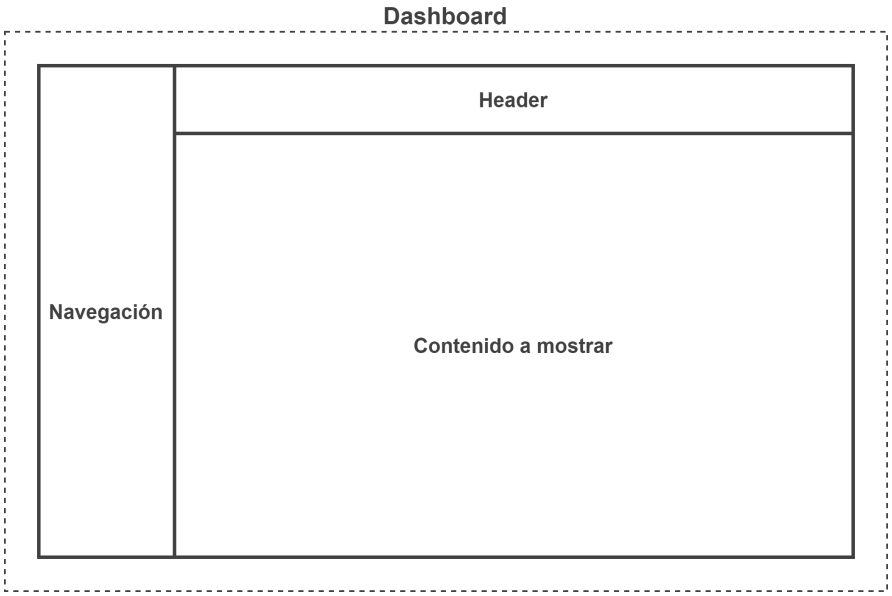


Figura 3.11: Estructura preliminar de dashboard

Capítulo 4

Desarrollo de la Interfaz Gráfica

4.1. Consideraciones en el Desarrollo del Back End

Para el desarrollo del back end de la interfaz gráfica fue necesario separar el trabajo en dos etapas. A continuación se describe cada una de estas de forma breve. Es importante destacar que para cada nueva funcionalidad o módulo se siguió el mismo procedimiento:

- **Desarrollo sin conexión:** Desarrollo de los servicios sin incluir conexión con el exterior mediante HTTP pero con manejo de archivos locales para probar funcionalidades y conexión con base de datos.
- **Desarrollo con conexión:** Integración entre los servicios desarrollados y *Express* para manejo y procesamiento de peticiones HTTP.

Debido a las funcionalidades que fue necesario implementar y para aprovechar mejor las características de *NodeJS*, el back end se dividió en dos programas. El primero de estos es el programa principal que se encarga de la recepción y procesado de archivos, entrega de datos desde el back end hacia el front end y

otras funcionalidades menores, funcionando como una API REST. El segundo de estos se desarrolló para procesar y almacenar un formato puntual de archivos CSV (Comma Separated Values) de gran tamaño y altamente demandante en tiempo de ejecución, por lo que al estar separado no bloquea el hilo único de NodeJS y no ralentiza el funcionamiento del programa principal.
En las siguientes secciones se describen los módulos desarrollados y los archivos utilizados dentro de cada uno de estos módulos para ambos programas (principal y secundario).

4.2. Módulos del Back End

A continuación se describen brevemente los módulos utilizados y las interacciones entre estos en el Back End.

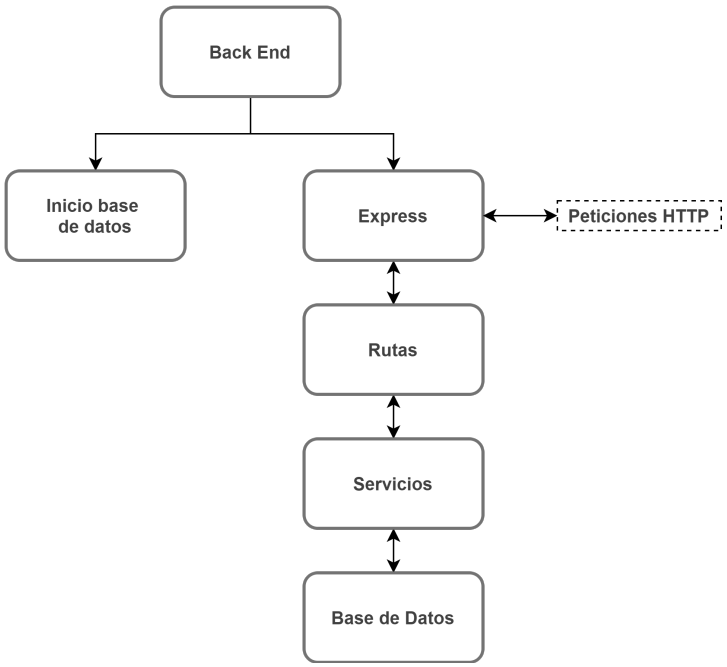


Figura 4.1: Módulos principales e interacciones del Back End

El módulo Express fue mencionado previamente en la sección 3.5. Notar que en la Figura 4.1 se añadió 'Rutas' en este nuevo diagrama como parte importante de la implementación de Express.

4.2.1. Inicio de Base de Datos

Es el Módulo encargado de la creación y configuraciones iniciales de las tablas de la base de datos. Este se realiza al inicio del programa y luego prosigue con la ejecución de este.

4.2.2. Rutas

Las rutas son el punto de acceso para el Back End mediante peticiones HTTP. Estas peticiones se realizan con una URI en particular. La URI es una dirección que indica a qué información acceder, dónde y cómo. A continuación en las Figuras 4.2 y 4.3 se muestra un ejemplo con una breve descripción.

```
1    scheme :// authority path ? query \# fragment
```

Figura 4.2: Estructura de URI

```
1    http :// codelco.test:3000 /api/files/delete ? id=76
```

Figura 4.3: Ejemplo de URI utilizado en el back end

- Scheme (esquema): proporciona información sobre el protocolo utilizado, http en este caso.

- Authority (autoridad): identifica el dominio. En el ejemplo tiene el valor de codelco.test:3000
- Path (ruta): muestra la ruta exacta al recurso. En el ejemplo tiene el valor de /api/files/delete
- Query (consulta): representa la acción de consulta. En el ejemplo tiene el valor de $id = 76$
- Fragment (fragmento): designa una parte del recurso principal. No utilizada en el ejemplo.

4.2.3. Servicios

Los servicios incluyen la lógica del programa para su correcto funcionamiento. Es importante separarlos entre si (distintos servicios) y entre las rutas y el manejo de la base de datos para lograr mayor escalabilidad e independencia en el programa.

4.2.4. Base de Datos

Módulo encargado del manejo de conexión entre el back end y la base de datos, así como la configuración de esta última.

4.3. Programa Principal del Back End

Tal como fue mencionado previamente en la sección 4.1 el programa principal es quien realiza gran parte de las operaciones requeridas para el back end tales como:

1. Procesado de archivos Excel,

2. Recepción de peticiones desde el front end, y

3. Almacenar y solicitar datos a la base de datos

A continuación se describen de forma breve los archivos utilizados en cada uno de los módulos descritos en la sección 4.2

4.3.1. Inicio de Base de Datos

- initDB.js: Archivo encargado de iniciar las tablas de la base de datos en caso de no existir. Cada vez que se inicia el programa trata de crearlas en caso que no existan. El detalle de las tablas creadas se muestra en la Figura 3.4.

4.3.2. Rutas

Dentro de esta sección, el indicador ** en las rutas de los distintos recursos hace referencia a que en esa ubicación varia el texto según lo que se quiere consultar.

- files.js: Archivo encargado de procesar las rutas de las peticiones HTTP correspondientes a manejo de archivos. La ruta de este recurso es /api/files/**.

- sensorData.js: Archivo encargado de procesar las rutas de las peticiones HTTP correspondientes a peticiones de KPI's extraídos por el programa secundario. La ruta de este recurso es /api/kpi/sensor/**

- status.js: Archivo encargado de responder el estado del Back End para revisar si está funcionando desde el exterior. La ruta de este recurso es /api/status/**

- stops.js: Archivo encargado de procesar las rutas de las peticiones HTTP correspondientes a la entrega de datos relacionados a detenciones. La ruta de este recurso es /api/stops/**.

- login.js: Archivo encargado de realizar el inicio de sesión y autenticación del usuario y en caso de ser positivo devuelve un JWT para futuras peticiones. La ruta de este recurso es /api/login/**.
- phm.js: Archivo encargado de procesar las rutas de las peticiones HTTP correspondientes a PHM e índice de degradación. La ruta de este recurso es /api/phm/**
- index.js: Archivo encargado de unir todos archivos los anteriores para conectarlo con Express y así permitir conexión con el exterior para estas rutas.

