Formando lideres para la Construcción de un Nuevo País en Paz

MONITOREO REMOTO DE LOS PARÁMETROS ENERGÉTICOS DEL AEROGENERADOR AIR 40, USANDO HERRAMIENTAS WEB MODERNAS Y SISTEMAS EMBEBIDOS.

Autor
DIEGO FERNANDO ESQUIVEL RANGEL

Director
LUIS ALBERTO MUÑOZ BEDOYA

Codirector
JOSE DANIEL RAMÍREZ CORZO

INGENIERÍA ELECTRÓNICA
DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA SISTEMAS Y TELECOMUNICACIONES
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
UNIVERSIDAD DE PAMPLONA
PAMPLONA
DICIEMBRE 2021

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
DEPARTAMENTO DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA, SISTEMAS Y TELECOMUNICACIONES
PROGRAMA DE INGENIERÍA ELECTRÓNICA
TRABAJO PRESENTADO PARA OPTAR POR EL TITULO DE INGENIERO ELECTRÓNICO

TEMA:

MONITOREO REMOTO DE LOS PARÁMETROS ENERGÉTICOS DEL AEROGENERADOR AIR 40, USANDO HERRAMIENTAS WEB MODERNAS Y SISTEMAS EMBEBIDOS.

NOMBRES Y FIRMAS DE AUTORIZACIÓN PARA LA SUSTENTACIÓN:

_____
Diego Fernando Esquivel Rangel
AUTOR

_____ _____
Luis Alberto Muñoz Bedoya — **DIRECTOR**
Yesid Eugenio Santafé — **DIRECTOR DE PROGRAMA**

JURADO CALIFICADOR:

_____ _____
German Arley Portilla González
Diego Alfonso Peláez Carrillo

_____
Luis Alberto Muñoz Bedoya

PAMPLONA NORTE DE SANTANDER
COLOMBIA

DEDICATORIA.

A mis padres, mis abuelos,
y mi compañera de vida.

AGRADECIMIENTOS.

Para llegar a la realización de este trabajo de grado agradezco a Dios, por darme la vida y la salud, a mis padres, quienes han estado siempre presentes con su apoyo y amor incondicional para que pueda lograr cada meta que me he propuesto. A mis hermanos quienes me dan alegría y ánimo en cada momento de mi vida. A mi compañera de vida, quien me hace mejor persona cada día y ha estado en cada paso de este camino, quien creyó en mí. A mis abuelos paternos, quienes desde el cielo me guían, me ayudan y me dan fortaleza para continuar en el camino de la vida. Al ingeniero quien confió este proyecto a mis manos y me apoyo de inicio a fin para la buena realización del mismo.

A mis padres, Inocencio Esquivel y Laura Rangel.

A mis hermanos, Pilar Esquivel, Camilo Esquivel, Laura Esquivel.

A mi compañera de vida, Nannell Lindarte Fuentes.

A mis abuelos paternos, Pedro Esquivel, María García.

A mis abuelos maternos, Hely Rangel, Yolanda Pabuence.

A mi director, Luis Alberto Muñoz Bedoya.

A mi codirector, José Daniel Ramírez Corzo.

Tabla de contenido.

Índice de figuras.

Índice de ecuaciones.

Capítulo 1.

1.1 Resumen.

Este trabajo describe la implementación de un monitoreo remoto de las características de producción energética del aerogenerador AIR 40. La Universidad de Pamplona cuenta con un generador de esta clase que será el utilizado. Para lograrlo se diseñará una tarjeta de adquisición por la cual se obtendrán los valores de temperatura y humedad relativa del ambiente por medio del sensor DHT11, y la velocidad del viento con el uso de un anemómetro de salida analógica. También se medirá el voltaje, la corriente y la potencia producidas por el aerogenerador. Debido a la falta de pines para la lectura de variables análogas de la Raspberry, dentro de la misma tarjeta creada se utiliza el ADC MCP3008 el cual proporciona 8 canales de entrada. Esta tarjeta alimentada a 12V por medio de una fuente externa de corriente directa. Para la seguridad y protección de la tarjeta creada, se ubico dentro de una caja plástica con orificios para la conexión de los sensores, la fuente y la Raspberry Pi 3B+- Estos datos obtenidos son ingresados a una base de datos web creada por medio de Mongo DB, guardando los datos como objetos JSON. Esta base de datos ubicada físicamente en un servidor en la sede principal de la Universidad de Pamplona, por medio del sistema embebido Raspberry Pi 3B+.

Desde la página web diseñada, el usuario puede ver los valores de la producción energética en tiempo real, pero también un informe estadístico de esta producción dependiendo de las características medioambientales en ese momento, teniendo en cuenta que el voltaje nominal del aerogenerador es 12V con velocidad de viento promedio de 5.4 m/s. con ayuda de una interfaz gráfica y amigable para el usuario. Por medio de herramientas modernas para crear páginas web, como JavaScript, React, Express, SocketIO, este monitoreo se hará en tiempo real desde cualquier lugar con conexión a internet.

1.2 Planteamiento del problema.

Para contribuir con el cuidado del medio ambiente y la implementación de fuentes de energía limpia, es necesario realizar un estudio de la capacidad de producción de energía renovable en el lugar que se va a implementar el sistema de producción para determinar la viabilidad del sistema en ese lugar, y de ser viable, determinar los tiempos en los que el sistema está en óptimo funcionamiento, produciendo mayor cantidad de energía limpia.

En diferentes casos se emplean aerogeneradores con el fin de alimentar circuitos o dispositivos de forma ecológica utilizando la energía alternativa que da la fuerza del viento, pero se desconoce si esta energía generada será capaz de dar la suficiente potencia para que el dispositivo funcione de forma correcta; debido a la distancia en la que se encuentra no es posible que se realice un monitoreo constante sobre el estado del generador y la energía que está produciendo. Al momento de realizar una inversión de equipos para la generación de energía eólica en gran cantidad y de forma comercial, es necesario realizar un estudio del historial de la velocidad de viento para determinar que velocidades de viento se presentan en el lugar y en que porcentaje, para así elegir el tipo de aerogenerador a utilizar, la altura de la torre, estudiar la necesidad de un sistema de generación híbrido (energía solar y energía eólica), o descartar la posibilidad de esta producción de energía., esto por medio de un sistema de bajo costo instalado en el lugar que se quiere estudiar,

equipado con sensores para variables medioambientales y eléctricas.

1.3 Justificación.

La Universidad de Pamplona cuenta con el aerogenerador AIR 40 pero no se conoce la capacidad de producción de energía eólica real que puede generar en la sede principal de la Universidad, y debido a que un estudiante no puede estar presente junto al equipo para tomar los datos en todo momento, no es posible realizar un monitoreo constante de las variables de producción energética. Por esto se propone un monitoreo remoto para conocer las prestaciones energéticas del aerogenerador AIR 40 dadas las condiciones climatológicas del lugar para obtener el mayor rendimiento del mismo y determinar que cargas eléctricas puede generar el dispositivo a diferentes horas del día. Esto se logrará utilizando herramientas para diseñar páginas web para poder visualizar los valores ingresados a una base de datos en tiempo real, por medio del sistema embebido Raspberry Pi 3B+ que cuenta con conexión a internet.

Con un estudio del funcionamiento mediante el registro de las variables energéticas y la comparación con las variables climatológicas se pueden establecer los intervalos de tiempo, en un día, durante los cuales el generador produce mayor cantidad de energía, ya que esto depende de la velocidad del viento y otros variables atmosféricas las cuales también serán registradas. Con este historial se pueden identificar los meses en los cuales se produce mayor cantidad de energía. Se utlizará el sensor DHT11 para temperatura y humedad relativa, un anemómetro de cazoletas y un sensor de voltaje para la producción del aerogenerador.

Se obtendrán estadísticas remotas del funcionamiento energético del aerogenerador AIR 40 en una ubicación específica dadas las diversas condiciones medioambientales y buscando el aumento de la implementación de estos dispositivos de energía alternativa renovable y limpia por medio del estudio y caracterización de la capacidad de producción de energía eólica de la Universidad de Pamplona.

Se creará una aplicación web basado en MERN, Mongo DB como base de datos para utilizar el formato JSON, Express para crear el servidor, React para generar el fronted, Node para ejecutar Javascript en el servidor, estas herramientas debido a que todas funcionan con el lenguaje JavaScript, logrando crear un aplicativo entero con este lenguaje.

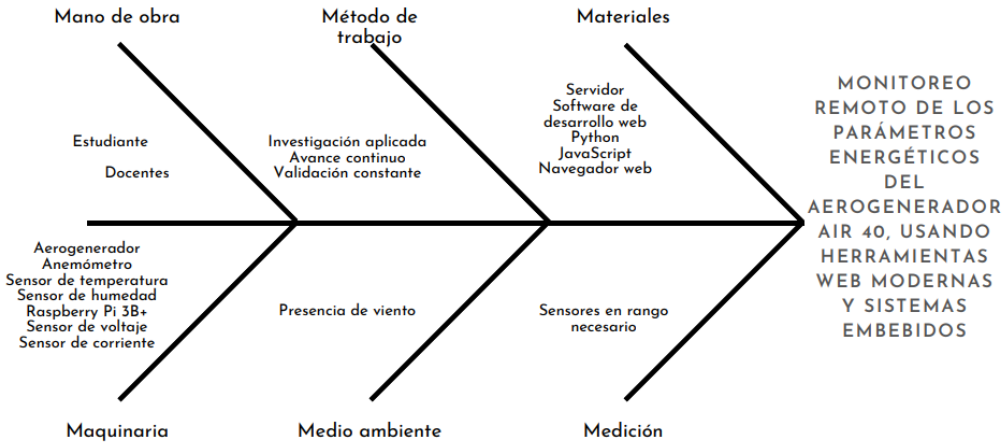

Figura 1. Diagrama causa-efecto del proyecto.

Fuente: Diagrama realizado en canva.

1.4 Delimitación.

1.4.1 Objetivo general.

Monitorear remotamente los parámetros energéticos del aerogenerador AIR 40, usando herramientas web modernas y sistemas embebidos.

1.4.2 Objetivos específicos.

- Establecer mediante un estudio los parámetros energéticos del aerogenerador AIR 40.
- Desarrollar una tarjeta de adquisición para registrar los parámetros establecidos y acoplar esta tarjeta a un sistema embebido con acceso a internet.
- Implementar sobre el sistema embebido (cliente de adquisición) los algoritmos para el registro de las señales del aerogenerador AIR 40.
- Desarrollar el aplicativo servidor con herramientas web e interfaz amigable para el monitoreo remoto de los parámetros energéticos.
- Realizar validación de la funcionalidad del sistema.

1.5 Acotaciones.

- Para la creación de la página web se usa NodeJS, Express, HTML, CSS, Bootstrap, Mongo DB, JavaScript, React y Webstorm como editor de código.
- Se tiene acceso a un servidor para alojar en él la página web.
- El funcionamiento depende en gran medida de la estabilidad y velocidad del internet en la Sede Social Villa Marina.
- Al no tener control sobre el viento presente en el ambiente, habrá intervalos de tiempo en los que el aerogenerador no producirá energía, pero serán también incluidos en los estudios.
- El funcionamiento se va a comparar con algunos sistemas de monitoreo remoto que se han desarrollado en la Universidad de Pamplona.

Capítulo 2. Estado del arte.

2.1 “AEROGENERADOR PORTÁTIL DE BICICLETA PARA BICIUSUSARIOS DE LA CIUDAD DE BOGOTÁ.”

Sánchez Fernández María Paula, Patiño Sierra Maicol, Salazar Torres María Lucía, Rico Torres Dhaily Zalenny, Betancourt Ramírez Karen Juliana.

Se identificó un problema potencial en la ciudad de Bogotá, que involucra a las personas que utilizan bicicletas, principalmente aquellos que trabajan a través de diferentes plataformas digitales haciendo entrega de domicilios quienes se ven obligados a interrumpir su trabajo mientras intentan mantener sus teléfonos cargados. En respuesta a esta demanda, se ha propuesto la implementación de un aerogenerador portátil en su bicicleta, que utiliza la energía eólica como fuente de energía alternativa, permitiendo la generación de energía y extendiendo el tiempo de carga del teléfono móvil sin detener las actividades laborales. [29]

2.2 “DISEÑO Y CONSTRUCCION DE UN AEROGENERADOR DE EJE VERTICAL PARA UN SISTEMA DE ILUMINACIÓN DE EMERGENCIA CON LUCES LED.”

Ching Valle Jonathan Xavier, Figueroa Briones Ángel Gerardo.

Se diseñó y se construyó un aerogenerador de eje vertical para cargar un banco de baterías, y que este a su vez alimente un sistema de alumbrado LED COB de emergencia en la Facultad Técnica para el Desarrollo. En el instante en que el servicio de energía eléctrica se suspenda, el sistema automatizado entra en marcha de tal manera que un contactor permite el funcionamiento de la fotocelda que detecta si es de día o de noche, si es de noche, un segundo contactor permite el paso de la energía eléctrica de la batería a los LEDs, caso contrario, si aún es de día, los LEDs no se encenderán. [30]

2.3 “DISEÑO DE UN AEROGENERADOR COMO FUENTE PRINCIPAL DE ENERGIA PARA UN CLÚSTER DE EXTRACCIÓN PETROLERA EN RUBIALES DE PUERTO GAITÁN.”

Fuentes Hernández Fabian Danilo.

Se diseñó un aerogenerador como fuente principal de energía para un clúster petrolero en Caño Rubiales. Este proyecto está enfocado al diseño de un aerogenerador para la zona de extracción de petróleos de Rubiales en Puerto Gaitán, Meta, el cual está asociado al análisis de la disponibilidad del potencial eólico en la zona; se recopiló el requerimiento de potencia que requieren los sistemas de extracción por medio de bombas PCP (Bombas de Cavidad Progresiva). Se determinaron las variables climáticas de la zona que pudieran tener relación con el funcionamiento del aerogenerador. [31]

2.4 “DISEÑO DE UN SISTEMA CON ENERGÍA LIMPIA PARA EL GIMNASIO DE LA UNIVERSIDAD DE MANIZALES.”

Rodríguez Loaiza Jehiner Andrés, Loaiza Duque Daniel Felipe,

Esta investigación se dio con el fin de diseñar un sistema de generación y almacenamiento de energía limpia para el gimnasio de la Universidad de Manizales. Se creó una solución informática

para la obtención de energía eléctrica mediante energía mecánica para la alimentación de dispositivos electrónicos, específicamente de teléfonos celulares. La Universidad de Manizales carece de fuentes de energías renovables, incluso en escenarios deportivos donde la energía "generada" por el ser humano puede ser almacenada para así ser reutilizada. Este proyecto de investigación se convierte en la primera alternativa al uso de energías renovables, en este caso, los estudiantes usuarios de gimnasios y entornos circundantes probablemente generarán suficiente energía para los teléfonos inteligentes al pedalear en bicicletas estáticas o elípticas. [32]

2.5 Diseño y simulación de un aerogenerador tripala en Boyacá, mediante dinámica de fluidos computacional.

Parra Báez Andrés.

El proyecto tuvo como objetivo diseñar y simular un aerogenerador de tres palas en funcionamiento con condiciones climáticas en la provincia de Boyacá. El proyecto inició con la historia del primer aerogenerador existente y su evolución hasta el día de hoy, descripción y función de las partes externas e internas que componen los aerogeneradores de forma global. Se investigaron diferentes variables y ecuaciones para determinar el funcionamiento de las turbinas eólicas para generar electricidad por medio de la energía cinética del viento. Se seleccionó un pueblo con buenas condiciones de velocidad de viento para después calcular el diseño del perfil aerodinámico que conforma la pala del aerogenerador. Toda la estructura se diseñó en el software Inventor y su aplicación de simulación en el software ANSYS Fluent® Dinámica de fluidos computacional (CFD). Se realizó un análisis y conclusiones con los resultados obtenidos para establecer posibles mejoras. [36]

2.6 Metodología para la determinación de características del viento y evaluación del potencial de energía eólica en Túquerres – Nariño.

Eraso Checa Francisco, Escobar Rosero Edisson, Paz Diego Fernando, Morales Carlos.