4.3.3. Servicios

- dataRequestService.js: Servicio encargado de manejar las consultas sobre las detenciones de equipos hacia la base de datos y la validación de los datos para estas mismas.
- dateTranslatorService.js: Servicio encargado de procesar fechas para adecuarlas al formato necesario según sea el uso.
- excelReaderService.js: Servicio encargado de la lectura y procesamiento de archivos Excel que contienen el detalle de las detenciones para posteriormente almacenar en la base de datos la información útil extraída de estos.
- fileReaderService.js: Servicio encargado de la lectura y manejo de archivos y carpetas.
- kpiRequestService.js: Servicio encargado de manejar las consultas hacia la base de datos y validación de los datos de estas mismas. Las consultas están relacionadas a las señales disponibles del proceso y sus datos extraídos por el programa secundario

- phmService.js: Servicio encargado del manejo, procesado y petición de datos relacionados a PHM e índice de degradación.

4.3.4. Base de Datos

- index.js: Archivo que maneja la conexión con la base de datos y la configuración de esta. Los distintos tipos de consultas a utilizar por los servicios se describen en este archivo.

4.3.5. Otros

Dentro del programa principal se encuentra el archivo auth.js que es el que contiene el *middleware* a utilizar en las peticiones HTTP. El *middleware* se ejecuta en cada ruta en la que se incluya y es el encargado de realizar una verificación de la data contenida en la petición HTTP. En caso de ser positiva la verificación se procede a la ruta de forma normal, y en caso de ser negativa se rechaza con error 401 (no autorizado). La verificación usada es un JWT dentro de los encabezados de la petición. Tal como se mencionó previamente, este JWT es generado luego de un inicio de sesión exitoso en la ruta /api/login/**

4.4. Programa Secundario del Back End

El programa secundario del back end se encarga únicamente de procesar archivos CSV relacionados a señales. Esta tarea está separada en dos etapas, a continuación se describen ambas:

1. Inicio del programa: Se configura la opción de buscar un archivo para procesar en un directorio definido (local) y llenar la base de datos con la información extraída. Esto fue pensado para poblar la base de datos al iniciar el servicio y que se realice una sola vez.

2. Durante ejecución del programa: Se deja una ruta para agregar datos de forma regular (pero con un tamaño máximo definido) desde el front end. Archivos con gran cantidad de datos deberían ser cargados al inicio del servicio, de manera de no bloquear el back end.

A continuación se describen de forma breve los archivos utilizados en cada uno de los módulos descritos en la sección 4.2

4.4.1. Inicio de Base de Datos

- initDB.js: Archivo encargado de iniciar las tablas de la base de datos en caso de no existir. Cada vez que se inicia el programa trata de crearlas en caso que no existan. El detalle de las tablas creadas se muestra en la Figura 3.5.

4.4.2. Rutas

Dentro de esta sección, el indicador ** en las rutas de los distintos recursos hace referencia a que en esa ubicación varia el texto según lo que se quiere consultar.

- sensorData.js: Archivo encargado de procesar las rutas de las peticiones HTTP correspondientes a carga de archivos CSV. La ruta para este recurso es /api2/sensordata/upload/**.

- index.js: Archivo encargado de unir todos los archivos anteriores para conectarlo con Express y así permitir conexión con el exterior para estas rutas.

4.4.3. Servicios

- csvReaderService.js: Servicio encargado de la lectura de los archivos .CSV y procesarlos para ser almacenados en la base de datos.

- excelReaderService.js: Servicio encargado de lectura y procesado de archivos de Excel para posteriormente almacenar en base de datos la información útil.

4.4.4. Base de Datos

- index.js: Archivo que maneja la conexión con la base de datos y la configuración de esta. Los distintos tipos de consultas a utilizar por los servicios se describen en este archivo.

4.4.5. Otros

Al igual que en el programa principal, se agrego una verificación de las peticiones mediante el mismo JWT y un *middleware*.

4.5. Consideraciones de Desarrollo del Front End

Para el desarrollo del front end de la interfaz gráfica se separó el trabajo en etapas incrementales. Este procedimiento se realizó para cada componente nuevo de React incluido en la interfaz. En caso de cambio de los requisitos o funcionalidades lo ideal sería repetir estas etapas de desarrollo.

1. **Revisión de requisitos:** Ver que información debe mostrar o que función debe cumplir y analizar cuales serán sus posibles entradas (*props*).

2. **Conexión con back end:** En caso de ser necesaria la petición de datos y estar implementado este recurso en el back end, se busca establecer la conexión en el componente para trabajar con datos reales. En caso de no ser posible se elaboran datos de prueba de forma manual (en el front end) con la estructura real de los datos a utilizar con tal de facilitar la interacción con el back end en un futuro.

3. **Desarrollo de funcionalidades y procesamiento de datos:** Se desarrollan los algoritmos y funciones básicas del componente para poder procesar los datos recibidos y los datos a mostrar. Se implementa una vista del componente que muestre los datos pero con un aspecto visual básico.

4. **Aplicar estilos al componente:** Con el componente ya desarrollado se busca mejorar su aspecto visual mediante los estilos de SASS [24].

En relación a la última etapa descrita en el desarrollo del front end ('Aplicar estilos al componente') se decidió utilizar SASS (extensión de CSS) para los estilos de la web y no CSS puro, esto debido a que SASS permite un manejo mucho mas ordenado, consistente y escalable de los estilos de una página web sin perder las características de CSS.

Respecto a los gráficos a utilizar en la interfaz, todos fueron desarrollados utilizando ChartJS para ReactJS [25]. Se optó por esta librería luego de analizar múltiples alternativas, esto debido a las siguientes razones:

- Aspecto visual acorde a lo que se tenia en mente para el desarrollo de esta interfaz.
- Gran comunidad activa, por lo que es fácil resolver dudas o encontrar ejemplos.
- El proyecto sigue activo y en desarrollo, por lo que constantemente se mejora y actualiza.
- La implementación de los gráficos es bastante sencilla y estos son altamente personalizables.

-

4.5.1. Paleta de Colores a Utilizar

Para que el aspecto de la aplicación web sea consistente entre todos sus componentes se establecieron colores base sobre los que se trabaja. A continuación se muestran los colores principales:

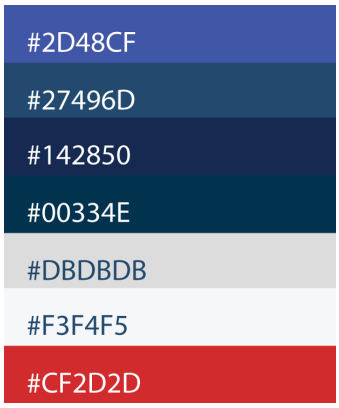


Figura 4.4: Paleta de colores principales de la interfaz gráfica

Notar que la Figura 4.4 solo muestra los colores principales, pero en el desarrollo se utilizaron variaciones de estos (mas transparentes, por ejemplo) y otros colores para situaciones puntuales dentro de los componentes desarrollados.

4.6. Desarrollo de Elementos Principales del Front End

Tal como se mostró en la Figura 3.11, la interfaz gráfica (dashboard) se compone visualmente de 3 elementos principales:

- Navegación
- Header
- Contenido a Mostrar

De estos 3 elementos, los correspondientes a 'Header' y 'Navegación' están presentes en todo momento en la interfaz, por lo que se describirán de forma separada al resto de los elementos que son cargados en 'Contenido a Mostrar'.
A continuación se muestra el desarrollo de los elementos principales y se procede a describir de forma breve qué componentes están presentes en cada uno.

Figura 4.5: Vista de elementos principales del dashboard

4.6.1. Navegación

El elemento 'Navegación' (barra lateral izquierda en Figura 4.5) se encapsuló dentro de un componente llamado 'SideNav' que está dividido a su vez en 3 componentes:

- SideNavTop: Componente que muestra la parte superior de SideNav. Contiene la imagen de CODELCO y el título 'MENU'.
- SideNavMiddle: Componente que muestra la parte intermedia de SideNav. Pertenece a la sección de texto y barras horizontales que separan SideNavTop y SideNavBottom

- SideNavBottom: Componente que se ubica debajo de SideNavMiddle. En este se cargan los distintos elementos de navegación (Inicio, Detenciones, PHM, etc.) para navegar dentro de la web mediante componentes llamados 'Item' o 'ItemDropDown' según el caso y que son descritos a continuación:

  - Item: El componente Item se utiliza para navegar a una dirección web al hacer click sobre el. Tanto la dirección, el texto y el icono mostrado (casa en el caso de Figura 4.6) son *props* del componente (haciéndolo altamente escalable)

Figura 4.6: Componente Item

  - ItemDropDown: El componente ItemDropDown se utiliza para encapsular distintos sub-items ('Visualizar' y 'Administrar' en Figura 4.7) de una categoría. Este componente tiene dos estados: desplegado y compacto. Al hacer click sobre el ítem principal ('Detenciones' en Figura 4.7) se alterna entre ambos estados mostrados en la Figura 4.7. Notar que los sub-items corresponden a instancias del componente Item, aprovechando su diseño escalable y reutilizable. Todos los iconos, textos y direcciones de navegación son recibidos como *props*.

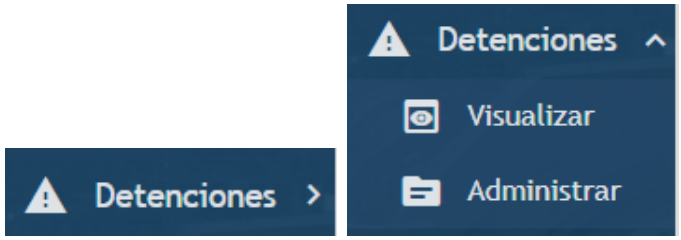


Figura 4.7: Componente ItemDropDown en ambos estados

4.6.2. Header

Componente simple compuesto de una barra horizontal y dos iconos (ver Figura 4.5). El que se sitúa a la izquierda cumple la funcionalidad de compactar o no la barra de navegación para tener mas espacio en la pantalla, mientras que el segundo botón se ubica en el lado derecho y fue pensado para el cierre de sesión (funcionalidad no implementada en este prototipo).

4.7. Módulos del Dashboard

Tal como se mencionó previamente, uno de los 3 elementos visuales principales del dashboard es el 'Contenido a Mostrar'. En este se ubica toda la información a desplegar. El desarrollo fue separado en módulos independientes entre si para mayor independencia y escalabilidad. A continuación se describen estos módulos y los principales componentes utilizados. Note que dentro de un componente pueden haber muchos otros componentes, pero por simplicidad se mencionará el componente padre que encapsula a los hijos y, en casos puntuales, se nombrarán algunos componentes hijos.

4.7.1. Módulo Inicio

Módulo en el que se cargan parámetros y gráficos relevantes. El objetivo es mostrar información de forma resumida y rápida para el usuario que utiliza la interfaz. Actualmente sólo se muestran datos de prueba pero se espera que en futuras etapas del desarrollo de este proyecto se implemente, por ejemplo, conexión en tiempo real a una base de datos de CODELCO y se actualicen constantemente los datos mostrados.

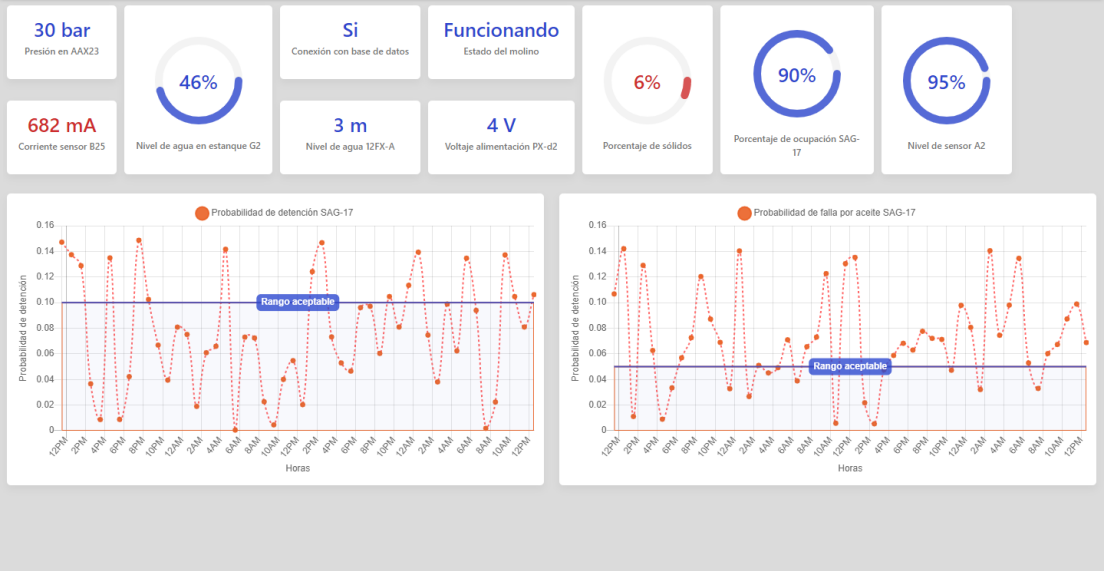


Figura 4.8: Vista del módulo de inicio

A continuación se describen los componentes principales utilizados en el módulo de inicio.

- **ShortInfo:** Componente utilizado para mostrar información de manera simple. Se le especifica como dato de entrada el texto, la unidad en caso de existir y la descripción breve de esta misma. Se le puede indicar si se ve en rojo o azul según sea necesario.

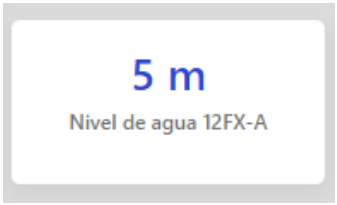


Figura 4.9: Vista del componente ShortInfo

- **ShortInfoPercentage:** Al igual que ShortInfo, se utiliza para mostrar información de manera simple pero en forma de porcentaje. Se le especifica el valor a mostrar y una descripción breve. Se le puede asignar un umbral

para que cambie de color entre azul y rojo (medición aceptable y fuera de rango respectivamente).

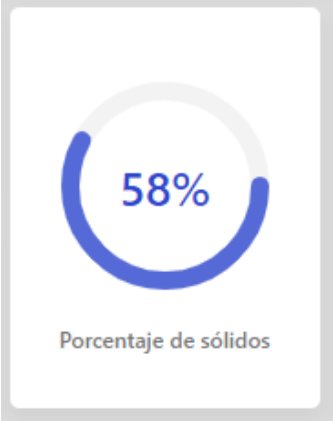


Figura 4.10: Vista del componente ShortInfoPercentage

- **StopProb:** Componente que encapsula a un gráfico con datos de prueba que muestra probabilidades en distintas horas y se establece un rango aceptable con una barra horizontal azul. Fue desarrollado para implementar en el futuro distintos indicadores de este estilo.

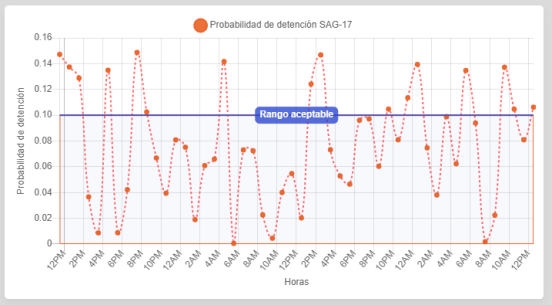


Figura 4.11: Vista del componente StopProb

4.7.2. Módulo de Visualización de Detenciones

Módulo utilizado para mostrar información relacionada a detenciones del molino SAG-17. Esta información a desplegar puede ser los registros históricos de las detenciones, indicadores importantes para el proceso (KPI) y gráficos

que entreguen otros datos. Respecto a los KPI mencionados previamente, en las primeras etapas de desarrollo de este módulo se implementaron algunos indicadores como prueba de concepto. De la retroalimentación obtenida de las reuniones con CODELCO se obtuvieron 5 KPI utilizados en el proceso y que sería gran aporte poder visualizar en este módulo. Estos indicadores fueron calculados utilizando la información extraída de los archivos Excel relacionados a detenciones provistos por CODELCO:

- Horas Disponibles
- Disponibilidad física
- Utilización en base disponible
- Coeficiente de marcha
- Disponibilidad física real

La vista de este módulo se puede ver en la Figura 4.12. Notar que en el lado derecho de la figura se muestran los KPI mencionados previamente en esta sección.

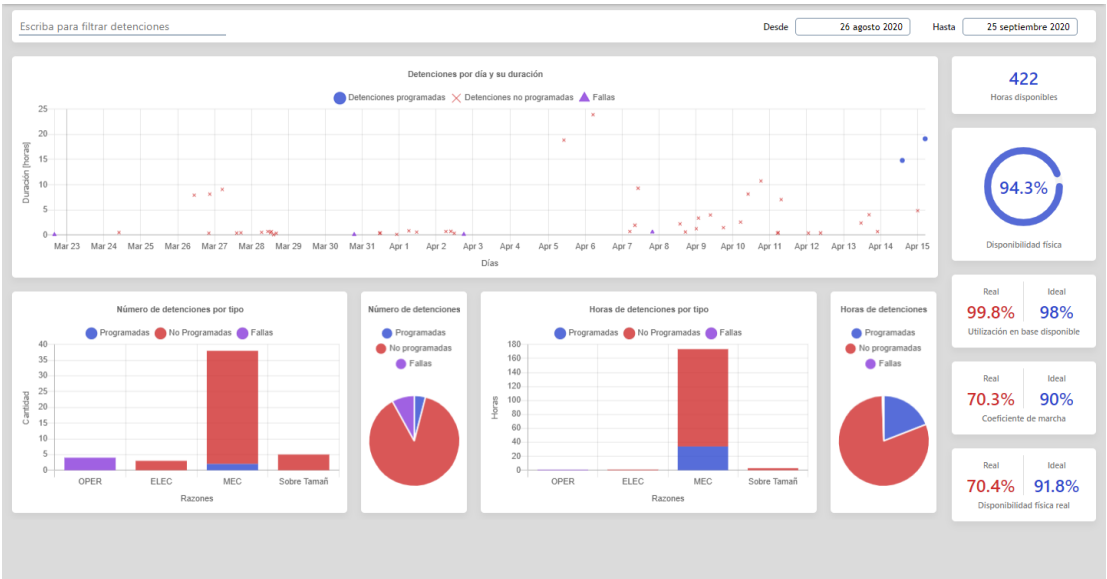


Figura 4.12: Vista del módulo de visualización de detenciones

A continuación se describen los componentes principales de este módulo, y en caso de repetirse se hará referencia a la sección en donde ya fue descrito.