En este artículo se expone el análisis realizado al potencial de generación de energía eólica en la sabana de Túquerres, la cual está ubicada en el departamento de Nariño, Colombia. Este potencial se obtuvo por medio de la medición de la velocidad del viento entre los meses de junio y diciembre del año 2015. Los datos fueron analizados según herramientas estadísticas como la medida de tendencia central, distribución de frecuencias y distribución de Weibull para la normalización de los datos dispersos. Se calculo la densidad de potencia basado en el modelo de una turbina eólica de eje horizontal y se realizó una simulación de la curva de generación eléctrica en la zona. La velocidad de viento promedio en la zona es de 4.4 m/s y la densidad de potencia encontrada es de 3.47 W/m2.[37]

2.7 Automatización de un banco de ensayos de generadores eléctricos para aplicación de energía eólica de baja potencia.

Agotegaray Juan Carlos, Pinzón Andrea, Lera Emanuel.

Se desarrolló un banco de ensayos automatizado de generadores eléctricos, con enfoque a una aplicación en energía eólica de baja potencia y para ser utilizado en zonas urbanas. La implementación de este banco de ensayos permite probar el funcionamiento de generadores

eléctricos bajo condiciones similares a las que sería sometido si estuviera conectado como generador acoplado a una turbina eólica en entornos urbanos. Basado en lo anterior, es posible determinar los parámetros específicos de la producción del generador como son: corriente, tensión, potencia, que permiten caracterizar este generador.[38]

2.8 Análisis de la distribución espacial del potencial eólico en el territorio colombiano.

Guzmán Manrique Jhon Alexander.

Este artículo analiza la velocidad del viento basado en el modelo Weather Research and Forecasting y los datos de esta velocidad adquiridos por las estaciones de Instituto de Hidrología, Meteorología y estudios ambientales como validadores, para lograr modelar la distribución espacial del potencial de energía eólica por medio de la distribución de Weibull y los métodos de interpolación geoestadísticos. Estos resultados apoyan a instituciones como la Unidad de Planeación Minero Energética en el momento de tomar decisiones buscando brindar nuevas opciones dentro del territorio colombiano.[39]

Capítulo 3. Marco teórico.

3.1 Energía.

Es la capacidad de actuar o realizar un trabajo. La energía se conserva, no se crea ni se destruye, solo se puede transformar de una forma a otra. Para que las máquinas completen el trabajo para el que fueron diseñadas necesitan una fuente de alimentación externa. La primera fuente utilizada fue el esfuerzo muscular, pero los requisitos energéticos coincidieron con los requisitos alimenticios. Las máquinas originalmente utilizaban combustibles fósiles. Aunque el uso de máquinas aporta muchos beneficios a los seres humanos, también tiene efectos negativos. El uso extensivo de combustibles fósiles (como el carbón) emitirá al medio ambiente monóxido de carbono, dióxido de azufre, dióxido de nitrógeno y otros gases que tienen un efecto adverso sobre el medio ambiente y son dañinos para los seres humanos y los animales. Además, se emite dióxido de carbono, que es un gas de efecto invernadero y una de las posibles razones del aumento de la temperatura media en la tierra.

Para suplir las necesidades energéticas por el avance tecnológico del mundo actual se utilizan dos tipos de fuentes de energía: energía renovable y energía no renovable.[1]

3.2 Energía no renovable.

La energía no renovable se refiere a las fuentes de energía que no se renovarán o tardarán mucho tiempo en hacerlo, que se agotan cuando se usan. Entre las fuentes de energía no renovables que se utilizan actualmente, podemos distinguir dos tipos: combustibles fósiles y energía nuclear:

3.2.1 Combustibles fósiles:

Desde la revolución industrial, las fuentes de energía más utilizadas son los combustibles fósiles: carbón, petróleo y gas natural, que se forman a partir de la radiación solar. Los bosques prehistóricos son la base para la formación del carbón y este proceso duró unos 300 millones de años. De igual manera, la formación de petróleo y gas natural es producto de un proceso entre 300 y 500 millones de años. Fueron creados por formas prehistóricas de vida en océanos, que descendieron al lecho marino y se mezclaron con sedimentos y arena. El proceso de conversión de los combustibles fósiles en otra fuente de energía se logra mediante el proceso de combustión, en el cual se genera calor para producir vapor, que moverá una turbina, y finalmente, según el principio de inducción magnética, generará electricidad.

3.2.2 Energía nuclear:

Se puede obtener del núcleo de un átomo. Hay dos formas de energía nuclear: fisión nuclear y fusión nuclear. En ambos casos, el calor se utiliza para generar energía. Si desea generar electricidad, el vapor generado por el calor se utiliza para impulsar una turbina, al igual que se genera electricidad a partir de combustibles fósiles. [1]

3.3 Energía renovable.

Una de las principales ventajas de las energías renovables además de ser inagotables es que su uso no genera efectos adversos sobre el medio ambiente, por lo que se consideran energías limpias. Por ejemplo, energía eólica, energía solar, energía hidroeléctrica. Hoy en día, las energías renovables se han convertido en una realidad en nuestra sociedad y sus beneficios ambientales son notorios. [2]

3.4 Energía eólica.

El viento es el movimiento natural del aire, que se produce por la diferencia de presión provocada por diferentes calentamientos radiactivos en la superficie terrestre. De esta manera, el aire caliente se expande para crear un área de baja presión, mientras que el aire frío se comprime para crear un área de alta presión. Debido a las distintas presiones que aparecen en la atmósfera y para buscar el equilibrio atmosférico, el aire suele fluir desde un punto de alta presión a un punto de baja presión, mostrando así cambios evidentes en el tiempo y el espacio. El uso de la energía eólica para generar electricidad se realiza mediante aerogeneradores o turbinas eólicas.

Esta es una clase de energía renovable, limpia, amigable con el medio ambiente e inagotable, porque no emite gases de efecto invernadero, sus proyectos de uso se pueden replicar en diferentes lugares, pero esta energía no es acumulable y es intermitente en el tiempo.

El uso más común de la energía eólica es el golpe o la presión que ejerce el viento sobre las palas del aerogenerador, que tienen un diseño aerodinámico para utilizar una cantidad relativamente grande de energía cinética del viento. Una turbina eólica de eje horizontal consta de un conjunto de palas de rotor. Las palas de rotor utilizan la energía cinética del viento para impulsar un mecanismo de multiplicación de velocidad.

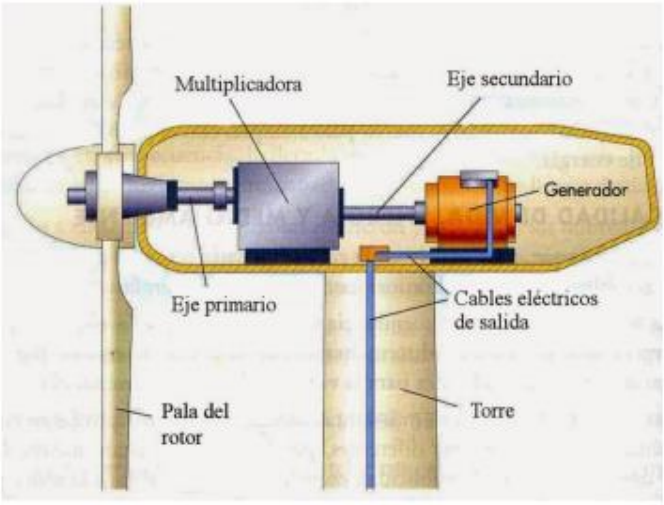

Figura 2. Aerogenerador de eje horizontal, partes.

Fuente: [3]

Este último transmite la energía cinética de rotación al generador a través de un eje de alta velocidad, que está conectado a la red de transmisión de energía mediante cableado.

Una de las ventajas de la instalación de aerogeneradores es que no contaminan el aire y si se instalan en una zona urbana en escala reducido se tiene un impacto mínimo en el medio ambiente. [3]

Esta energía es una de las fuentes de energía más económicas. Hoy en día, si se considera el costo de reparar el daño ambiental, la energía eólica puede competir con otras fuentes de energía tradicionales (como las centrales eléctricas de carbón, las centrales eléctricas de gas e incluso la

energía nuclear). La electricidad generada por el viento no produce gases tóxicos, no produce efecto invernadero, no destruye la capa de ozono y no produce lluvia ácida. No producen subproductos peligrosos ni residuos contaminantes.

Cada KWh de electricidad generada por energía eólica en lugar de fuente termoeléctrica (carbón), evita:

- 0,60 kg de CO_2, dióxido de carbono.
- 1,33 gr de SO_2, dióxido de azufre.
- 1,67 gr de NOx, óxido de nitrógeno.

La principal desventaja de la generación de energía eólica es que el viento no se puede controlar. Dado que no es una fuente de energía muy predecible, no se puede utilizar como la única fuente de generación de energía. Para evitar o mitigar caídas de tensión cuando no hay suficiente viento para producir energía eólica, se deben apoyar otros tipos de fuentes de energía. [4]

3.5 Inducción magnética.

Este es el proceso de inducir corriente a través del flujo de campo magnético de un circuito de material conductor (metal). Este fenómeno, también conocido como Ley de Faraday, fue descubierto por Michael Faraday en 1831. Es el principio básico del funcionamiento de transformadores, generadores y otros motores. Estos generadores constan de dos partes:

- El estátor, la parte estática. Actúa como inducido.
- El rotor, la parte móvil conectada al eje del generador. Actúa como inductor.

El rotor puede estar constituido por un imán permanente o más frecuentemente, por un electroimán que es un dispositivo formado por una bobina enrollada en torno a un material ferromagnético por la que se hace circular una corriente, que produce un campo magnético. El campo magnético producido por un electroimán tiene la ventaja de ser más intenso que el de uno producido por un imán permanente y además su intensidad puede regularse.

El estátor está constituido por bobinas por las que circulará la corriente. Cuando el rotor gira, el flujo del campo magnético a través del estátor varía con el tiempo, por lo que se generará una corriente eléctrica.[1]

3.6 JAVASCRIPT.

Es uno de los lenguajes de programación más utilizados en la actualidad, en conjunto con HTML y CSS dan vida al diseño de páginas web. JavaScript ya no es exclusivo solo para el fronted, sino que proyectos como NodeJS, Electron y React Native lo incluyeron al ámbito de los servidores, programas de escritorio y aplicaciones móviles. [11] JavaScript ha sido durante mucho tiempo el estándar de facto para las secuencias de comandos del lado del cliente, ya que es el único idioma compatible con todos los navegadores principales y es un lenguaje que es adecuado para casi cualquier tarea informática de propósito general.[5]

3.7 BACKEND.

Son las herramientas que utiliza el servidor para gestionar las solicitudes de información recibidas y manejar las bases de datos alojadas en este servidor. La información procesada se devolverá al dispositivo para su visualización a través de la tecnología front-end. Las tecnologías más utilizadas en el backend son PHP, Java, Python, MySQL, NodeJS, MondoDB, etc. [6]

El código escrito por los desarrolladores de back-end es lo que comunica la información de la base de datos al navegador. Cualquier cosa que no pueda ver fácilmente con el ojo, como bases de datos y servidores, es el trabajo de un desarrollador de back-end.[7]

3.8 NODE.JS.

Es un marco de E / S asíncrono sin bloqueo basado en eventos, que utiliza el motor JavaScript V8 de Google. Permite a los desarrolladores crear varias herramientas del lado del servidor y aplicaciones JavaScript en el lado del cliente y del servidor, para que puedan beneficiarse de la reutilización del código y la falta de cambio de contexto. Es multiplataforma y de código abierto. Las aplicaciones Node.js están escritas en JavaScript puro y pueden ejecutarse en el entorno Node.js de sistemas operativos como Windows y Linux. [8]

3.9 EXPRESS.

Es el framework web más popular de NodeJS que proporciona un poderoso conjunto de características para aplicaciones web y móviles. Mediante el uso de varios métodos de utilidad de middleware y HTTP, se pueden crear potentes API de forma rápida y sencilla. Express proporciona una capa delgada de funcionalidad básica de aplicación web sin ocultar la funcionalidad de Node.js. [9]

3.10 PROMESAS EN JAVASCRIPT

Son un concepto con el que se puede ejecutar código asíncrono, ya que en principio se espera que se algo se cumpla y se tienen dos consecuencias.

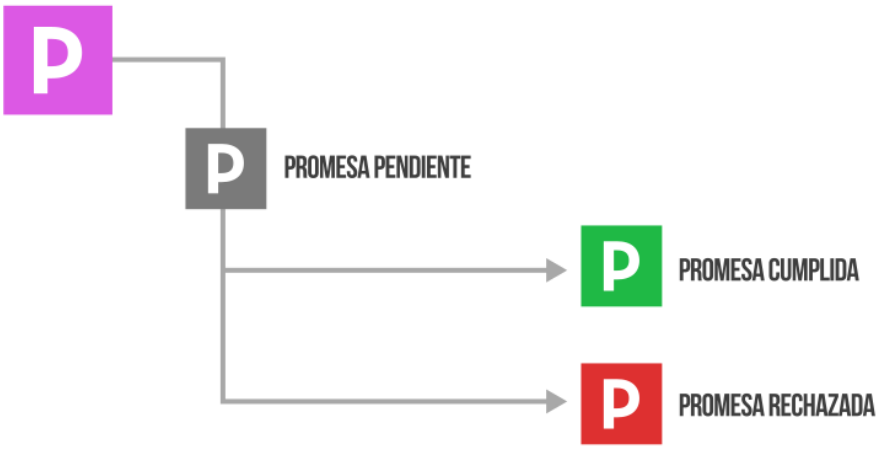

Figura 3. Estructura de una promesa en JS.

Fuente: [10]

Como se ve en la imagen anterior, se tiene una promesa, si se cumple se ejecuta un código por medio de: .then(resolve), si no se cumple se puede ejecutar un código diferente por medio de: .catch(reject), donde resolve y reject son las funciones a ejecutar para cada caso de la promesa.[10]

3.11 HTTP: Protocolo de transferencia de hipertextos.

De sus siglas en inglés *HyperText Transfer Protocol*, es un protocolo de comunicación que permite realizar peticiones de datos o recursos como por ejemplo archivos HTML. HTTP está baso en el principio cliente-servidor, las peticiones son enviadas por una entidad: un usuario o un proxy. Mayormente los clientes son navegadores web, pero pueden ser también otros programas, como un programa robot que explore la web para un buscador de internet o un script ejecuta en Python.

Cada una de estas peticiones se envía a un servidor, que es el encargado de gestionarla y enviar una respuesta.[11]

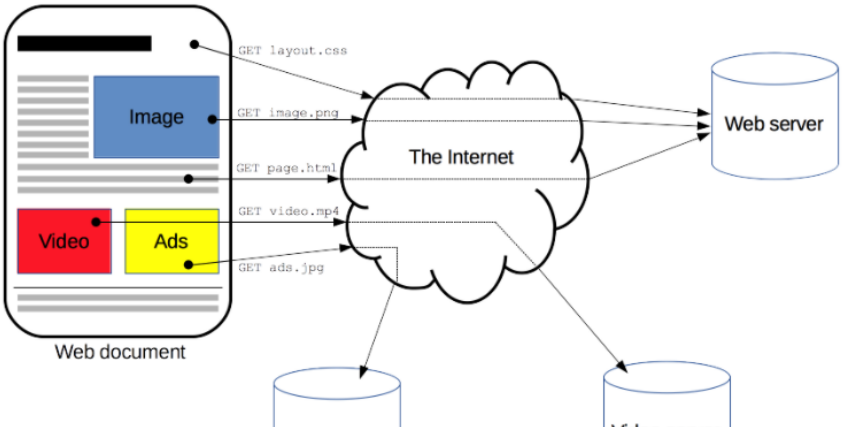

Figura 4. Estructura del funcionamiento del protocolo HTTP.