- **DatePickerWrapper:** Componente utilizado para seleccionar rangos de fechas y una barra para escribir palabras que es utilizada como buscador según palabras clave en las detenciones a mostrar en este módulo. Recibe como entrada si muestra la barra buscadora y qué formato de fecha utilizar (día-hora o sólo día).

Figura 4.13: Vista del componente DatePickerWrapper

- **FailuresGraph:** Componente que encapsula un gráfico de puntos utilizado para detenciones pero funcional para cualquier otro tipo de información debido a su estructura. Al situarse sobre un punto del gráfico con el cursor se despliega información extra sobre esa detención. Recibe como entrada los distintos conjuntos de datos a mostrar (en el caso de la Figura 4.14 se muestran tres tipos de datos).

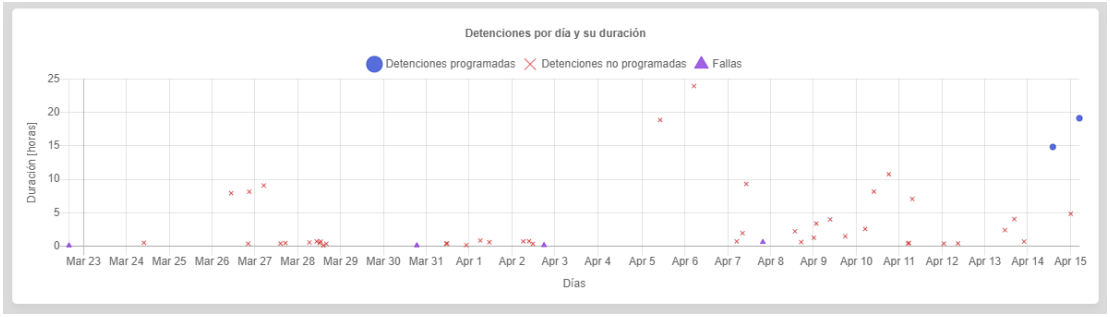


Figura 4.14: Vista del componente FailuresGraph

- **FailuresClustersGraph:** Componente que encapsula un gráfico de barras utilizado para mostrar como se distribuyen las detenciones, pero pensado para ser utilizado con cualquier tipo de datos. Recibe como entrada los

distintos conjuntos de datos a mostrar (en el caso de la 4.15 se muestran tres tipos de datos).

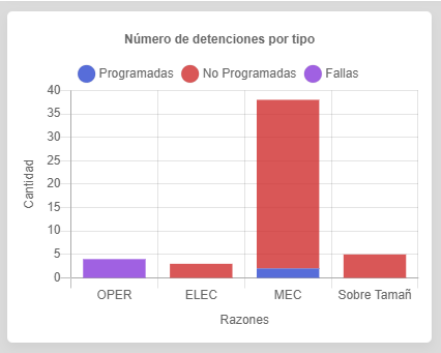


Figura 4.15: Vista del componente FailuresClustersGraph

- **PieChart:** Componente que encapsula un gráfico de torta utilizado para mostrar de forma fácil como de distribuyen las detenciones, pero pensado para ser utilizado con cualquier otro tipo de datos. Recibe los distintos conjuntos de datos a mostrar como entrada (en el caso de la Figura 4.16 se muestran tres tipos de datos).

Figura 4.16: Vista del componente PieChart

- **ShortInfo:** Ya definido en la sección 4.7.1.

- **DoubleShortInfo:** Componente de similar estructura a ShortInfo pero pensado para comparar datos contra un valor ideal. Recibe como entrada ambos valores (real e ideal), la unidad y la descripción.

Figura 4.17: Vista del componente DoubleShortInfo

- **ShortInfoPercentage:** Ya definido en la sección 4.7.1

4.7.3. Módulo de Administración de Detenciones

La finalidad de este módulo es administrar la información que se muestra en el módulo de visualización de detenciones (ver 4.7.2). Esto se lleva a cabo con dos funcionalidades: Eliminar registros de detenciones según fecha y carga de nuevas detenciones según fecha. A continuación se muestra la vista de este módulo y se describen los componentes principales dentro de este

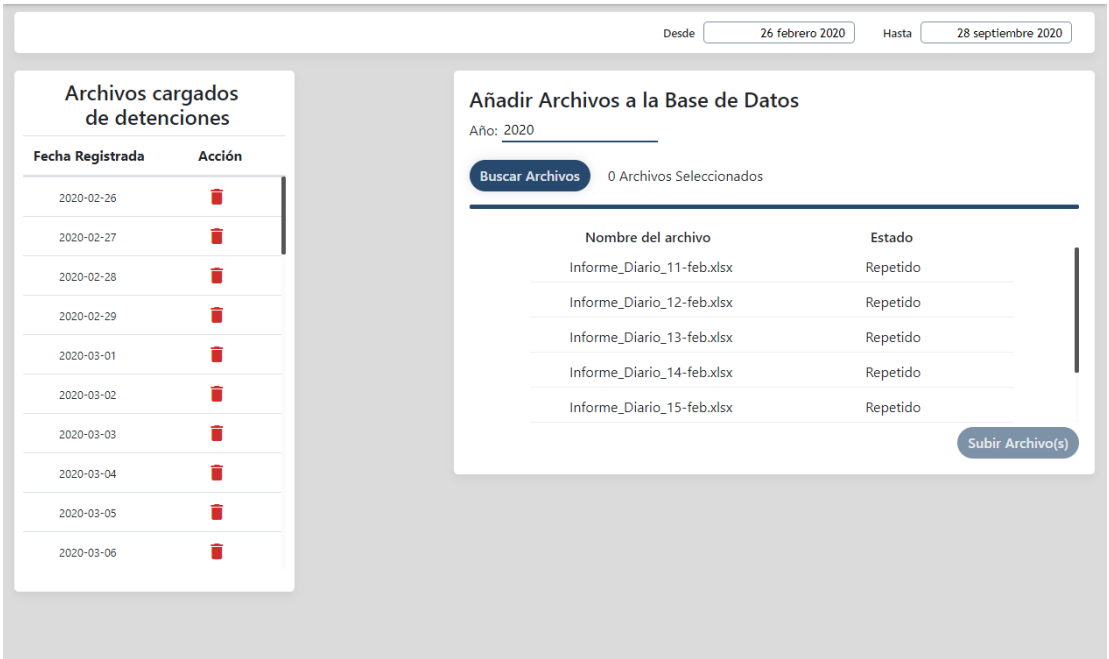


Figura 4.18: Vista del módulo de administración de detenciones

- **DatePickerWrapper:** Ya definido en la sección 4.7.2, pero se puede apreciar que al no ser necesaria la barra de buscador esta no se incluye. Esto es configurable mediante las *props* del componente.

- **CrudList:** Componente utilizado para desplegar información en forma de lista con distintos campos (columnas) y una columna de acción junto a un ícono funcional. Recibe como entradas toda la información respecto a los elementos a mostrar (títulos, iconos e información). En este caso es utilizado para eliminar archivos relacionados a detenciones.

Figura 4.19: Vista del componente CrudList

- **FileForm:** Componente utilizado para buscar archivos y enviarlos al back end para ser procesados. Posee una sección que muestra la respuesta del back end para cada archivo y así saber el resultado de la carga de estos. Recibe como entrada el tamaño máximo de los archivos, la cantidad de archivos y la URL del back end a la que se le enviaran los archivos cargados. En la siguiente figura se puede ver el componente antes de cargar los archivos y después de cargarlos respectivamente.

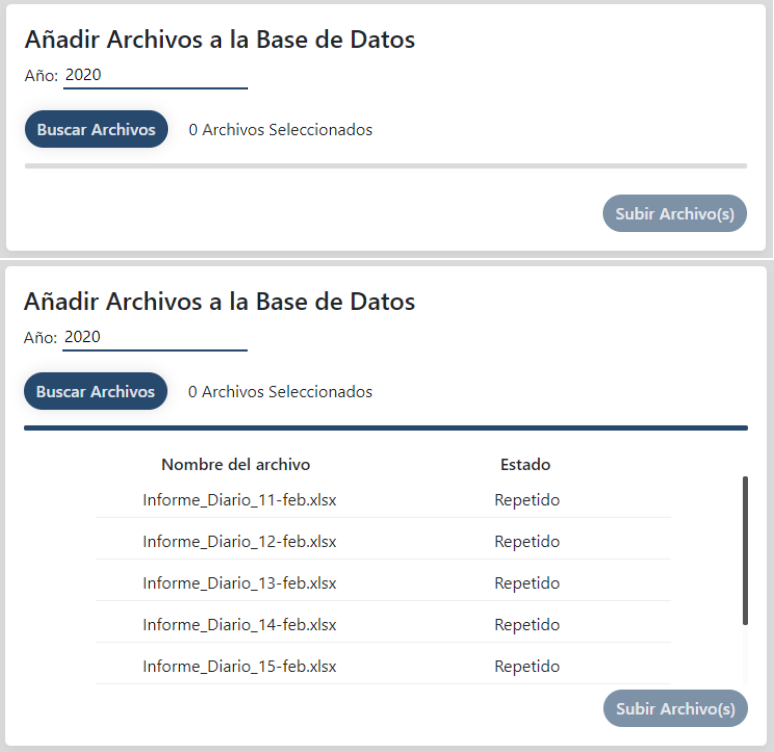


Figura 4.20: Vistas del componente FileForm

4.7.4. Módulo de Visualización de Señales

Módulo utilizado para ver los valores de distintas señales correspondientes al molino SAG-17 en una ventana de tiempo definida. Para poder comparar señales en un mismo instante de tiempo se implementaron 4 visualizadores simultáneos. A continuación se puede ver el módulo y una descripción de los componentes principales utilizados.

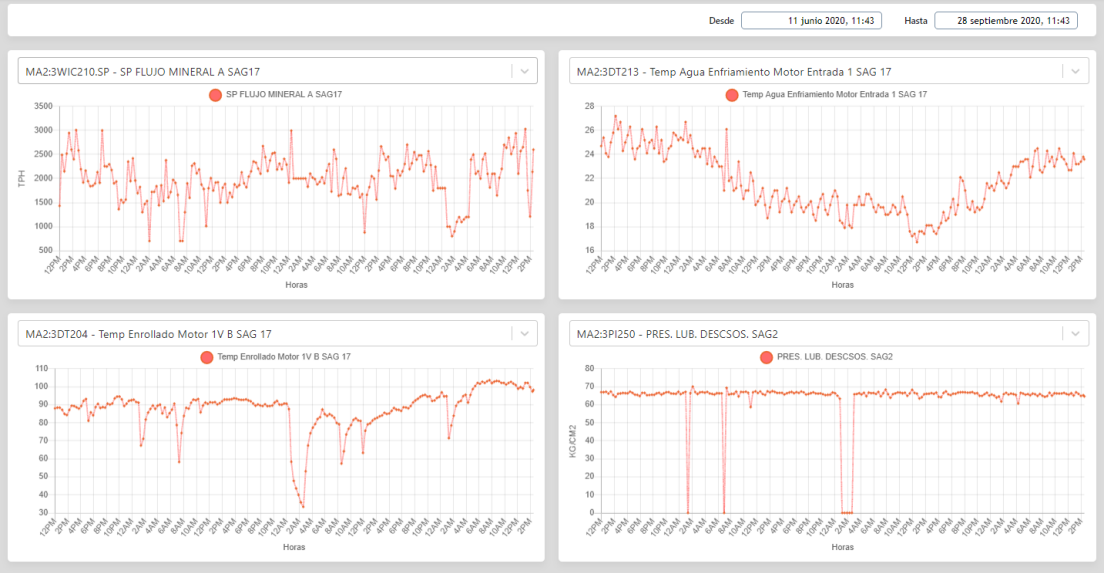


Figura 4.21: Vista del modulo visualización de señales

- **DatePickerWrapper:** Definido en la sección 4.7.2, pero esta vez utilizado en formato día-hora para mayor precisión de las señales a mostrar (el componente incluye la opción de elegir formato en las *props*).

- **SensorDataViewer:** Componente utilizado para seleccionar una señal y mostrar su valor en un determinado instante de tiempo. Para seleccionar se ocupa un componente que incluye buscador por palabras clave ('Select') y para mostrar los datos un componente que encapsula un gráfico de línea ('SensorDataGraph'). En la Figura 4.22 se presentan sus dos vistas posibles: la primera (arriba hacia abajo) corresponde al visualizador mostrando una señal seleccionada mientras que la segunda muestra la barra de búsqueda desplegada sobre el componente en caso de querer seleccionar otra señal. Este recibe como entrada la lista de señales a mostrar y al seleccionar una se consultan los datos correspondientes al back end.

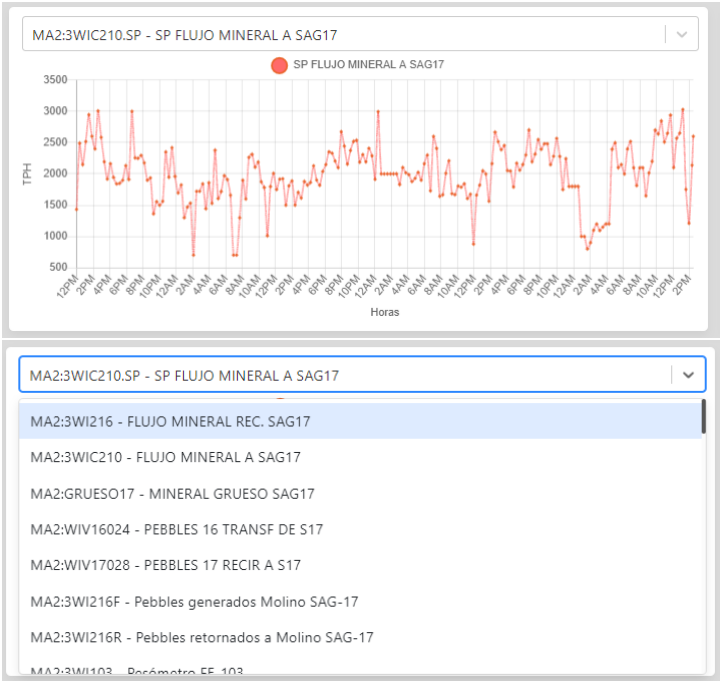


Figura 4.22: Vistas del componente SensorDataViewer

4.7.5. Módulo de Administración de Señales

Módulo pensado para administrar los archivos CSV con información respecto a las señales del proceso. Este módulo es bastante simple ya que su única finalidad es poblar la base de datos con nuevos registros en caso de ser necesario. Se establece un tamaño máximo de los archivos para que el procesado sea rápido. A continuación se muestra tanto la vista del módulo como una descripción de los componentes principales utilizados.

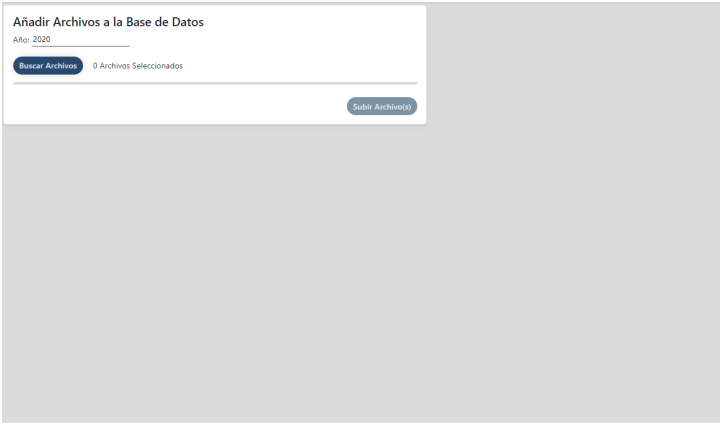


Figura 4.23: Vistas del módulo de administración de señales

- **FileForm:** Ya definido en la sección 4.7.3

4.7.6. Módulo de Visualización PHM

Este módulo muestra los indicadores relacionados a PHM. Actualmente solo está implementado un modelo básico de un índice de degradación utilizando datos históricos del molino SAG-17. Los datos y parámetros utilizados son configurados en el módulo de administración de PHM (ver la sección 4.7.7). A continuación se muestra la vista del módulo y una descripción de los componentes principales utilizados.

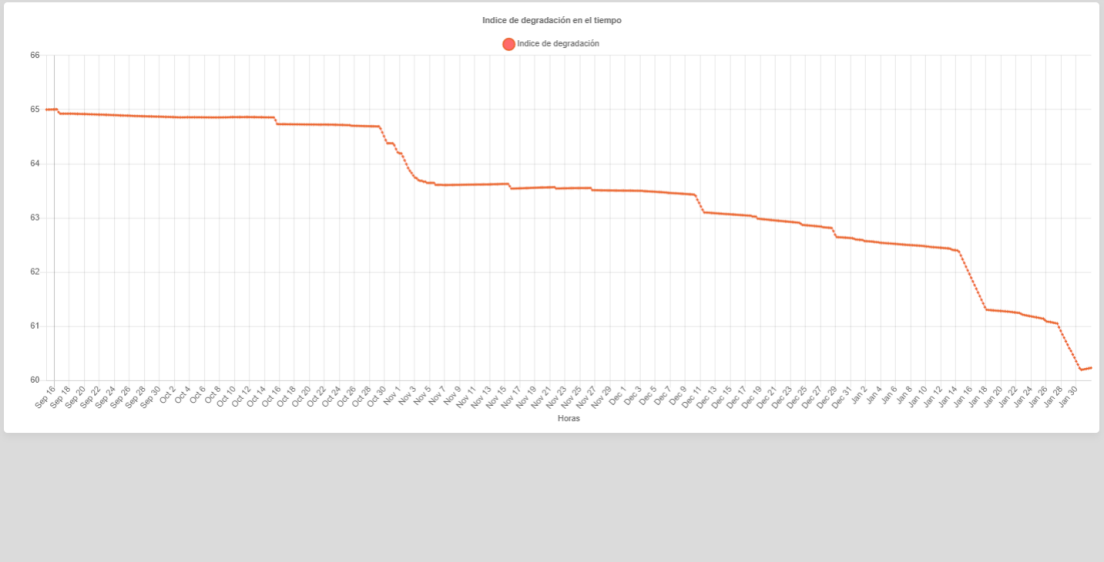


Figura 4.24: Vistas del módulo de visualización de PHM

- **SensorDataGraph:** Componente utilizado para desplegar gráficos de línea altamente configurables. Este ya se había mencionado debido a su uso dentro de SensorDataViewer (ver la sección 4.7.4). Como entrada necesita el conjunto de datos a mostrar e información relacionada a este. En la Figura 4.24 se muestra un vista del componente.