Fuente: [11]

Existe un conjunto de métodos de petición para indicar el tipo de acción que se quiere realizar, estos son llamados verbos HTTP, ya que describe en infinitivo la acción, y son:

- GET: Este método solicita la representación de un recurso. Las peticiones GET solo recuperan o reciben datos.
- POST: Este método se utiliza para enviar datos desde un cliente a un recurso, esto causa una alteración en el servidor y este emite una respuesta en consecuencia a esta petición.
- DELETE: Este método se utilizada para eliminar un recurso especifico.
- PUT: Este método se utilizada para actualizar un recurso especifico con los datos enviados en la petición.[12]

Existen otras peticiones como head, connect, pero las más utilizadas son las que se expusieron anteriormente.

3.12 BASES DE DATOS

Una base de datos es una colección de datos organizada de modo que se pueda acceder fácilmente a la información que contiene en el futuro. Una computadora es un dispositivo que le permite procesar información, ya sea en forma de texto, números, imágenes o video. Sin embargo, la computadora debe almacenar información antes de que se pueda hacer referencia a ella o cambiarla, y también debe asegurarse de que puede encontrar la información correcta en el momento correcto. La base de datos es una forma de que las computadoras resuelvan estos dos problemas.

Los datos de la base de datos deben organizarse de acuerdo con un conjunto de principios básicos, lógicamente con cierta relación o similitud entre ellos. El término modelo de datos describe la estructura lógica de una base de datos, que determina las reglas sobre cómo organizar y manipular la información que contiene. La realización del modelo de datos en una base de datos en particular se denomina esquema de base de datos. Este esquema es un modelo de una base de datos en particular, que describe información detallada sobre cómo desea implementar la base de datos, el tipo de datos requeridos u otras restricciones. El esquema de la base de datos es lo que la distingue de una lista u hoja de cálculo; utilizando un esquema, puede asegurarse de que los datos de la base de datos se organicen de acuerdo con un conjunto específico de reglas.

Un sistema de administración de bases de datos (DBMS) es un software que permite a los usuarios crear, modificar y administrar bases de datos, así como definir, almacenar, manipular y recuperar datos en estas bases de datos. Algunos ejemplos de sistemas de gestión de bases de datos incluyen MySQL, MongoDB, Oracle, FileMaker y Airtable.[13]

3.13 BASES DE DATOS SQL.

SQL son las siglas de Structured Query Language (Lenguaje de consulta estructurado). Se utiliza en bases de datos relacionales. Una base de datos SQL es una colección de tablas que almacenan un conjunto de datos estructurados específicos.

Estas bases de datos almacenan datos en tablas usando filas y columnas. Los valores de las tablas están relacionados entre sí, y las tablas también pueden estar relacionadas entre sí, de

ahí el término relacional. Esta relación hace posible acceder a los datos en varias tablas con una sola consulta.[14]

3.14 BASES DE DATOS NoSQL.

Recientemente han surgido nuevas tecnologías para satisfacer las necesidades de los servidores de bases de datos. Estas soluciones pueden manejar conjuntos de datos extremadamente grandes con velocidades extremadamente altas sin sacrificar la estabilidad o la disponibilidad.

Las bases de datos NoSQL, not only SQL (no sólo SQL), se han vuelto cada vez más populares para satisfacer estas demandas. Las bases de datos NoSQL alojan sus datos de manera diferente a las bases de datos relacionales, utilizando bases de datos basadas en JSON o de objetos clave valor para nombrar algunos de los tipos de almacenamiento comunes.[15]

3.15 MONGO DB

Es una base de datos de uso gratuito de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado. MongoDB es una base de datos distribuida en su núcleo, por lo que la alta disponibilidad, la escalabilidad horizontal y la distribución geográfica están integradas y son fáciles de usar. [6] MongoDB es una base de datos NoSQL orientada a documentos que se utiliza para el almacenamiento de datos de gran volumen. En vez de utilizar tablas y filas como en las bases de datos comunes SQL, MongoDB hace uso de colecciones y documentos. Los documentos constan de pares clave-valor que son la unidad básica de datos en MongoDB ya que utiliza notación JSON. Las colecciones contienen conjuntos de documentos y funciones que son equivalentes a las tablas de bases de datos relacionales. Esta base de datos tuvo origen a mediados de la década de los 2000.[16]

3.16 OBJETOS JSON

JavaScript Object Notation (notación de objetos de JavaScript), es un formato de intercambio de datos ligero, fácil de leer y escribir para los desarrolladores de código, y fácil de interpretar para las máquinas. JSON se convirtió es un estándar de comunicación debido a la reducción en el tamaño de los archivos y el volumen de datos que es necesario transmitir en comparación a otros estándares como XML.

Inicialmente JSON estaba ligado a JavaScript, por esto el nombre que lleva, puede decirse que está inspirado en la notación de objetos de JS, pero actualmente es un estándar de datos independiente, no está ligado a ningún lenguaje en concreto. Debido a esta independencia es necesario convertir al formato JSON un arreglo antes de utilizarlo, leerlo o guardarlo como uno.[17]

Hay dos elementos centrales en un objeto JSON: claves (Keys) y valores (Values).

- Las Keys deben ser cadenas de caracteres (strings). Como su nombre en español lo indica, estas contienen una secuencia de caracteres rodeados de comillas.

- Los Values son un tipo de datos JSON válido. Puede tener la forma de un arreglo (array), objeto, cadena (string), booleano, número o nulo.

Un objeto JSON comienza y termina con llaves {}. Puede tener dos o más pares de claves/valor dentro, con una coma para separarlos. Así mismo, cada key es seguida por dos puntos para distinguirla del valor.[18]

```
[
    {
        "id":1,
        "titulo":"titulo de la entrada",
        "autor":"autor de la entrada"
    },
    {
        "id":2,
        "titulo":"titulo de la segunda entrada",
        "autor":"autor de la segunda entrada"
    }
]
```

Figura 5. Estructura de un objeto JSON.

Fuente: Autor.

3.17 WEBSOCKETS.

Es un estándar utilizado en comunicación web bidireccional en tiempo real entre el servidor y los clientes. Estos websockets se aplican generalmente donde el tiempo de respuesta debe ser muy corto. La implementación de este estándar permite reducir la latencia durante las conexiones ya que tiene menor carga en los servidores permitiendo más conexiones simultaneas en los equipos.

La comunicación por medio de websockets es más rápida que con HTTP, y al no utilizar este último paquete se reduce el uso de la red. Websockets presenta mayor escalabilidad en la web y ayuda a mantener conexiones de forma continua con los servidores.[19]

3.18 SOCKET.IO

Es una dependencia que permite la implementar comunicación en tiempo real. Es bidireccional y está basada en eventos generados entre el navegador (cliente) y el servidor. Aunque socket.io utiliza WebSocket para transportar información, no es una implementación de este último, ya que agrega metadatos adicionales a cada paquete de datos. Por esto un cliente WebSocket no puede conectarse a un servidor Socket.io ni viceversa.

Socket.io ofrece confiabilidad, reconexión automática al servidor, transmisión a todos los clientes o segmentación por salas y multiplexación.[20]

3.19 FRONTEND.

Son herramientas que se utilizan en el lado del cliente para diferentes dispositivos conectados al servidor a través de Internet. Estas tecnologías y lenguajes de programación se implementan en diferentes navegadores web existentes, que son intérpretes de estos códigos. Las tecnologías más utilizadas en el front-end son HTML, CSS, JavaScript, React, jQuery, Ajax, Bootstrap, Angular, etc. [21]

El desarrollo de front-end gestiona todo lo que los usuarios ven en su navegador o aplicación. Los desarrolladores de front-end son responsables de la apariencia de un sitio. Para esta tarea se requiere tecnología como HTML5 para manejo del contenido, CSS para crear el 3 diseño, maquetación y la parte visual de la aplicación y el lenguaje de programación Javascript para usar algunos frameworks o librerías que expanden capacidades de interacción de las interfaces y consumir los servicios proporcionados por el backend.[7]

3.20 HTML: Lenguaje de Marcado de Hipertexto.

La definición errónea que se tiene de HTML es que es un lenguaje de programación, pero no, este es un lenguaje de marcado de hipertexto, por sus siglas en inglés *HyperText Markup Language*, que define la estructura de un contenido. Este lenguaje consiste en una serie de elementos que se utilizan para encerrar diferentes partes de un contenido logrando que se vean y/o se comporten de formas específicas.

Las principales partes de un elemento HTML son:

- Etiqueta de apertura, define donde empieza a tener efecto el elemento, consiste en el nombre del elemento que se quiere encerrado entre paréntesis angulares de apertura y cierre.
- Etiqueta de cierre, define donde termina el efecto del elemento, se representa igual que la etiqueta de apertura, pero con una barra antes del nombre de la etiqueta.
- Contenido, es lo que va a obtener el efecto de la etiqueta, puede ser texto, imágenes, otras etiquetas, etc.
- Elemento, es la equivalencia de la etiqueta de apertura, más la etiqueta de cierre, más el contenido.

Figura 6. Partes de un elemento HTML.

Fuente: [22]

Los elementos pueden tener atributos, es información adicional acerca del elemento, pero esta no será impresa dentro del contenido. Comúnmente se utiliza el atributo *class* parar darle al elemento un nombre identificativo, con el cual se puede referir a él desde otro archivo, por ejemplo, para darle estilos desde un documento de CSS.[22]

Figura 7. Atributos en un elemento HTML.

Fuente: [22]

3.21 CSS: Hojas de estilo en cascada.

De sus siglas en inglés *Cascade StyleSheet*, hace referencia al conjunto de instrucciones que definen la apariencia de diferentes elementos HTML dentro de una página web. Si se crean los estilos en un archivo separado, se pueden diferenciar claramente los estilos del contenido de la página. CSS proporciona estilos como tamaño, color, bordes, fondo, etc., a los elementos HTML presentes en la página web.[23]

En el archivo HTML se asigna un atributo por medio de class y, en el archivo principal de CSS (.css), se refiere a él colocando entre llaves los estilos que se quieren dar.

3.22 REACT.

Es una librería de JavaScript de código abierto mantenida por Facebook que se puede usar para crear vistas renderizadas en HTML, interfaces de usuario. Fue creada por Jordan Walke. Se desplegó por primera en Facebook en 2011 y en Instagram en 2012. Esta librería permite crear interfaces de usuario interactivas, con la posibilidad de cambiar la información mostrada sin que sea necesario recargar la página por completo. Esto dio origen al concepto de Componente, los cuales permiten separar u organizar la interfaz de usuario en partes independientes, reutilizables y analizar cada pieza de forma en independiente, su estructura y lógica. Un botón, una tabla, una imagen, pueden ser componentes, y el llamado de estos puede hacerse dentro de otro componente.[24]

3.23 CREATE REACT APP.

Existen diferentes maneras para crear un proyecto con ReactJS, se deben instalar, desde Node con npm, cada una de las dependencias necesarias como Babel, Webpack, etc, esto requiere un conocimiento grande en cuanto a configuraciones de cada dependencia y comunicación entre archivos.

Create-react-app es una forma más sencilla de poder crear proyectos preconfigurados y listos para usarse, omitiendo de esta manera la configuración inicial y permitiendo trabajar de inmediato en el

proyecto que se quiere crear. Al ejecutar: *npm install -g create-react-app* en la terminal o centro de comando, se instala de manera global el paquete créate-react-app.[25]

3.24 Axios.

Es una biblioteca basada en promesas para hacer peticiones HTTP desde un cliente o desde node.js. Axios está basada en XMLHttpRequest, que se puede utilizar en un navegador o en un servidor con Node.js. Axios funciona de forma asíncrona, lo que permite recibir un objeto JSON como respuesta a la petición a la API REST. Es uno de los clientes basados en promesas más populares, es simple, liviano y muy fácil de personalizar.

React no cuenta con una forma propia de consumir una API REST, por esto es necesario utilizar una librería externa como axios, apollo client, etc. [33]

3.25 Chart JS.

Es una de las bibliotecas para visualización de datos más populares, completa y sencilla de utilizar. Chart JS proporciona una visualización interactiva de sus datos con una codificación mínima. Los datos deben ser cargados en un objeto estándar de JavaScript dentro del objeto Chart, después se pueden agregar estilos y otras configuraciones usando una estructura declarativa simple basada en objetos. Esta librería es una excelente opción para comenzar a crear visualizaciones de datos para la web.[34]

3.26 RASPBERRY PI 3B+.

Raspberry Pi es una computadora de bajo costo con un tamaño compacto (llegando al tamaño de una tarjeta de crédito). Raspberry Pi 3B+ posee un Procesador de cuatro núcleos a 1,4 GHz y 64 bits, LAN inalámbrica de doble banda, Bluetooth 4.2 / BLE, Ethernet más rápida y compatibilidad con alimentación por Ethernet (con PoE HAT independiente).[26]

Dentro de este sistema embebido se pueden encontrar pines GPIO (General Purpose Input Output, entradas y salidas de propósito general), que son una serie de pines que se utilizan como entradas y salidas digitales. Estos pines están incluidos en todos los modelos de Raspberry.[27]

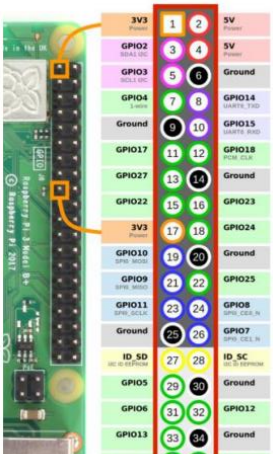

Figura 8. Pines de la Raspberry Pi 3B+.

Fuente: [27]

3.27 PYTHON

Es un lenguaje de programación de alto nivel, multiparadigma ya que admite programación imperativa, programación orientada a objetos y programación funcional. Python fue creado por Guido van Rossum de Stichting Mathematisch Centrum en los Países Bajos a principios de la década de 1990 y es el sucesor del lenguaje ABC. [28]

Al utilizar Python en una Raspberry tenemos la posibilidad de conectar el mundo digital con el mundo real por medio de los pines GPIO de este sistema embebido.

Python permite la visualización de datos avanzada e incluso interactiva utilizando bibliotecas o librerias propias. Una de las bibliotecas básicas para la visualización de datos es Matplotlib. Aunque es una de las bibliotecas más básicas, también es la más utilizada en proyectos que necesitan realizar una visualización más avanzada, porque es muy eficiente y contiene una variedad de tipos de gráficos compatibles, desde gráficos de líneas hasta gráficos de puntos o mapas de calor.[35]

Capítulo 4. Metodología.

Durante el desarrollo de la aplicación se utilizó el sistema operativo Ubuntu 20.04 LTS, distribución de Linux, en el computador en el cual se hizo la programación de la aplicación web; debido a que la mayoría de servidores donde se puede hacer despliegue de aplicaciones son Linux y su terminal funciona con los mismos comandos que la del sistema operativo instalado en la Raspberry, ya que ambos sistemas son distribuciones de Linux. Como editor de código se utilizó Webstorm 2021.1 el cual proporciona una licencia gratuita para estudiantes y un ambiente de programación ordenado y amigable.

Las aplicaciones web están conformadas por el Fronted y el Backend, aunque en ambas se utiliza JavaScript, se crearon en carpetas diferentes para tener un mayor orden y mejor control de los archivos. Se desarrollaron en servidores separados, en diferentes terminales se mantiene la ejecución de cada servidor, pero comunicados entre sí por el protocolo HTTP y Socket.io, obteniendo valores de una base de datos de Mongo DB alimentada desde una Raspberry Pi 3B+.

4.1 Adecuación del sistema embebido Raspberry Pi 3B+.

El sistema embebido Raspberry ha tenido 4 versiones desde su creación en el año 2012, la versión que se utilizó en este proyecto es la Raspberry Pi 3B+ ya que esta cuenta con una conexión a internet WIFI, necesaria para realizar el monitoreo remoto por medio del envío de datos a un servidor web.

Figura 9. Raspberry Pi 3B+, carcasa y ventilador.

Fuente:
https://www.aliexpress.com/item/32945416376.html?spm=a2g0s.9042311.0.0.27424c4d7ihKak

Para mayor protección, el modelo adquirido de Raspberry consta de una carcasa de pasta que protege de polvo y golpes a la placa, como se ve en la figura anterior; integra también un ventilador que se conecta a los pines 4 y 6 de la Raspberry y es activado desde el momento en el que se enciende la placa.

El sistema operativo recomendado para instalar en una Raspberry es Raspbian Pi OS, este es una distribución de GNU/Linux basado en Debian. Por medio de una herramienta oficial se instala este sistema operativo en una memoria micro SD y se inserta en la Raspberry, al conectar la alimentación de 5V el sistema enciende de forma completa. Esta herramienta instala el sistema operativo y algunos programas complementarios como por ejemplo navegador web, visor de imágenes, reproductor de audio y video, thonny python, etc.

```
pi@raspberrypi:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
```

Figura 10. Versión de Raspbian instalada.

Fuente: Terminal de Raspbian.

Por medio de python se administra el uso de los pines GPIO de la Raspberry. Por defecto está instalado Thonny Python que es un entorno de desarrollo integrado (IDE) de python 3 que facilita aprender y enseñar programación en este software.

4.2 Sensado de temperatura y humedad.

Para medir estas variables se utilizó el sensor de humedad relativa y temperatura DHT11, el cual proporciona una salida digital y se conecta mediante un solo cable para recibir los dos datos, haciendo mas fácil su lectura por medio de los pines GPIO de la Raspberry Pi 3B+. Este sensor puede ser alimentado desde 3.5V a 5V, con un rango de medida de temperatura de 0°C a 50°C y precisión de ±2°C, y para la humedad, consta de un rango de 20% a 90% RH (humedad relativa) con precisión de ±5% RH.

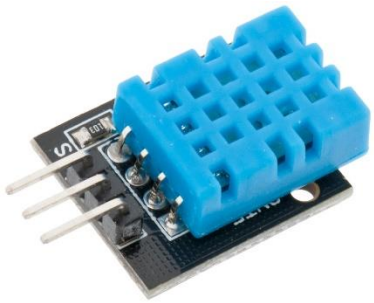

Figura 11. Sensor DHT11

Fuente: https://www.hwlibre.com/dht11/

El sensor consta de tres pines como se muestra en la figura anterior, de izquierda a derecha son, señal de salida, voltaje positivo y tierra. Debido a que el sensor utilizado está en una pequeña PCB y ya cuenta con una resistencia de pull-up entre la señal y el voltaje positivo, no es necesario colocar otra resistencia en paralelo entre estos dos pines. +. El sensor DHT11 para temperatura y humedad, comunica estos valores directamente a un pin GPIO de la Raspberry.

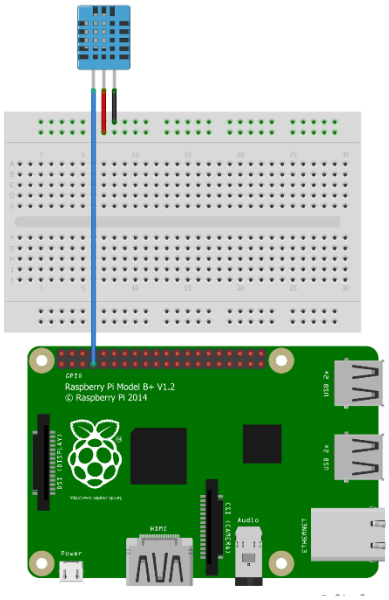

Figura 12. Esquema de circuito para conexión del sensor DHT11 con Raspberry PI 3B+ realizado en fritzing.

Fuente: Autor.

Gracias al tipo de salida del sensor, se pudo conectar directamente con el pin GPIO 4 de la Raspberry Pi 3B+. Para realizar la lectura de este sensor se utilizó la librería de python *Adafruit_Python_DHT*, que se clonó desde un repositorio en git. Estos son los pasos que se realizaron en el terminal de Raspbian para poder obtener esta librería:

- *git clone https://github.com/adafruit/Adafruit_Python_DHT.git*
- *cd Adafruit_Python_DHT/*
- *sudo python3 setup.py install*

Para visualizar los valores de humedad y temperatura en python shell se utilizó el programa del anexo 1, el cual dio como resultado los valores de temperatura y humedad guardadas en una variable con el mismo nombre.

dht11.py

```
import Adafruit_DHT

sensor = Adafruit_DHT.DHT11

pin = 4

while True:
    humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)
    print('Temp={0:0.1f}*C Humedad={1:0.1f}%'.format(temperatura, humedad))