4.7.7. Módulo de Administración PHM

El módulo de administración de PHM cumple la funcionalidad de configurar los coeficientes y datos utilizados para el cálculo del modelo básico del índice de degradación mencionado en la sección 3.4 y mostrado en la sección 4.7.6. Estos son enviados hacia el back end para almacenarlos y realizar el cálculo del índice cada vez que sea solicitado por el módulo de visualización de PHM. A continuación se muestra la vista de módulo y la lista de componentes utilizados con una breve descripción.

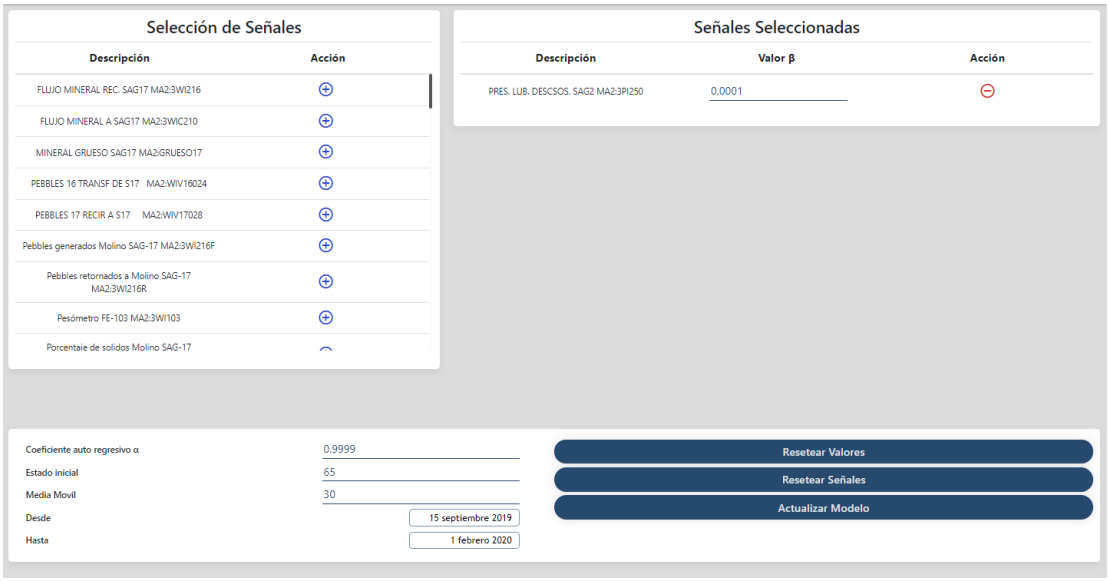


Figura 4.25: Vistas del módulo de administración de PHM

- **CrudList:** Componente ya definido en la sección 4.7.3. Es relevante mencionar que en este módulo ambas listas ('Selección de Señales' y 'Señales Seleccionadas' en Figura 4.25) son instancias de CrudList por lo que se puede ver su alto grado de configuración y diferentes usos.

- **ConfigPHM:** Componente utilizado para configurar parámetros del índice de degradación del molino tales como: coeficientes y fechas. Por otra parte tiene un conjunto de botones (lado derecho) que permite realizar distintas configuraciones y un manejo mas fácil de las señales y valores. Recibe como entrada los valores a mostrar en cada coeficiente.

Figura 4.26: Vista del componente ConfigPHM

Capítulo 5

Despliegue del Sistema y Pruebas del Prototipo

En el presente capítulo se procede a describir el despliegue del software en la plataforma Docker previamente seleccionada en el estado del arte (ver Sección 2.3). En la Figura 5.1 se muestra un diagrama que resume el despliegue del sistema completo dividido en 4 contenedores principales (ver 'Contenedor de Docker' en Glosario ante dudas)

- Contenedor 1: Contenedor que encapsula el front end del sistema, desarrollado en ReactJS.
- Contenedor 2: Contenedor que encapsula el programa principal del back end del sistema, desarrollado en NodeJS.
- Contenedor 3: Contenedor que encapsula la base de datos del sistema, desarrollada en PostgreSQL.
- Contenedor 4: Contenedor que encapsula el programa secundario del back end del sistema, también desarrollado en NodeJs

El servicio de Docker se monta sobre una máquina virtual de Ubuntu debido a que es mas simple de utilizar que sobre Windows y es mas común encontrar servidores funcionando en Ubuntu que en Windows. De esta forma la configuración queda lista para tal sistema operativo. En caso de hacer el despliegue en otro sistema operativo, no debería haber mayor problema mas que la revisión de las versiones de Docker que son compatibles en este mismo y las configuraciones para que funcione Docker. Para desplegar el sistema con las configuraciones actuales, se debe tener la versión 1.13.0 o superior de Docker.

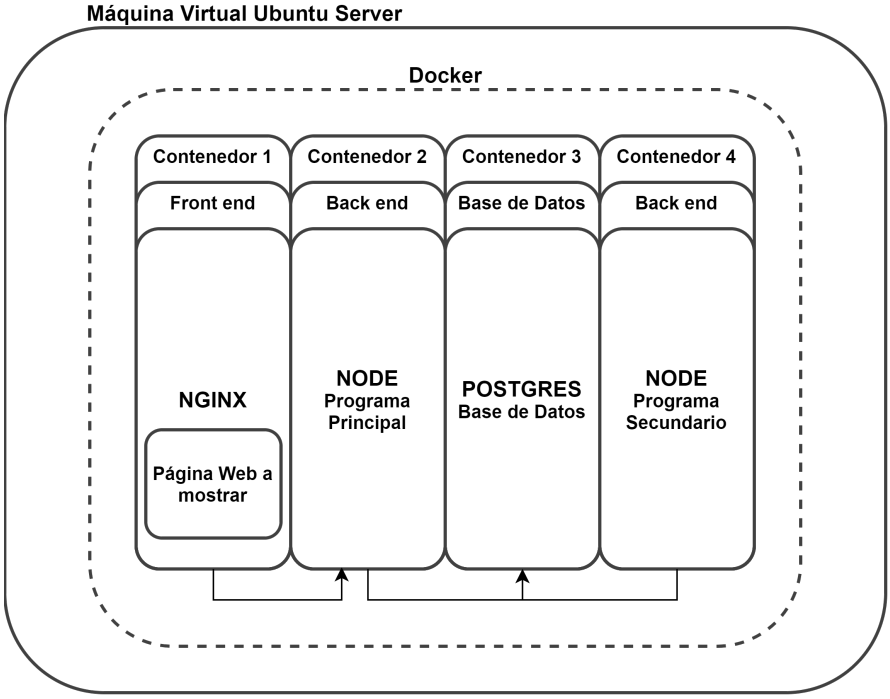


Figura 5.1: Arquitectura del despliegue del sistema en Docker

5.1. Consideraciones en el Despliegue del Sistema y Archivos Necesarios

Para el despliegue de todo el sistema, se utilizo la herramienta docker-compose para un manejo mas ordenado, escalable y simple de todos los contenedores. En

palabras sencillas se resume en un único archivo que se va a utilizar en cada contenedor. En la Figura 5.2 se pueden ver tanto los archivos (color blanco) y las carpetas (color gris) necesarias para el correcto funcionamiento de docker-compose. Las flechas negras indican contenido dentro de una carpeta mientras que las punteadas de color azul hace referencia a que contenedores se apuntan desde docker-compose. En la Figura 5.3 se muestra el archivo docker-compose utilizado.

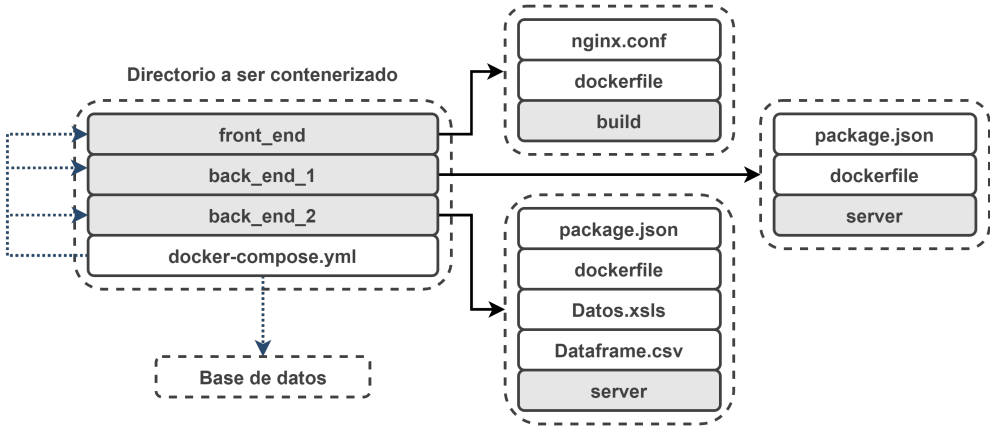


Figura 5.2: Estructura de archivos necesaria para despliegue del sistema

```yaml
version: '3'
services:
  front-end:
    build: ./front_end
    restart: always
    ports:
      - "12001:80"
    volumes:
    - "/app/node_modules"
    - ".:/app"
  back-end:
    build:
      context: ./back_end_1
      dockerfile: Dockerfile
    volumes:
      - /app/node_modules
    ports:
      - "12000:3000"
    restart: always
    depends_on:
      - database
    environment:
      DATABASE_URL: postgres://anibalwala:anibalwala@database:5432/codelco
    networks:
      - backend
  database:
    restart: always
    image: postgres:12.4-alpine
    container_name: codelco_database_postgres
    ports:
      - "12003:5432"
    environment:
      POSTGRES_USER: "username"
      POSTGRES_PASSWORD: "password"
      POSTGRES_DB: "database_name"
    volumes:
      - /srv/codelco/psql/data:/var/lib/postgresql/data
    networks:
      - backend
  back-end-csv:
    restart: always
    build:
      context: ./back_end_2
      dockerfile: dockerfile
    volumes:
      - /app/node_modules
    ports:
      - "12002:3002"
    depends_on:
      - database
    networks:
      - backend
    environment:
      DATABASE_URL: postgres://anibalwala:anibalwala@database:5432/codelco
networks:
  backend:
    driver: "bridge"
```

Figura 5.3: Archivo docker-compose.yml utilizado para despliegue

5.2. Despliegue Front End

En la Figura 5.1 se puede ver que dentro del contenedor número 1 se encapsula todo lo que es front end, pero dentro de este se tiene un servidor web NGINX y un sub-elemento denominado 'Página web a mostrar'. Ambos elementos son parte fundamental y necesaria para el funcionamiento del front end.
En las secciones a continuación se describen las consideraciones y configuraciones para el despliegue de cada una de estas.

5.2.1. Aplicación Web

Para montar y desplegar el front end se pueden tomar dos caminos:

- Servir archivos dinámicos: Los archivos a cargar en el buscador son generados por el compilador en el momento en que se accede a cada elemento. Este tipo de despliegue es comúnmente utilizado en etapas de desarrollo.
- Servir archivos estáticos: Se realiza una *build* del proyecto para obtener archivos que contienen todo el proyecto (no se generan de forma dinámica según requerimiento). Los archivos son los 3 tipos descritos en 2.2.3:
  - HTML
  - CSS
  - JavaScript

  Esto es comúnmente utilizado en producción (cuando el proyecto está listo y con acceso al público)

En este caso se optó por el uso de archivos estáticos debido a su escalabilidad, uso de recursos y velocidad de despliegue ante modificaciones. Para el funcionamiento correcto de los archivos archivos estáticos se debe editar el archivo .env en la

carpeta del proyecto previo a generarlos utilizando el comando *'npm run build'* y leyendo las indicaciones dentro de este para alternar entre producción y desarrollo.

5.2.2. Servidor Web NGINX

Para montar el servidor web NGINX de forma exitosa en este proyecto fue necesario cumplir las siguientes configuraciones (escritas en archivo nginx.conf del repositorio):

- Se debe establecer un directorio para alojar archivos estáticos (mencionados en la sección 5.2.1) dentro de la carpeta de NGINX que será contenerizada.
- Se debe configurar la URL de acceso en la que se mostrarán los archivos estáticos mencionados en el item anterior.
- Se debe configurar la dirección IP en donde es alojado el back end. De esta forma, ante eventuales modificaciones de esta dirección IP, se hace muy fácil el cambio y levantamiento del servicio.
- Se debe definir el puerto en el que se accede al servicio (12001 en la configuración actual).

En la Figura 5.4 se muestra la estructura de los archivos y carpetas necesaria para el funcionamiento.

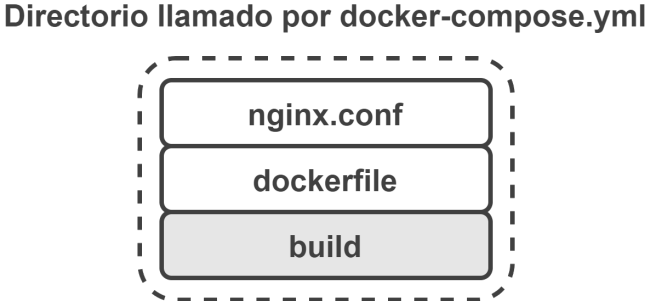


Figura 5.4: Estructura de carpetas y archivos a ser contenerizados en Docker

5.3. Despliegue Back End y Base de Datos

El despliegue del back end está dado por tres contenedores: Programa principal, Base de Datos y Programa secundario. Estos tres contenedores están conectados y dependen entre si (como se muestra en la Figura 5.1). En las siguientes secciones se muestra la configuración y estructura que deben tener las carpetas a ser desplegadas en Docker y contenerizadas.

5.3.1. Despliegue Programa Principal

Para llevar a cabo el despliegue del programa principal (descrito en la sección 4.3) se deben tomar en cuenta las siguientes configuraciones:

- Dentro del archivo .env se tiene configuración tanto para desarrollo como producción. Para el despliegue se deben eliminar (o comentar) las relacionadas a desarrollo y dejar no comentadas las relacionadas a producción. Dentro del archivo se indica de forma clara cual es cual.
- Dentro de la carpeta 'server' deben encontrarse los archivos del proyecto (ver Figura 5.5).
- Es importante tener el archivoa package.json para poder descargar e instalar en el contenedor los módulos utilizados en el desarrollo (ver Figura 5.5).

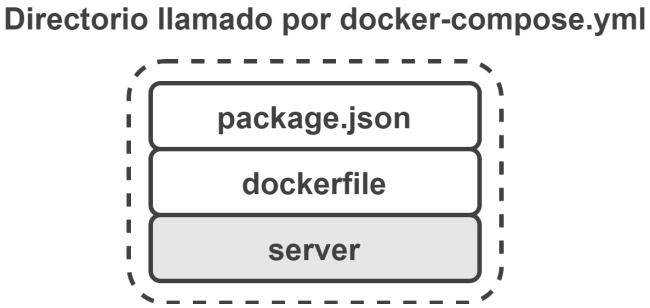


Figura 5.5: Estructura de carpetas y archivos del programa principal a ser contenerizados en Docker

5.3.2. Despliegue Base de Datos

Para el despliegue del contenedor de la base de datos, es necesario tener en cuenta lo siguiente:

- Debe definirse su configuración en el archivo docker-compose.yml utilizado en el despliegue del sistema.
- Dentro del archivo docker-compose.yml se debe establecer el nombre de usuario, contraseña y nombre de la base de datos a utilizar.
- Se debe crear un directorio en la máquina virtual (o donde se aloje Docker) en donde se almacenan todos los datos. El utilizado actualmente es /srv/codelco/psql/data.

5.3.3. Despliegue Programa Secundario

Para el despliegue del programa secundario se debe tener en cuenta las mismas consideraciones que para el despliegue del programa principal (detalladas en la sección 5.3.1) y, adicionalmente lo siguiente:

- Debe estar presente el archivo Datos.xslx (con ese nombre y extensión) en donde se especifican los tags de cada variable de dataframe.csv y una breve descripción de estos.
- En caso de querer poblar la base de datos, debe estar presente el archivo dataframe.csv. Esta instrucción se indica en el archivo dockerfile y se recomienda realizarlo una única vez. En caso de no querer hacerlo basta con no indicarlo en el archivo dockerfile.