```

Shell

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run dht11.py

  Temp=22.0*C Humedad=65.0%
  Temp=22.0*C Humedad=65.0%
  Temp=22.0*C Humedad=65.0%
  Temp=22.0*C Humedad=65.0%
```

Figura 13. Valores de temperatura y humedad medidas por el sensor DHT11 en Python.

Fuente: Autor.

Este sensor permitió determinar variables del entorno medioambiental de manera sencilla y precisa, con un rango de medida acorde con lo que se necesitó para este proyecto.

4.3 Sensado de velocidad del viento.

Para determinar la velocidad del viento se utilizó un anemómetro el cual es un sensor que mide esta variable. El instrumento posee salida de voltaje análogo proporcional a la velocidad del viento, este voltaje varía desde 0.4 V (vientos de 0 m/s) hasta 2.0 V (para una velocidad del viento de 32,4 m/s).

Figura 14. Anemómetro con salida de voltaje análogo.

Fuente:https://www.didacticaselectronicas.com/index.php/sensores/ambientales/anem%C3%B3metro-con-salida-anal%C3%B3gica-de-voltaje-adaf-1733-sensores-medidores-de-velocidad-viento-aspas-anemometros-adafruit-detail

Este sensor del fabricante ADAFRUIT, tiene un rango de voltaje de alimentación de 7V a 24V y mide velocidades de 0 m/s hasta 32.4 m/s con una resolución de 0.1m/s. En este proyecto su voltaje de entrada fue de 12V DC.

Se utiliza un anemómetro con salida analógica con límite máximo de 32.4 m/s, debido a la capacidad de acoplar su salida por medio de un ADC a la Raspberry Pi 3B.

Debido a la falta de pines para lectura de variables análogas de la Raspberry pi 3B+ fue necesario implementar el convertidor análogo-digital (ADC) MCP3008 para realizar la lectura del valor del voltaje producido por el anemómetro. Se utilizó el primero de sus ocho canales de conversión. Este conversor funciona a 5V y tiene una resolución de 10 bits. Se hizo la conversión interna para manejar el valor en voltios dentro del programa.

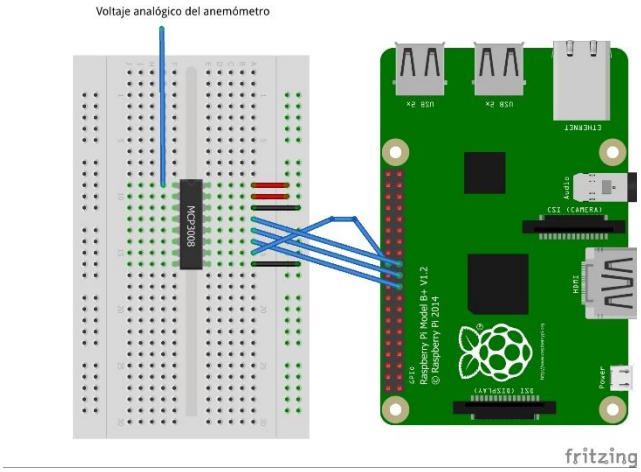

Figura 15. Esquema de circuito para conexión de la salida análoga del anemómetro por medio del convertidor MCP3008 con Raspberry PI 3B+ realizado en fritzing.

Fuente: Autor

Para determinar la velocidad del viento se realizó la ecuación de la recta con los dos puntos que el fabricante da, de la siguiente manera:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{32.4 - 0}{2 - 0.4} = 20.25$$

Ecuación 1. Pendiente de la recta del anemómetro.

$$y - y_1 = m(x - x_1)$$

$$y = 20.25x - 8.1$$

Ecuación 2. Recta de funcionamiento del anemómetro.

Donde *y* representa la velocidad del viento y, *x* equivale al valor de voltaje producido por el anemómetro.

Desde la Raspberry Pi 3B+ se realizó el programa que pudiese hacer la lectura del mcp3008, y convertir ese valor con la ayuda de la ecuación de la recta antes presentada. Para esto fue necesario instalar la librería para controlar los pines GPIO desde Python, de la siguiente manera:

- *sudo apt-get update*
- *sudo apt-get install rpi.gpio*

Dentro de un script de Python se definió una función encargada de realizar la lectura del ADC recibiendo como parámetro el número del pin del MCP3008 el cual se quiere leer. En un bucle se hace el llamado a la función del ADC y se convierte el valor retornado a un valor entre 0V y 5V para poder aplicar la ecuación de la recta del anemómetro.

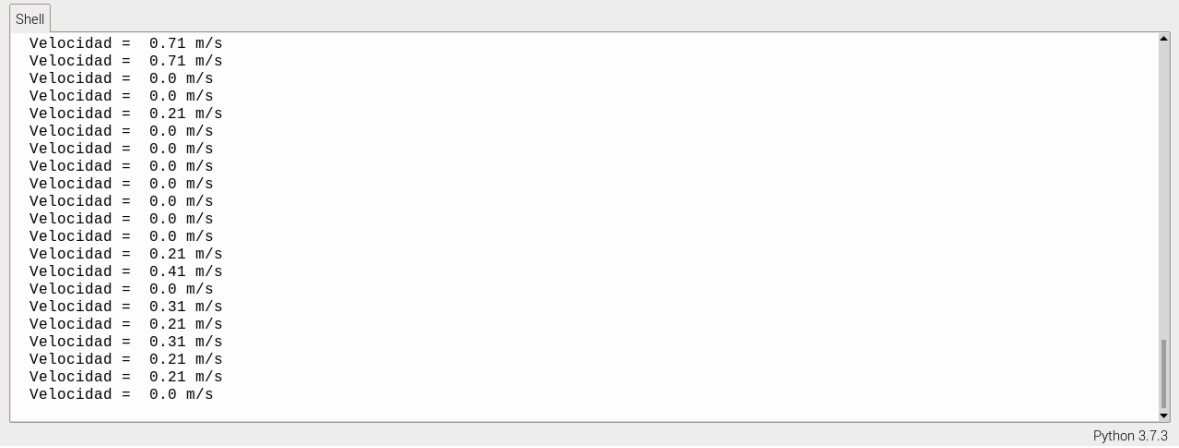

Figura 16. Valores de velocidad del viento medidos por el sensor anemómetro en Python.

Fuente: Autor.

El script anterior dio como resultado la obtención de una variable (v) que guarda la velocidad del viento obtenida por el anemómetro en metros por segundo. El algoritmo se encuentra en el anexo 2.

4.4 Sensado de voltaje, corriente y potencia.

Para la obtención de estos valores se utilizó el sensor de voltaje FZ0430, que mide tensiones en un rango de 0V hasta 25V. Por medio de una bornera, permite conectar los terminales del nodo al cual se le medirá el voltaje. El sensor tiene tres pines, de izquierda a derecha, en la siguiente imagen, son, conexión a tierra, conexión a 5V y salida análoga. No es necesario realizar la conexión de 5V ya que la alimentación la recibe del voltaje de las borneras.

Figura 17. Sensor de voltaje FZ0430 con señal análoga de salida.

Fuente: https://leantec.es/tienda/modulo-voltaje-shield-sensor-tension-voltaje-arduino-electronica/

Este dispositivo posee una salida análoga entre 0V-5V, la cual fue conectada al segundo canal de conversión del ADC MCP3008 utilizado anteriormente para la salida análoga del anemómetro.

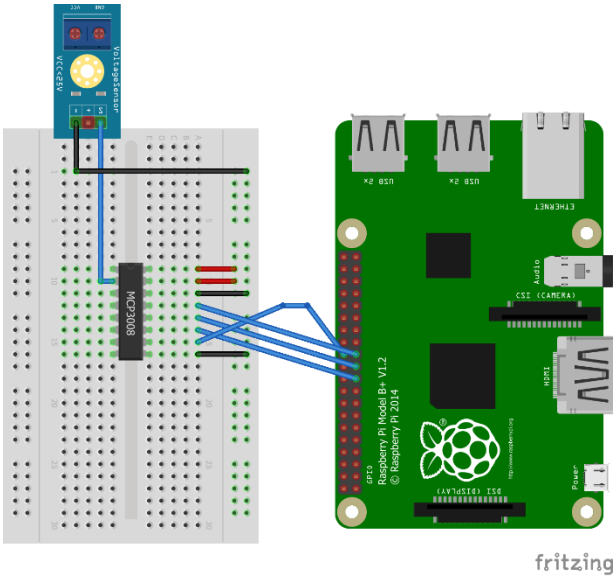

Figura 18. Esquema de circuito para conexión de la salida análoga del sensor de voltaje por medio del convertidor MCP3008 con Raspberry PI 3B+ realizado en fritzing.

Fuente: Autor

Para la medición del voltaje, se conectó este sensor a los cables de la salida del aerogenerador, los cuales estaban en una bornera en el lugar donde se encuentra el generador, como se ve en la siguiente imagen:

Figura 19. Bornera de conexión de la salida del aerogenerador con el sensor de voltaje.

Fuente: Autor

Figura 20. Valores de voltaje, corriente y potencia medidos en un script de Python.

Fuente: Autor.

El script de Python desarrollado para la lectura de este sensor dio como resultado la obtención de las variables voltaje y corriente, con una carga de 100 Ω y en el mismo de calcula la potencia generada como se ve en la imagen anterior. El programa se encuentra en el anexo 3.

4.5 Diseño de PCB.

Uno de los factores importantes para este proyecto es la adquisición de los valores de las variables climatológicas y eléctricas presentes en el lugar y en el momento de la generación de energía por el aerogenerador, por esto se realizó una placa de circuito impreso que uniera cada uno de los circuitos necesarios para la medición de temperatura, humedad y velocidad del viento mostrados anteriormente. Para las variables eléctricas se habilito la conexión de la salida del sensor de voltaje en el pin 2 del integrado MCP3008. El integrado ADC presenta un alto rendimiento y un bajo consumo de energía CMOS con una salida para resolución de 10 bits generando valores numéricos entre 0 y 1023.

Se realizó un esquema del circuito en el software Proteus, y en este mismo se diseñó el gráfico de la placa.

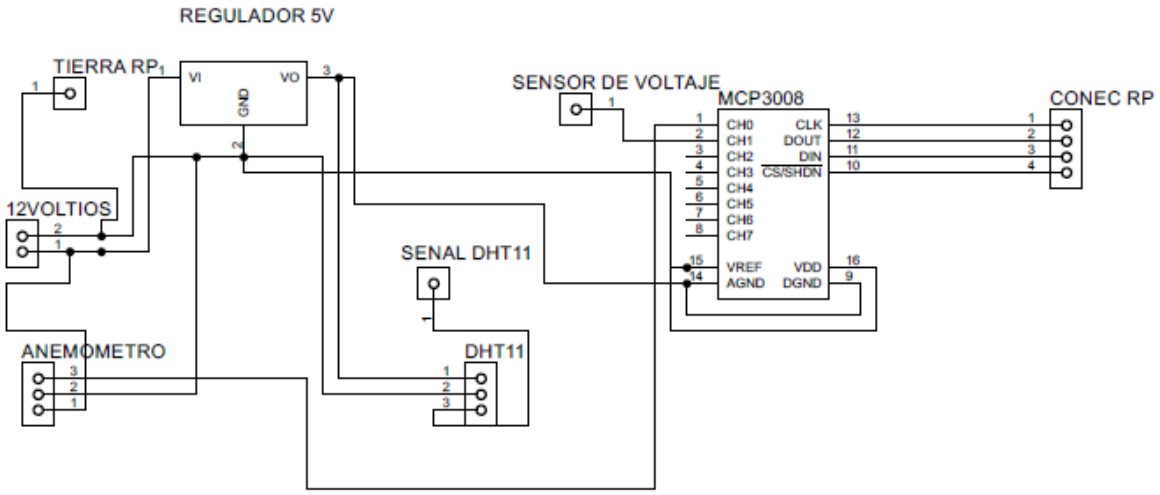

Figura 21. Esquema de circuito para adquisición de variables medioambientales (temperatura, humedad y velocidad del viento), y sensor de voltaje, realizado. en proteus.

Fuente: Autor.

Al diseñar el circuito se tuvo en cuenta la conexión necesaria para la unir la tierra de la fuente con la de la Raspberry, así como los pines para la adquisición de los valores de los sensores por medio de los pines GPIO del sistema embebido.

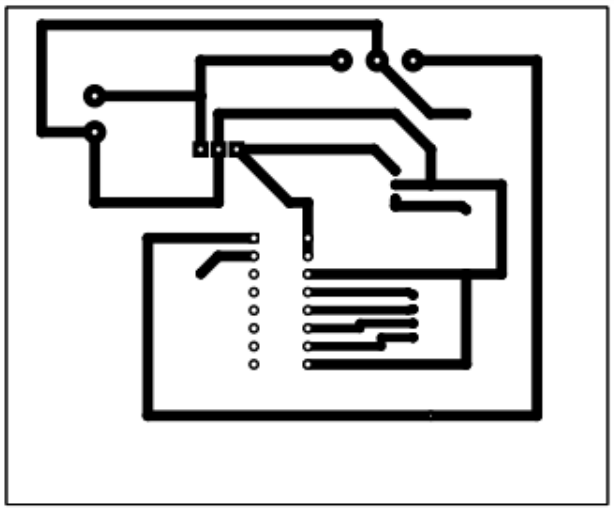

Figura 22. Esquema de impresión de PCB para variables medioambientales (temperatura, humedad y velocidad del viento), sensor de voltaje realizado. en proteus.

Fuente: Autor.

Las conexiones del anemómetro y la fuente de alimentación están representadas por borneras, ya que por medio de ellas se hizo la conexión. El sistema es alimentado por una fuente de 12V, con capacidad máxima de 10A, esta para alimentar la PCB, debido a que el anemómetro tiene un rango de voltaje de funcionamiento de 7v a 24v.

Se utilizó el regulador de voltaje L7805CV el cual puede recibir voltajes de entrada de 7V a 35V y una salida fija de 5V con 2% de tolerancia. Tiene una capacidad de corriente de 1.5A. El sensor DHT11tiene un consumo de 2.5mA y el convertidor MCP3008 un consumo máximo de corriente de 500 µA, estos dos dispositivos funcionan con 5V y por la capacidad que tiene el regulador de voltaje es posible alimentar estos dos dispositivos por su pin de salida sin esperar un mal funcionamiento por calentamiento en el regulador o una mala lectura en los sensores por falta de capacidad de corriente. Por medio de estos dispositivos se asegura que la tarjeta de adquisición diseñada tenga un bajo consumo de corriente. El sensor de voltaje no es alimentado por la fuente de alimentación de 12V, por esto no se incrusto dentro de la PCB diseñada, solo se definió un pin para la entrada de esta señal al ADC.

La placa del circuito permitió unificar los componentes necesarios para la adquisición de las tres variables ambientales y el valor del voltaje para poder ser enviados a la Raspberry.

Esta placa se realizó por medio del método de transferencia térmica, con ayuda de ácido férrico.

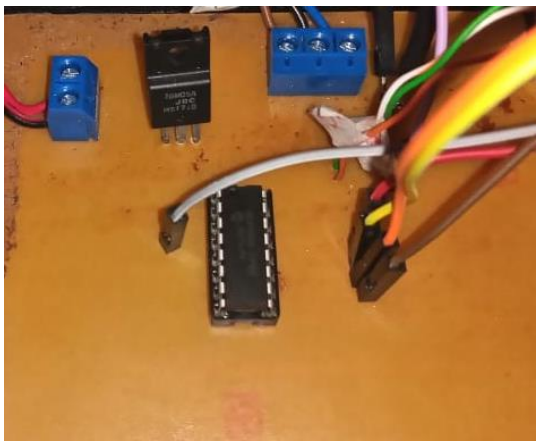

Figura 23. PCB diseñada con componentes soldados.

Fuente: Autor.

Para mayor protección la PCB se fijó en una caja con orificios solo para los cables de conexión.

4.6 Creación de hilos.

Debido a los diferentes scripts que se tienen en Python para la lectura de cada sensor, se hizo necesario unir todos estos programas en uno sólo que será ejecutado siempre desde el arranque de la Raspberry, esto se hizo por medio de la técnica de Python que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso que se conoce como *threading*. A cada flujo de ejecución que se crea durante el procesamiento se le denomina hilo o subproceso.

Como se ve en el anexo 4, se crearon 3 subprocesos en los que se obtienen los valores de las variables.

Figura 24. Valores de velocidad del viento, voltaje, corriente, potencia, temperatura y humedad tomadas en el mismo script de Python.

Fuente: Autor.

La impresión de los valores desde un *print* es generada desde cada hilo especifico, pero en el *Shell* se evidencian todos los resultados, evidenciando la ejecución de todos los subprocesos.

Esta ejecución en hilos permite que un solo script de Python sea ejecutado desde el arranque de la Raspberry, si el flujo eléctrico se corta, al volver, la Raspberry ejecuta este programa automáticamente volviendo a hacer la lectura de los sensores. Esta configuración se realizó por medio de la herramienta *crontab*, que es un archivo de texto el cual contiene la lista de scripts que se quieran ejecutar desde el arranque del sistema. Se configuro de la siguiente manera desde la terminal de las Raspberry Pi 3B+:

- *sudo crontab -e*

En la parte inferior del documento de texto se insertó la instrucción para ejecutar el script especifico, en este caso de Python, de la siguiente manera:

- *@reboot sleep 46; /usr/bin/python3 /home/pi/Documents/hilosConPost.py*

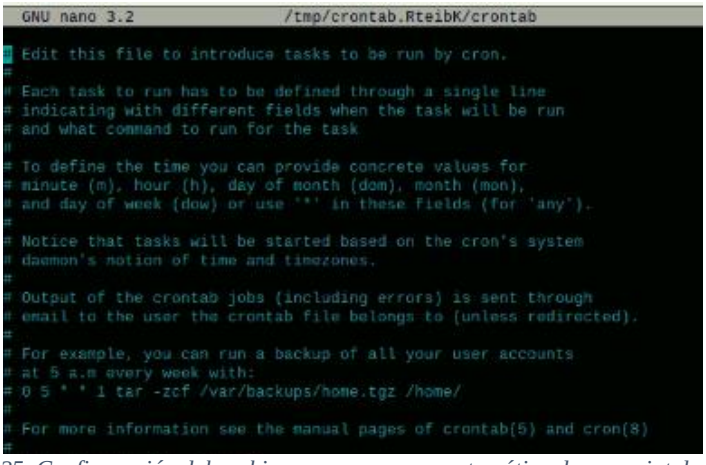

Figura 25. Configuración del archivo para arranque automático de un script de Python.

Fuente: Autor.

La instrucción anterior dio como resultado la ejecución automática del script de Python *hilosConPost.py*, por medio del intérprete de python3, desde el momento que se arranque el sistema de la Raspberry Pi 3B+, con un retardo de 46 segundos para asegurar que exista en ese momento conexión a internet.

4.7 Envío de datos desde la Raspberry al servidor.

Durante el desarrollo de la aplicación web se ejecutó el servidor en el puerto local 8080: *localhost:8080* el cual posee una dirección IP disponible sólo para la misma red en la que está conectado el equipo en el cual está ejecutando el servidor. Por esto fue posible implementar el envío de datos desde la Raspberry al servidor, teniendo presente que la dirección del servidor cambia cuando se monte la aplicación en el servidor final.

Para lograr el envío de los datos se utilizó la librería *requests* de Python la cual permite hacer peticiones HTTP a un servidor específico que para este caso es el puerto local 8080. Estos son los pasos que se realizaron en el terminal de Raspbian para poder obtener esta librería:

- *git clone git://github.com/kennethreitz/requests.git*
- *cd requests*
- *sudo python setup.py install*

En cada hilo se crea un objeto que contiene los datos medidos y se convierte al formato JSON. Por medio de una petición post se envía este objeto al servidor especificando una ruta para cada variable, como se ve en el anexo 5.

Ya que los procesos de los hilos se ejecutan permanentemente, se estarán haciendo peticiones post cada vez que sea leído algún sensor. Para el envío de datos a la base de datos se crearon unas rutas diferentes a las cuales se envía un dato promedio de los últimos 5 datos de voltaje, corriente, potencia y velocidad del viento, esto para no saturar la base de datos. El objeto JSON de temperatura y humedad se envía cada vez que se lea el sensor, ya que este posee una pausa interna.

Se hace el envío de datos a la dirección IP del servidor especificando la ruta de la API que recibe los datos, está dirección se modifica respecto a la dirección del servidor final. Se obtiene una respuesta al terminar la petición enviada desde el servidor.

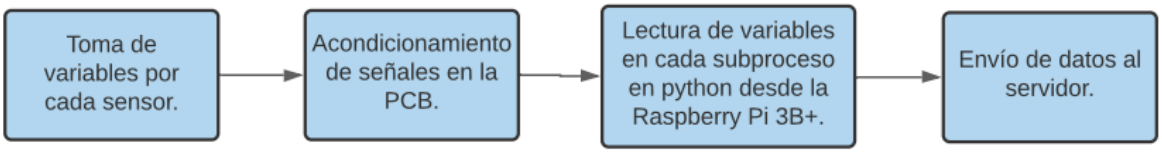

Figura 26 Diagrama de flujo para la toma de variables y el envío de datos al servidor.

Fuente: Autor.

4.8 Identificación del aerogenerador.

La línea de aerogeneradores AIR40 posee diferentes modelos basados en el voltaje nominal de salida y rango de velocidades de viento. Este aerogenerador AIR40, como se observa en la siguiente imagen, tiene un voltaje nominal de 12V, con capacidad de generación de corriente de 25A.

Figura 27 Características del aerogenerador AIR40.

Fuente: Autor.

Esta etiqueta se encuentra en un aerogenerador no instalado con el cual cuenta la Universidad de Pamplona, siendo igual al dispuesto en Villa Marina, debido a que anteriormente se encontraba en otra cabaña de esta sede social.

El aerogenerador AIR40 está diseñado para funcionar de forma independiente o para ser combinado con generadores de energía solar para subsanar la falta de velocidad de viento por variaciones climatológicas, logrando generar mayor energía.

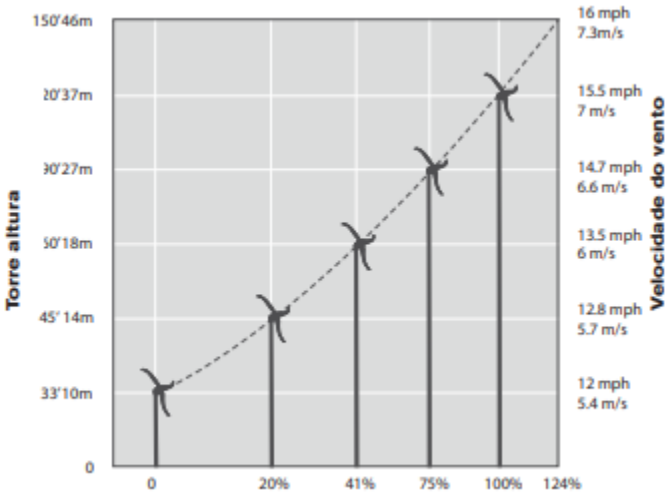

Figura 28 Velocidades de viento disponibles para diferentes alturas de una torre.

Fuente: Primus Air Manual Spanish.pdf (primuswindpower.com).

De la hoja de características, se definen las alturas necesarias en las torres para alcanzar diferentes velocidades de viento en el aerogenerador AIR40. La velocidad del viento aumenta con la altura. Si se excede una altura máxima, se supera la turbulencia del aire evitando el buen funcionamiento y una buena fuerza del viento en las palas de aerogenerador.

4.9 Desarrollo Backend.

4.9.1 Creación de servidor.

Para ejecutar JavaScript fue necesario instalar Node.js, y así, poder crear un servidor, ejecutando el siguiente comando en la terminal de Ubuntu:

- *sudo apt install nodejs*

Node Package Manager o npm, es un gestor de paquetes que facilita el trabajo con Node.js, pues permite instalar cualquier librería disponible con solo una línea de código, administra los módulos y la distribución de paquetes. Cuando se instalan nuevos paquetes desde npm se hace de forma local en la carpeta *node_modules*. Existen módulos instalados por defecto en Node.js por lo que no será necesario usar *npm install* para darles uso.

Para la creación de un proyecto de Node se creó la carpeta *backend*, que contendrá todos los archivos necesarios para el desarrollo del backend, y dentro de ella, desde la terminal, se utilizó el siguiente código el cual crea el archivo *package.json* que contiene información relevante del proyecto, los módulos y dependencias instaladas, así como los scripts de ejecución creados:

- *npm init --y*

En *package.json* se define el archivo principal que será el primero en ejecutarse, el que arranca la aplicación y se define cómo *index.js*, este archivo se organizó dentro de la carpeta *src*, la cual contiene los archivos que se crean para el backend, y junto a Express, framework de Node.js, fue mucho más sencilla la creación del servidor, pero con la misma funcionalidad que directamente con Node.

Ingresando desde la terminal al directorio *backend* creado, se instaló Express así:

- *npm install express*

Cada vez que se realiza un cambio, o se agregan elementos o funciones, el servidor se debe reiniciar para ver reflejados dichos cambios. Para evitar este proceso se utilizó *nodemon* que nos permite reiniciar el código del servidor automáticamente cada vez que se guarden los cambios de un archivo. Se instaló de la siguiente manera, en la terminal de la carpeta backend:

- *npm install nodemon*

En el archivo package.json se creó un script para ejecutar nodemon y permitir que se quede escuchando cualquier cambio en el código, así:

- *"dev": "nodemon src/index.js"*

Ya que el archivo principal es index.js, se le dice al servidor que inicie con nodemon este archivo.

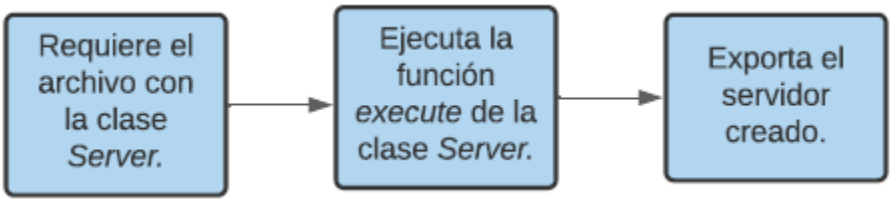

Figura 29. Diagrama de flujo para la creación y ejecución del servidor en el archivo principal: index.js.

Fuente: Autor.

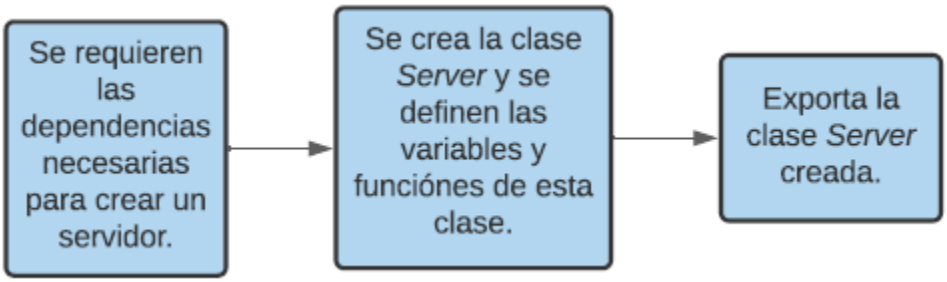

Figura 30. Diagrama de flujo para la creación de la clase Server.

Fuente: Autor.

El servidor se creó en un archivo llamado *server.js* el cual es requerido y ejecutado en el index.js como se ve en el anexo 6. El servidor web se creó como una clase (*class*) proporcionada por una versión de JS llamada ecmascript6 (ES6) con la cual se pueden definir variables y funciones que estén relacionadas dentro una misma clase. En este archivo se definió el puerto en el que se ejecuta el servidor, las validaciones necesarias antes de recibir peticiones HTTP (middlewares), las rutas de la aplicación y se creó una función que inicie el servidor con las configuraciones dadas, esta función es llamada desde el archivo index.js desde el cual se ejecuta la misma para iniciar el servidor. El código se encuentra en el anexo 7.

4.9.2 Creación de servidor con sockets.

Para utilizar sockets dentro del proyecto, se debió instalar, al igual que *cors* para configurar el socket, de la siguiente manera:

- *npm install socket.io cors*

Dentro del archivo server.js se creó la variable *io* que define una capa, sobre el mismo servidor de node, para el servidor de socket.io que se utilizó para la comunicación el tiempo real entre el servidor y los clientes. La función *configurarSockets(),* definida en la clase Server, crea un nuevo servidor de sockets enviando como parámetro los datos establecidos en la variable *io* a la clase Sockets requerida desde el archivo *socket_server.js*. Este archivo crea el socket para cada conexión que se realice y emite o escucha los eventos encargados de la comunicación entre el cliente y el servidor como se observa en el anexo 8.

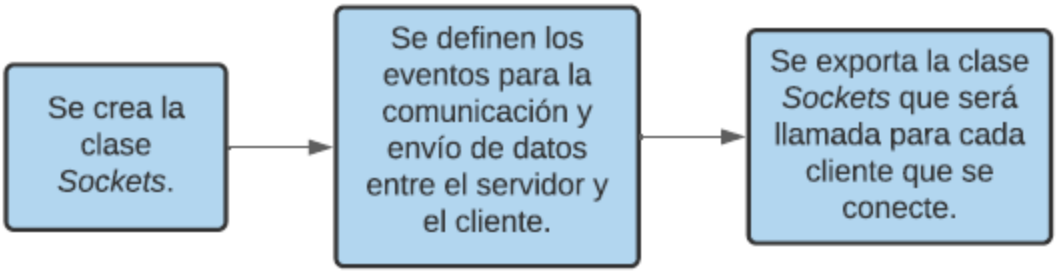

Figura 31. Diagrama de flujo para la creación del servidor de Sockets por medio de una clase.

Fuente: Autor.

Desde el controlador de cada ruta de la API que recibe los datos de la Raspberry, se hace el llamado a este socket especificando el evento y los datos a emitir. Los eventos que se crearon son:

- 'datos-temp-hum' , para emitir los datos de temperatura y humedad.
- 'datos-velocidad' , para emitir el dato de la velocidad del viento.
- 'datos-vol-vor-pot' , para emitir los datos de voltaje, corriente y potencia.

4.9.3 Conexión del servidor con la base de datos.

Mongo DB es una base de datos que utiliza lenguaje JavaScript para sus consultas y almacena datos JSON en colecciones. Para iniciar mongo debió ser instalado en el sistema operativo de la siguiente manera:

- *sudo apt update*
- *sudo apt install -y mongodb*

Por defecto, esta base de datos no se ejecuta desde el arranque del sistema, por eso, antes de utilizarla se debe ejecutar desde la terminal:

- *sudo service mongod start*

Dentro de src se creó el archivo *database.js* en el cual se definió la conexión con la base de datos como se observa en el anexo 9, por medio de un módulo llamado *mongoose* el cual permite definir modelos de datos para almacenar en colecciones dentro de mongo DB. Este módulo se instaló dentro del proyecto desde la terminal así:

- *npm install mongoose*

En este archivo se define la dirección de ubicación de la base de datos y el nombre que se quiere asignar, por medio del método *connect* de mongoose se inicia la conexión con su respectiva promesa. En el archivo server.js, se requiere el archivo de conexión de la base de datos y se ejecuta.

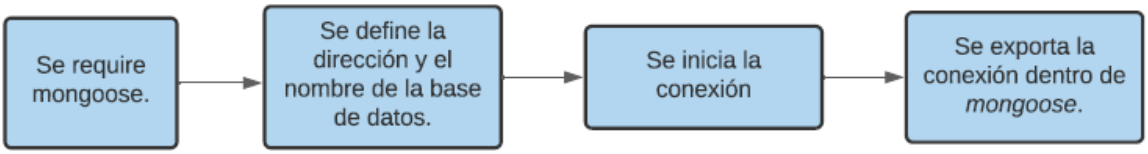

Figura 32. Diagrama de flujo para la conexión de la base de datos con el servidor.

Fuente: Autor.

4.9.4 Creación de rutas.

Se creó una *Rest API* (Interfaz de aplicaciones para transferir datos) con la cual se realizó el envío y la recepción de datos entre un cliente y el servidor. En este proyecto el cliente que envía datos es la Raspberry. Esta API utiliza HTTP como protocolo de comunicación ya que se pueden hacer diferentes peticiones a una misma dirección. Como se ve en el anexo 5, cada subproceso del script de Python hace una petición post al servidor, esta debe ser entendida por el servidor y realizar un proceso dentro de ella. Cada dato es recibido en una dirección diferente, por esto fue necesario crear una carpeta, dentro de src, llamada *routes* con un archivo dentro determinado como *api.js* en el cual, por medio del enrutador de express, se definen las acciones a realizar por cada ruta dependiendo del tipo de petición HTTP como se observa en el anexo 10.

Las rutas de la API son:

- /velocidad
- /velocidad-db
- /temp-hum
- /temp-hum-db
- /vol-cor-pot
- /vol-cor-pot-db
- /dia-especifico
- /semana-especifica
- /dia-ultimo
- /hora-ultima
- /dia-especifico-temperatura
- Semana-especifica-temperatura

Para que el servidor pudiera utilizar este archivo de rutas creado, fue necesario hacer la importación de este dentro del archivo server.js, desde el cual se da una dirección anterior por defecto: */api*, para diferenciar la API de las demás ubicaciones y se ejecuta el módulo importado.

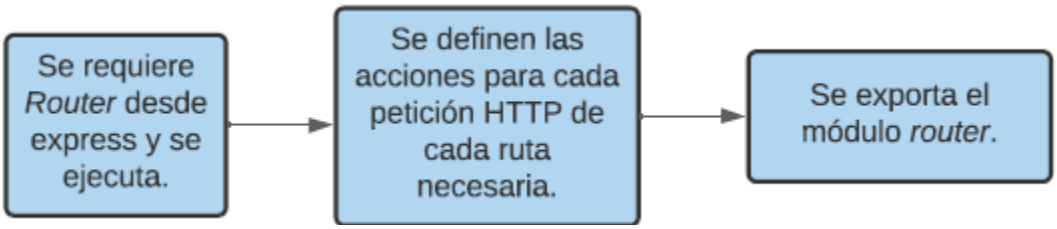

Figura 33. Diagrama de flujo para la creación de rutas.

Fuente: Autor.

4.9.5 Creación de modelos de datos.