A continuación se muestra la estructura de carpetas y archivos del programa secundario a ser contenerizado en Docker:

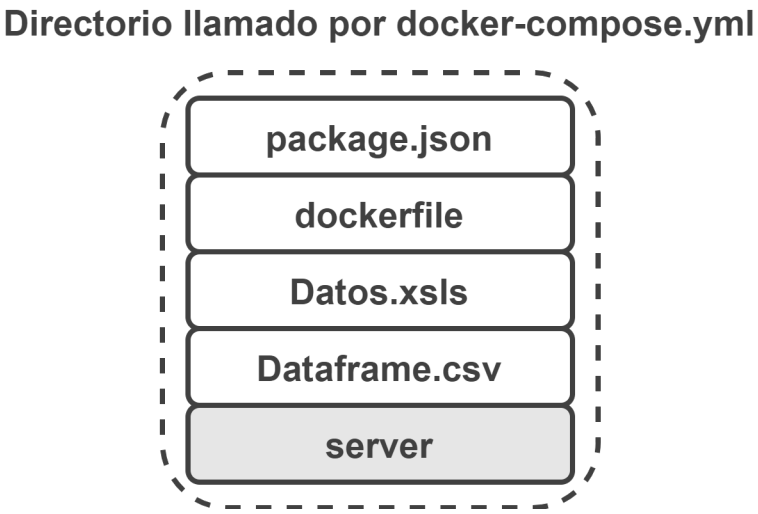


Figura 5.6: Estructura de carpetas y archivos del programa secundario a ser contenerizados en Docker

5.4. Pruebas del Prototipo

5.4.1. Contexto de las Pruebas

Entro los objetivos de este trabajo se estableció que la realización de pruebas en terreno para validación del trabajo y ajustes de este mismo. Debido a la situación sanitaria (COVID-19) el trabajo en la faena misma no fue posible, pero a lo largo del del desarrollo de la memoria se sostuvo reuniones periódicas entre AC3E y CODELCO en las que se mostraba el avance y se recibía retroalimentación sobre los módulos y funcionalidades implementadas. De esta forma la versión actual del prototipo del dashboard recoge dichas observaciones y satisface los requerimientos de CODELCO para esta etapa.

Respecto a las pruebas de funcionamiento del prototipo, se llevaron a cabo con el software funcionando en Docker. Esto permite replicar este despliegue del dashboard en cualquier ambiente que posea Docker, haciendo muy fácil el despliegue en distintos lugares fisicos.

5.4.2. Pruebas y Funcionamiento

Con el software completo funcionando en Docker se procedió en una primera instancia a comprobar que hay conexión entre front end, back end (ambos programas) y que estos pueden acceder a la base de datos. A continuación se enumeran las pruebas realizadas al sistema:

1. Acceder al dashboard desde otro equipo dentro de la misma red de internet según sea la dirección dirección IP de la máquina virtual donde está alojado.

2. Ver indicador de conexión con el back end en modulo inicio (ver Figura 4.7.1).

3. Poblar la base de datos con registros de detenciones y comprobar que estos fueron ingresados de manera correcta.

4. Poblar la base de datos con registros históricos de señales obtenidas del archivo .csv. Esta tarea es ejecutada por el programa secundario

5. Revisar desde el dashboard que la base de datos fue efectivamente poblada con los valores extraídos del archivo .csv.

6. Revisar las configuraciones dentro del módulo de PHM y el cálculo del módulo básico del índice de degradación a partir de las señales almacenadas en la base de datos.

7. Revisar los registros de cada contenedor en Docker para verificar que no hayan mensajes extraños y a ser analizados.

Es importante destacar que estas pruebas no fueron aprobadas en una primera instancia por distintos problemas. Por esto se hicieron las configuraciones y correcciones necesarias para lograr cumplir cada una de ellas y tener un prototipo funcional del dashboard desarrollado.

Capítulo 6

Conclusiones y Trabajo Futuro

En la presente memoria de titulación se presentó el diseño y desarrollo del software de la interfaz gráfica orientada al monitoreo y predicción de fallas en un molino SAG. Respecto a los objetivos iniciales planteados, en términos generales se puede establecer que se cumplieron:

1. Se desarrolló una plataforma tipo dashboard que permite monitorear las señales de un molino SAG y el análisis de las detenciones de este mismo, obteniéndose indicadores relevantes para el proceso (KPIs). El desarrollo de este dashboard fue orientado a que fuera de manejo intuitivo y la información se entregara de forma clara.

2. Se desarrollaron todos los módulos tanto en el front end como en el back end para que, a futuro, se implemente el índice de degradación que está siendo desarrollado en el trabajo de titulo "Desarrollo de un modelo de degradación de un molino SAG para mantenimiento predictivo"[1]. Junto a esto se implementó una prueba conceptual que consiste en un modelo básico de un índice de degradación en base a las señales disponibles.

3. Si bien las pruebas en terreno no pudieron ser realizadas por razones sanitarias (COVID-19), el desarrollo de esta interfaz fue validado mediante

reuniones periódicas con CODELCO, realizando modificaciones y ajustes en pos de un prototipo que cumpliera con sus necesidades y fuese validado por ellos.

Respecto al futuro de este trabajo, y específicamente, a requerimientos que pueden ser importantes de implementar para llegar a un producto comercial terminado son los siguientes:

1. Integración con la base de datos oficial de CODELCO, para trabajar con datos tanto en tiempo real como históricos, de manera de tener un seguimiento del proceso minuto a minuto

2. Extender el alcance de algunos módulos del software a otros equipos del proceso además del molino SAG-17. En algunos módulos esto ya está implementado para mayor facilidad de escalamiento del software.

3. Implementación de mayor seguridad en todo el software para evitar ataques o vulneraciones al sistema.

4. Implementación de Redux en el desarrollo del front end para un manejo mas ordenado de la aplicación web y poder abordar problemas más complejos de manera más sencilla, como un posible inicio de sesión dentro del dashboard o la necesidad de pasar datos entre módulos o componentes. Si bien esto se puede hacer sin Redux, implementarlo ayuda a tener un mejor manejo del flujo de la información y un mayor control sobre el dashboard y su comportamiento al momento programar.

5. Integración con el módulo del índice de degradación (aún en desarrollo por otro memorista) y estudiar las alternativas para esto. En particular, debe definirse si se accede a este módulo como un servicio externo al que se le consulta (desarrollado en Python por ejemplo), o si se incluye como parte

del mismo programa principal dentro del back end o si se desarrolla un tercer programa en NodeJS.

6. Utilización de herramientas ORM (*Object Relational Mapping*) para manejo mas sencillo y escalable de la base de datos. Un ejemplo de esto es Sequilize

A modo de cierre de este trabajo, creo que es importante mencionar lo relevante que es la visualización y procesamiento de datos disponibles en la industria y el migrar hacia procesos mas integrados con esto para agilizar la toma de decisiones. Estas decisiones debieran ser en base a la mayor información posible a partir de mediciones y datos. Este proyecto apunta en esa dirección y contribuye a la colaboración entre la academia y la industria. En particular en el trabajo con CODELCO se pudo apreciar la relevancia de visualizar claramente los registros de detenciones (programadas y no programadas) y su cruce con las señales del proceso. Asimismo, la integración de un índice de degradación permitirá complementar estos análisis, por ejemplo, con fines de mantenimiento predictivo de un equipo muy importante en el proceso minero como lo es un molino SAG.

Bibliografía

[1] Eduardo Grendi. Desarrollo de un modelo de degradación de un molino sag para mantenimiento predictivo, 2020.

[2] Mozaik. Sitio web mozaik. http://mozaik.rocks/, 2020.

[3] Inc Red Hat. Sitio web dashbuilder. http://dashbuilder.org/index.html, 2020.

[4] Grafana Labs. Sitio web grafana. https://grafana.com/, 2020.

[5] Metabase. Sitio web metabase. https://www.metabase.com/, 2020.

[6] MySql. Sitio web mysql. https://www.mysql.com/, 2020.

[7] PostgreSQL. Sitio web postgresql. https://www.postgresql.org/, 2020.

[8] Microsoft SQL Server. Sitio web sql server. https://www.microsoft.com/es-es/sql-server/, 2020.

[9] MongoDB. Sitio web mongodb. https://www.mongodb.com/es, 2020.

[10] C. Sitio web documentación c. https://docs.microsoft.com/en-us/dotnet/csharp/, 2020.

[11] Java. Sitio web java. https://www.java.com/es/, 2020.

[12] Node.Js. Sitio web node.js. https://nodejs.org/es/, 2020.

[13] Python. Sitio web python. https://www.python.org/, 2020.

[14] World Wide Web Schools. Sitio web w3schools. https://www.w3schools.com/, 2020.

[15] World Wide Web Consortium (W3C). Sitio web w3c. https://www.w3.org/, 2020.

[16] Angular. Sitio web angular. https://angular.io/, 2020.

[17] React. Sitio web react. https://es.reactjs.org/, 2020.

[18] Bootstrap. Sitio web bootstrap. https://getbootstrap.com/, 2020.

[19] Material. Sitio web material design. https://material.io/, 2020.

[20] VirtualBox. Sitio web oracle vm virtualbox. https://www.virtualbox.org/, 2020.

[21] Heroku. Sitio web heroku. https://www.heroku.com/, 2020.

[22] Docker. Sitio web docker. https://www.docker.com/, 2020.

[23] Express. Sitio web express. https://expressjs.com/es/, 2020.

[24] Sass. Sitio web sass. https://sass-lang.com/, 2020.

[25] ChartJS. Sitio web chartjs. https://www.chartjs.org/, 2020.