Se debió definir los esquemas o modelos de datos que se almacenan dentro de la base de datos de mongo llamada *datos-monitoreo* como se le asignó en el anexo 9, esto nos permite determinar qué tipos de datos se van a recibir, el nombre de las claves y exportar este esquema.

Dentro de la carpeta *src* se creó el directorio *models* y dentro de él, un archivo con extensión *.js* para cada modelo de datos; para esto se requirió el módulo *mongoose*, que se utilizó en la creación de la conexión entre la base de datos y el servidor, pero en este caso se utilizaron sus funciones para crear esquemas y modelos.

Cada modelo es almacenado dentro de la base de datos como una colección diferente. Desde la Raspberry se envían tres objetos, uno contiene los valores de temperatura y humedad, otro el valor de la velocidad del viento y el último los valores de voltaje, corriente y potencia. Para cada objeto se creó un esquema como se ve en el anexo 11. Se crearon estos esquemas de forma tal que cada vez que sea llamado el esquema, se guarde por defecto la fecha y hora en la que se agregó el dato.

Los modelos creados fueron:

- *temp_hum*, para los datos de temperatura y humedad
- *velocidad*, para los datos de la velocidad de viento.
- *volCorPot*, para los datos de voltaje, corriente y potencia.

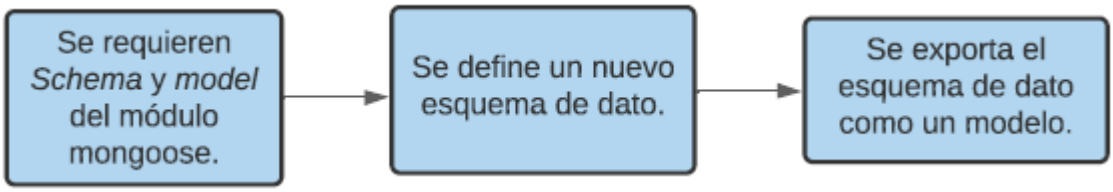

Figura 34. Diagrama de flujo para la construcción de los modelos de datos.

Fuente: Autor.

4.9.6 Creación de controladores de rutas.

Como se aprecia en el anexo 10, para cada ruta creada se tienen diferentes peticiones HTTP con métodos específicos. Estas acciones se crearon desde un archivo controlador para cada ruta, en el cual se define una función para cada petición. Cada controlador maneja una petición GET o POST dependiendo del uso que tenga la ruta.

Los controladores para las peticiones GET, devuelven como respuesta al cliente, el objeto JSON con los datos recibidos desde la base de datos. Estas peticiones se ejecutan para generar la gráfica de la ultima hora y el ultimo día, ya que no es necesario enviar parámetros desde el cliente

Los controladores para las peticiones POST, reciben el objeto JSON enviado desde Python, o desde el navegador, especificando la fecha o las fechas para las cuales se buscarán los datos. Como se observa en el anexo 5, cada subproceso del script de Python tiene una ruta de la API a la cual hace la petición POST. El servidor responde con un mensaje de éxito de la petición. Si se envía una petición POST a */api/temp-hum* el servidor recibirá el objeto enviado y por medio de los eventos de sockets envía esos datos al fronted para ser pintados. En la llamada al socket se especifica el mensaje y los

datos a enviar. Cada vez que se realiza una petición POST los datos se actualizan en el fronted en tiempo real.

El controlador para las peticiones POST enviadas a las rutas terminadas en *-db* recibe el objeto JSON el cual es guardado en la colección especifica de la base de datos. SI se envía una petición POST a */vol-cor-pot-db* el servidor recibe el objeto y lo guarda en la base de datos *datos-monitoreo* en la colección *volcorpots*. Las tres colecciones en la base de datos son:

- *temp_hums*, para los datos de temperatura y humedad, enviados a la dirección: */api/temp-hum-db* por medio de una petición POST.
- *velocidads*, para el dato de la velocidad del viento, enviado a la dirección*: /api/velocidad-db* por medio de una petición POST.
- *volcorpots*, para los datos de voltaje, corriente y potencia, enviados a la dirección*: /api/vol-cor-pot-db* por medio de una petición POST.

Estos datos son los utilizados para los estudios estadísticos.

Estos controladores se encuentran en el anexo 12. En el archivo api.js se importan estos controladores y se asignan a cada ruta especifica.

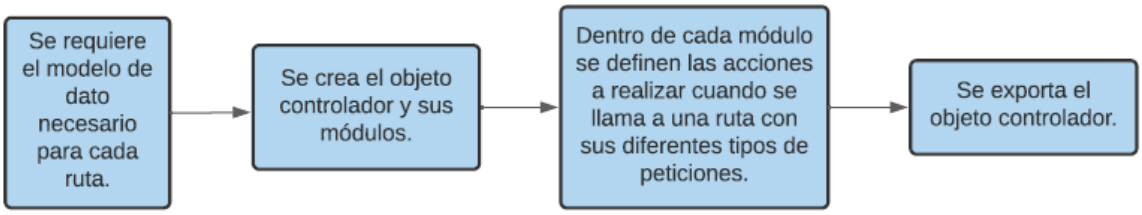

Figura 35. Diagrama de flujo para la construcción de los controladores para las rutas de la API.

Fuente: Autor.

Los controladores creados fueron:

- *dia-especifico*, para la ruta /dia-especifico.
- *dia-especifico-temperatura*, para la ruta /dia-especifico-temperatura.
- *dia-ultimo*, para la ruta /dia-ultimo.
- *hora-ultima*, para la ruta /hora-ultima.
- *semana*, para la ruta /semana-especifica.
- *semana-temperatura*, para la ruta /semana-especifica-temperatura.
- *temp-hum.controller*, para las rutas /temp-hum y /temp-hum-db.
- *velocidad.controller*, para las rutas /velocidad y /velocidad-db.
- *vol-cor-pot.controller*, para las rutas /vol-cor-pot y /vol-cor-pot-db.

4.10 Desarrollo Fronted.

4.10.1 Creación de servidor de Fronted.

En paralelo a la carpeta *backend*, se creó el directorio *fronted*, el cual se generó por medio de la herramienta de React: *create-react-app*, por lo cual se escribió el siguiente código en la terminal de Ubuntu:

- *npx create-react-app fronted*

Con lo anterior se creó un proyecto completo de React, con algunas dependencias ya instaladas, una configuración de comunicación entre archivos, soporte para JSX por medio de conversión interna y algunos archivos de ejemplo, de los cuales solo se utilizaron: App.css, App.js, index.css e index.js. A estos archivos se les borró el contenido para iniciar el proyecto sin ninguna vista en el navegador. Los demás archivos dentro de la carpeta *src* fueron eliminados. Para iniciar el servidor de React, se escribe el siguiente comando en la terminal, ingresando primero a la carpeta fronted:

- *cd fronted*
- *npm start*

Con lo anterior se inicia por defecto un servidor en el puerto 3000, y es posible debido al script *start* creado por *create-react-app* en el archivo *package.json*.

El archivo principal: *index.js*, renderiza a la sección *root* del archivo *index.html*, como componente de React, el archivo creado *MonitoreoApp.js*, como se ve en el anexo 13; este último documento importa *App.js* y lo exporta incluyendo las funciones de los sockets (anexo 14).

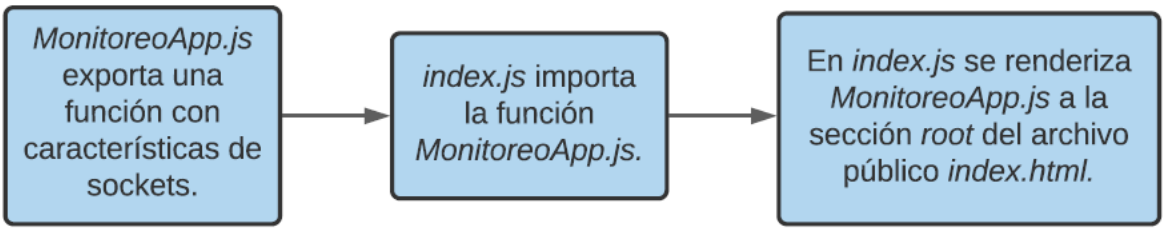

Figura 36. Diagrama de flujo para el envío de archivos hasta el documento público .HTML.

Fuente: Autor.

En *App.js*, se creó el esquema general de la página, como una función, sus rutas, navegación y pie de página (anexo 15). Este esquema estará presente en todas las vistas, debido a que cambia solamente el componente central, pero la navegación es necesaria para cambiar de vista. Se instaló el framework Bootstrap, de CSS, para estilizar los componentes y utilizar plantillas predefinidas. Se instaló la librería: *react-router-dom* para importar *Browser-router* y *Route*, el primero es el contenedor que permite crear rutas en la aplicación, y el segundo contiene las rutas que se quieren crear.

Las rutas creadas fueron:

- */* Página inicial.
- */tiempo-real* Vista para los valores de los sensores en tiempo real.
- */estadísticas* Gráficos y estudios estadísticos con los datos en la DB.
- */about* Información del proyecto, autor, director, universidad.

Dentro de los *Route* se define el *path* y el componente a renderizar para cada dirección, importados en el mismo archivo. Se exportó la función *App*.

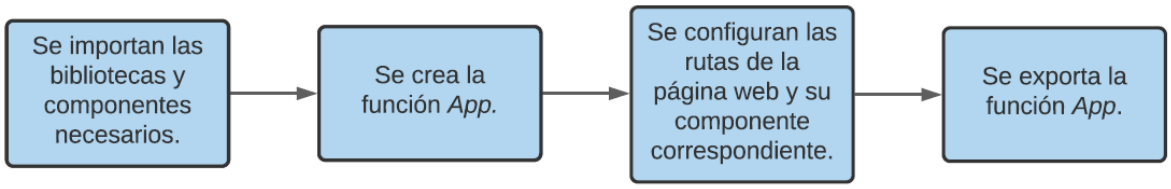

Figura 37. Diagrama de flujo para la construcción del archivo principal con las rutas necesarias.

Fuente: Autor.

4.10.2 Creación de componentes de React.

Dentro de la carpeta *src* (source), se creó el directorio *components*, que contiene todos los componentes que se crearon para lograr las vistas de la página, como la navegación, las tablas, y lo renderizado en cada ruta. En cada componente se creó una clase o una función la cual es exportada en el mismo archivo e importada en *App.js*.

4.10.2.1 Componente: Navigation.

En el sitio oficial de Bootstrap se encuentran los ejemplos de componentes que se pueden utilizar, de ahí se tomó el código para la navegación (https://getbootstrap.com/docs/5.1/components/navbar/). Dentro de la carpeta *components*, se creó el archivo *navigation.js*, en el cual se insertó el código de la plantilla de la navegación. Debido a que estos ejemplos están preparados para ser utilizados con HTML, CSS y JS, se presentaron algunos errores puesto que en React se trabaja con JSX. Fue necesario cambiar la forma de llamar algunos componentes para lograr el correcto funcionamiento, aunque no fueron cambios drásticos, se evitaron incompatibilidades, como, por ejemplo, la etiqueta *class*, se modificó como *className*.

Para evitar que al cambiar de vista por la navegación se recargue la página, sino que todo se mantenga en la misma aplicación, se importó el módulo *Link* desde *react-router-dom*, para configurar hacia donde enviar al usuario sin recargar la página. Esto se hace con el atributo *to* del módulo *Link*, en el cual se definió a que vista o ruta se enviará el usuario con cada botón de la navegación. Al estar en Inicio y presionar Acerca de, se muestra el componente *about*, que contiene la información acerca del proyecto, sin refrescar la página.

Figura 38. Componente navegación.

Fuente: Autor.

Se modificaron los aspectos de la navegación para lograr la vista que se quería como se ve el anexo 16.1.

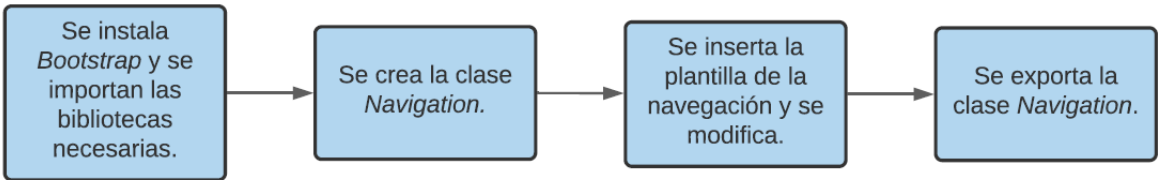

Figura 39. Diagrama de flujo para la construcción del archivo principal con las rutas necesarias.

Fuente: Autor.

4.10.2.2 Componente: Carrusel.

El carrusel permite comunicar a los usuarios información importante o relacionada con el tema de la página web, con imágenes y textos cortos. Para crear este componente se instaló *reactsrap* con *npm*. Reactstrap permite trabajar en React utilizando toda la funcionalidad y facilidad de Bootstrap.

En el sitio oficial de Reactsrap se encuentran los ejemplos de los componentes que podemos usar, de allí se tomó el código para el carrusel utilizado (https://reactstrap.github.io/?path=/docs/components-carousel--carousel). En la carpeta *components* se creó el archivo *carrusel.js*, en el cual se insertó el código seleccionado. Desde ahí se modificaron las imágenes, los colores, los vínculos de cada imagen y los textos como se ve en el anexo 16.2.

Figura 40. Componente Carrusel.

Fuente: Autor.

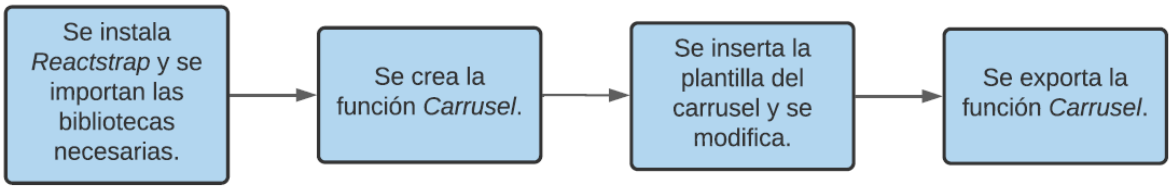

Figura 41. Diagrama de flujo para la elaboración del componente Carrusel.

Fuente: Autor.

4.10.2.3 Componente: Inicio.

Es la vista principal de la página web. Se creó el archivo *inicio.js* dentro de la carpeta *components*. En este se importó el componente *carrusel* y las librerías necesarias para la creación de este componente.

Figura 42. Componente Inicio.

Fuente: Autor.

Se creó una función encargada de retornar la vista con el diseño esperado, dentro de la cual se llamó al carrusel, se creó un elemento div para dar un resumen del proyecto y dos botones para ver los sensores en tiempo real o las estadísticas de producción. El pie de página fue importado como otro componente insertado al final de la página. Por último, se exporto la función creada como se ve en el anexo 16.3.

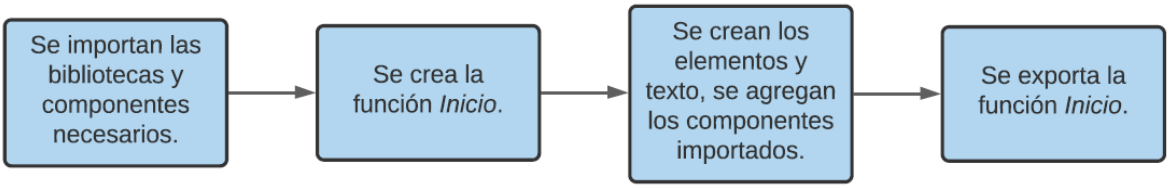

Figura 43. Diagrama de flujo para la construcción del componente Inicio.

Fuente: Autor.

4.10.2.4 Componente: Footer.

Dentro de la carpeta *components* se creó el archivo *fotter.js* el cual contiene una función que retorna el pie de página del sitio web como se observa en el anexo 16.4.

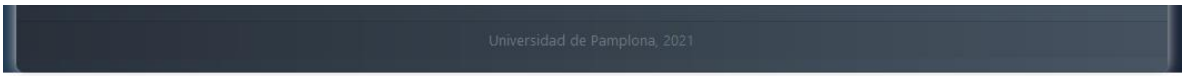

Figura 44. Componente Footer.

Fuente: Autor.

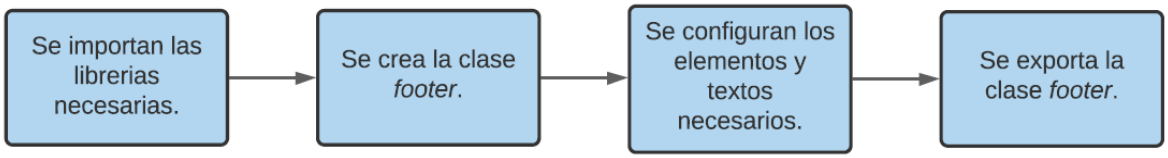

Figura 45. Diagrama de flujo para la construcción del componente Footer.

Fuente: Autor.

4.10.2.5 Componente: About.

Se creó el documento *about.js* donde se da información relevante del proyecto, título, objetivos, como se desarrolló, autor, director, contacto, por medio de una función que retorna esta información (anexo 16.5).

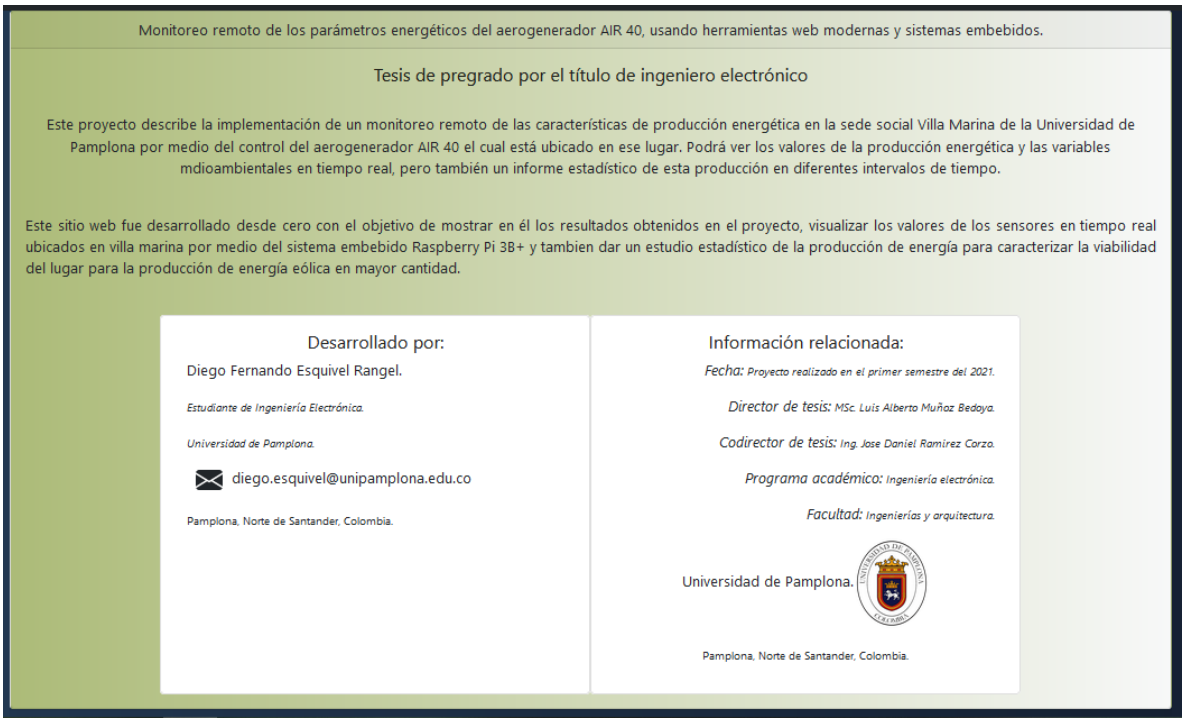

Figura 46. Componente About.

Fuente: Autor.

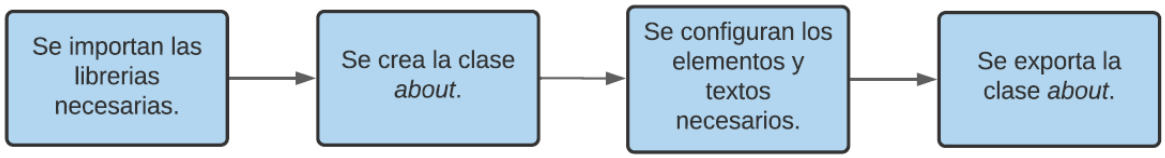

Figura 47. Diagrama de flujo para la construcción del componente About.

Fuente: Autor.

4.10.2.6 Componente: Tiempo_real.

Los Hooks de React permiten utilizar estado y otras características de React sin escribir dentro de una clase.

Este componente es el encargado de escuchar los eventos emitidos desde el servidor de sockets del backend con *socket.on*. Por medio del Hook de React, *useState*, se pueden crear variables y definir la función para modificar este dato. Para cada evento se creó un *useState* que permite definir el valor de cada variable antes de escuchar por primera vez el socket, por esto al abrir la página web en un navegador los valores que se aprecian son 0.

UseEffect, es un hook que se ejecuta cuando el componente es renderizado y cuando el componente se actualiza. Por medio de este hook se escuchan los eventos emitidos desde el backend y los datos recibidos se envían al useState especifico.

La función *TiempoReal*, creada en este componente, retorna la vista de los datos que llegan por medio del socket. Dentro de una etiqueta *<p>*, son enviadas las variables de cada *useState* que contienen los datos recibidos. Esta configuración se encuentra en el anexo 16.6.

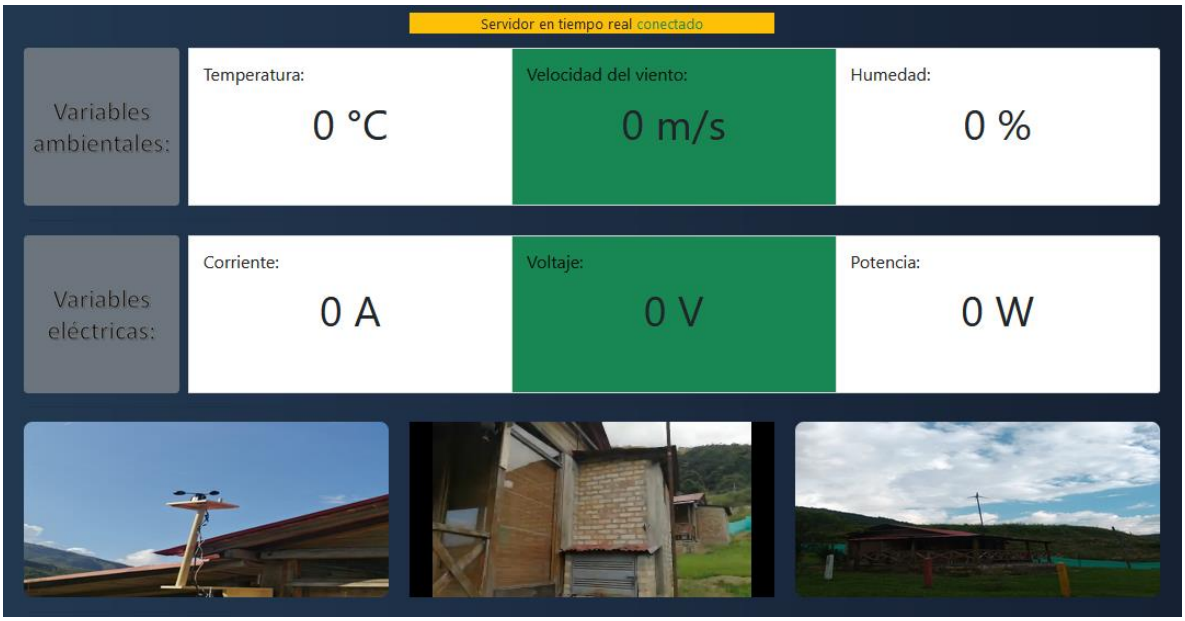

Figura 48. Componente Tiempo real.

Fuente: Autor.

Para monitorear la conexión con el servidor se creó una condición evaluando la existencia de la variable *online* importada por medio de *useContext*, que permite ver el estado de esta conexión.

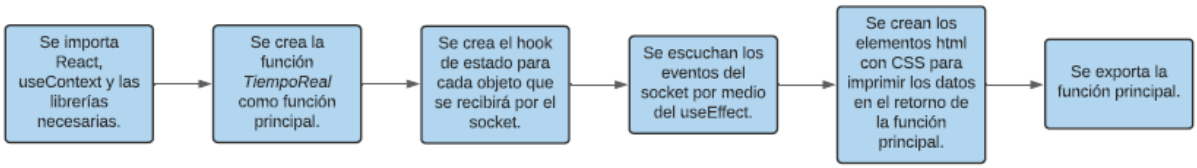

Figura 49. Diagrama de flujo para la construcción del componente Tiempo real.

Fuente: Autor.

4.10.2.7 Componente: Estadisticas.

Este es el componente que muestra las gráficas creadas a partir de los datos disponibles en mongo DB. Se creó la clase *Estadisticas*, como componente de React, la cual hace el llamado a otros dos componentes que generan las gráficas. Acá se determinaron los contenedores, la ubicación en el espacio y la organización de cada una de las gráficas necesarias. Por defecto se exportó la clase creada como se ve en el anexo 16.7.

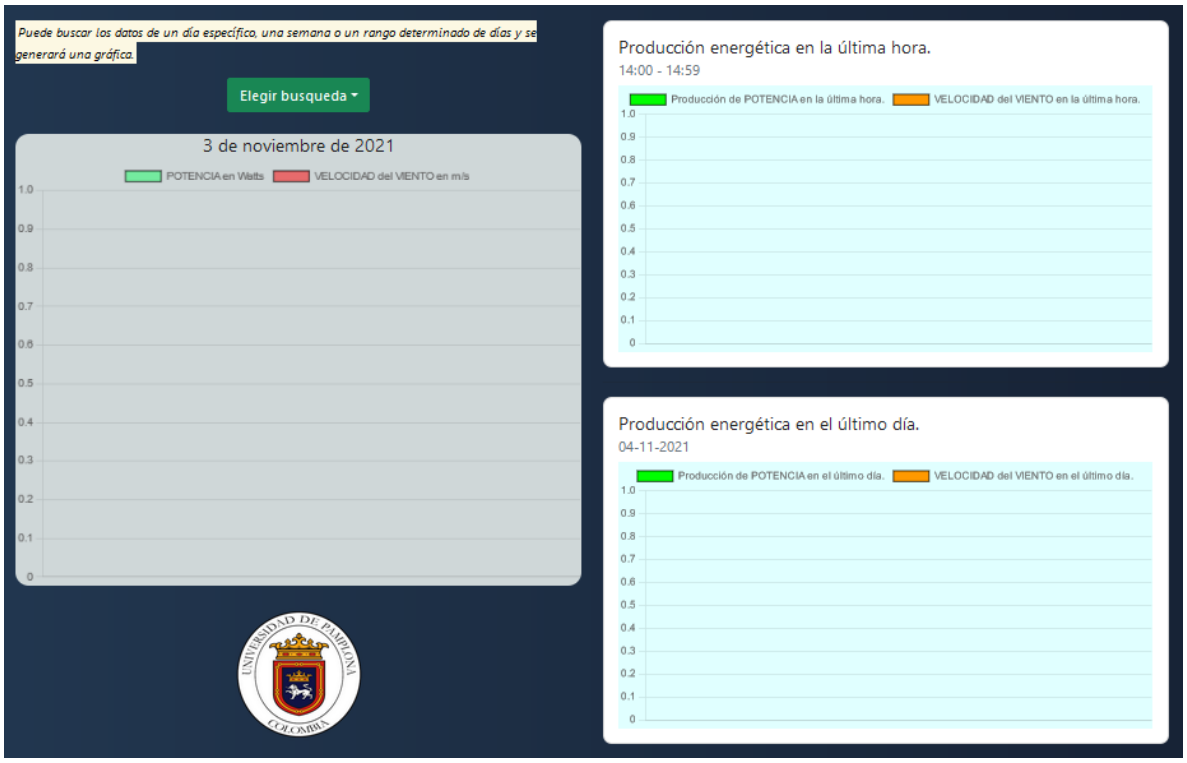

Figura 50.Componente Estadísticas.

Fuente: Autor.

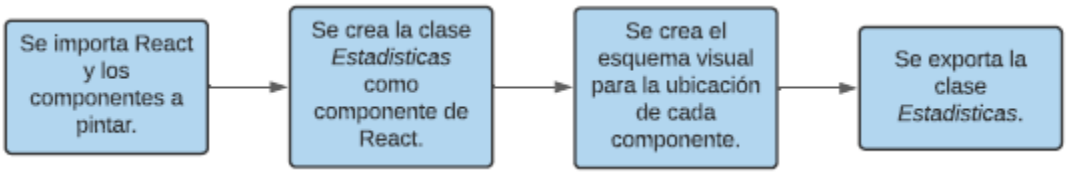

Figura 51. Diagrama de flujo para la construcción del componente Estadisticas.

Fuente: Autor.

4.10.2.8 Componente: Consulta.

En este componente se realiza la consulta a la base de datos y se genera la gráfica con los datos recibidos. Por medio de la clase *Consulta*, creada como componente de React, se dan dos opciones de búsqueda para el cliente, Día y Semana, las cuales permiten generar una gráfica de los datos de un día específico o de un rango determinado de días.

Por medio de *axios*, que permite hacer peticiones HTTP desde React, se realiza una petición POST al backend en la que se envía la fecha, o el rango de fechas, que el cliente eligió, recibiendo como respuesta los datos de la velocidad del viento y la potencia en esa fecha.

Chart.js es un plugin de JS sencillo y completo, que permite insertar gráficas en una página web. Se instaló por medio de *npm* y se importó en el componente. Los datos recibidos de la petición POST

Figura 52. Componente Consulta.

Fuente: Autor.

se almacenan en un objeto JSON el cual es enviado a la función que generara la gráfica para colocar los datos en los ejes correspondientes con las etiquetas y las especificaciones dadas (anexo 16.8). Esta gráfica es insertada en una etiqueta *canvas*.

Se exportó por defecto la clase creada la cual es pintada desde el componente *Estadisticas*.

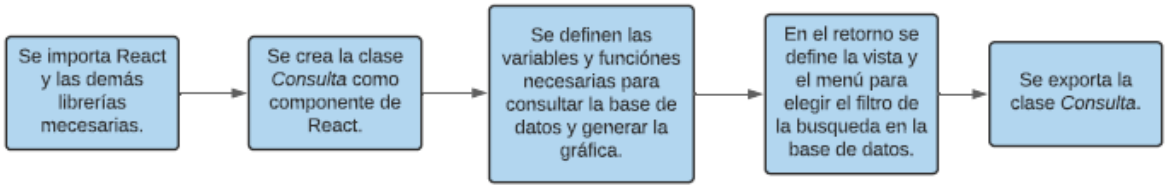

Figura 53. Diagrama de flujo para la construcción del componente Consulta.

Fuente: Autor.

4.10.2.9 Componente: Ultimas.

Este componente es el encargado de generar las gráficas automáticas, de la última hora y del último día. Se creó la clase *Ultimas*, como componente de React, desde la cual se hace la petición GET al

backend para recibir el objeto JSON con los datos en Mongo DB. Con Chart JS se genera las gráficas y se ubican en sus contenedores.

Al renderizar la clase se ejecutan las funciones para generar las gráficas, logrando que se muestren al entrar en el componente, el cliente no tiene que solicitarlas. Las gráficas son mostradas por medio de una etiqueta *canvas* con el id correspondiente para cada caso (anexo 16.9).

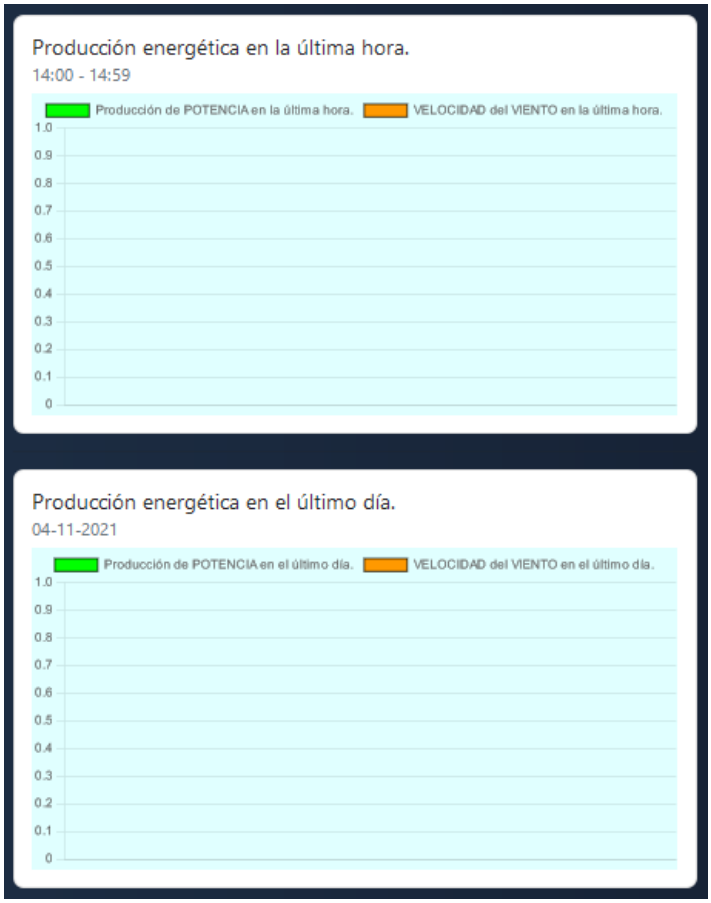

Figura 54. Componente Ultimas.

Fuente: Autor.

Se exporta por defecto la clase creada la cual es pintada desde el componente *Estadisticas*.

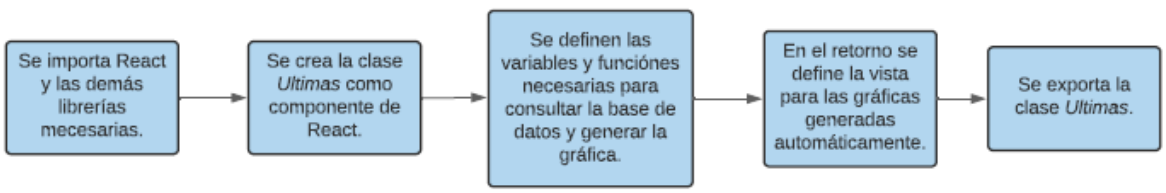

Figura 55. Diagrama de flujo para la construcción del componente Ultimas.

Fuente: Autor.

4.10.2.10 Componente: Temperatura.

Es llamado desde *consulta.js*, ya que genera una gráfica que representa la temperatura y humedad. Se crearon dos funciones dentro del archivo, una para cuando se busca por día y otra cuando es por semana o un rango de días. Estas funciones fueron exportadas, y se requieren en el componente *consulta.js*, el cual realiza la llamada a estas enviando como parámetro el día que se quiere buscar o el día inicial y final de la búsqueda. Este código se encuentra en el anexo 16.10.

Desde cada función se genera una petición POST, por medio de *axios* para recibir los datos desde el servidor, y se grafican con el uso de *Chart*, enviando la gráfica creada al elemento con *id* especifico, ubicado en el componente *consulta.js* y se pinta en el componente *Estadisticas*.

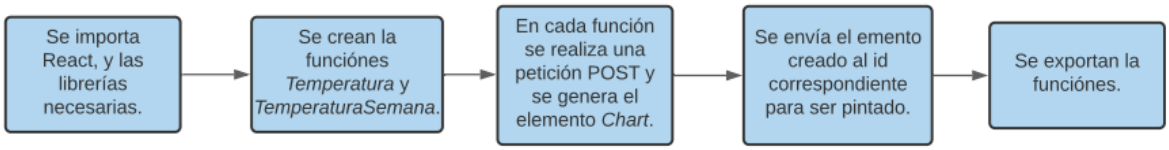

Figura 56. Diagrama de flujo para la construcción del componente Temperatura.

Fuente: Autor.

4.10.3 Custom Hook: useSocket.

Dentro de la carpeta *src*, se creó el directorio *hooks*, dentro del cual se estableció el archivo *useSocket.js*, como se aprecia en el anexo 17. Se importó React y los hooks useEffect y useState. Para el uso de los sockets del lado del servidor fue necesario instalar *socket.io-client* mediante *npm*. Esta Liberia se importó bajo el nombre de *io* en el archivo *useSocket.js* creado y se definió la constante *useSocket* como una función principal del archivo.

La función principal recibe como parámetro la dirección del servidor del backend donde está el socket. Por medio del método *connect* de *io* se realizó la conexión con el servidor guardándose en la constante socket.

Por medio de un *useState* en la constante *online* se monitorea la conexión del servidor del socket en el fronted con el del backend, el cual da un valor *true* cuando se conecta o se recupera la conexión y un valor *false* cuando la conexión se pierde. Con *useEffect* se escuchan los eventos *'connect'* y *'disconnect'* del socket para saber el estado de la conexión y setear el *useState*.

La función principal retorna las constantes *socket* y *online*.

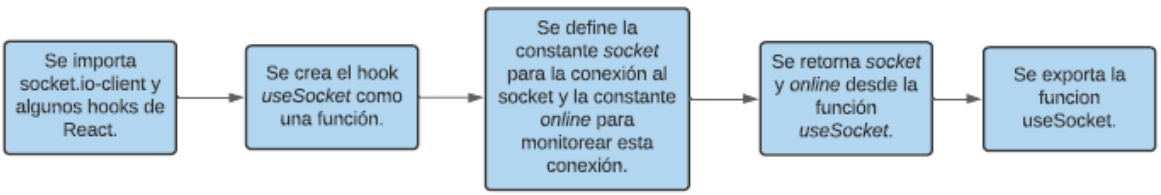

Figura 57. Diagrama de flujo para la construcción del hook para el uso del socket.

Fuente: Autor.

4.10.4 SocketContext.

Para evitar que se cree una conexión diferente al servidor de socket en el backend cada vez que se utilice el socket, el *customHook* creado se alojó en el *Context* de React, para desde él distribuir información a todos los componentes hijos. Dentro de *src* se creó la carpeta *context* y dentro el archivo *SocketContext.js*. En este archivo se creó la constante *SocketContext* con *createContext* de *React*, como se ve en el anexo 18.

Se creó *SocketProvider* para facilitar la comunicación del componente principal con los hijos. Esta función recibe como parámetro el *children* que será pintado desde el componente *SocketContext.Provider* creado en el retorno de esta función.

Se importó el hook *useSocket* para crear el socket en el context, se envía como parámetro la URL donde se ubica el servidor del backend. Dentro del componente *SocketContext.Provider* se coloca a disposición de toda la aplicación de React los objetos *socket* y *online*.

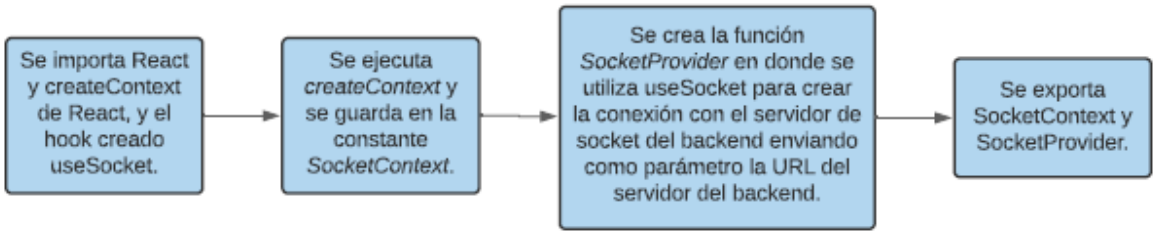

Figura 58. Diagrama de flujo para la construcción del hook de Contexto de la aplicación.

Fuente: Autor.

4.10.5 Index.html.

En la carpeta *public* se ubican los archivos estáticos, que serán enviados al navegador para pintarlos. Por esto, desde que se creó el proyecto de React con *créate-react-app*, se creó este directorio, y dentro de él el archivo *index.html,* (anexo 19), con una estructura básica de html-5. En este documento se definió el titulo e icono de la página web y se creó el componente con id *root*, el cual recibirá y pintará en el navegador todos los componentes enviados desde la carpeta *src*.

4.11 Instalación del sistema de adquisición de datos.

El aerogenerador AIR 40 se encuentra instalado en la Sede Social Villa Marina de la Universidad de Pamplona, en la cabaña 2 y cuenta en su base con un cuarto de control, en el cual se ubicó la Raspberry Pi 3B+, la fuente de alimentación de 12V y la caja plástica de la PCB realizada.

Figura 59. Cuarto de control ubicado cerca al aerogenerador.

Fuente: Autor.

Para la ubicación del anemómetro fue necesario construir una base en madera para buscar la mayor cercanía del sensor con el aerogenerador y lograr evitar que las paredes y el techo de la cabaña hicieran barrera al paso del viento hacia el sensor y generar una mala medición de la velocidad del

Figura 60. Anemómetro y sensor DHT11 instalados en una base de madera.

Fuente: Autor.

mismo. En esta misma base se situó el sensor DHT11 para una mejor medida de la temperatura del ambiente.

Debido a la ausencia de internet en Villa Marina fue necesario utilizar un router para proporcionar una conexión a internet a la Raspberry y lograr enviar los datos hacia el servidor. El router Huawei E5172 utilizado, recibe señal de alimentación de internet por medio de una SIM CARD y es conectado a la Raspberry por medio del puerto RJ45, aprovechando que los dos dispositivos se encuentran en el cuarto de control.

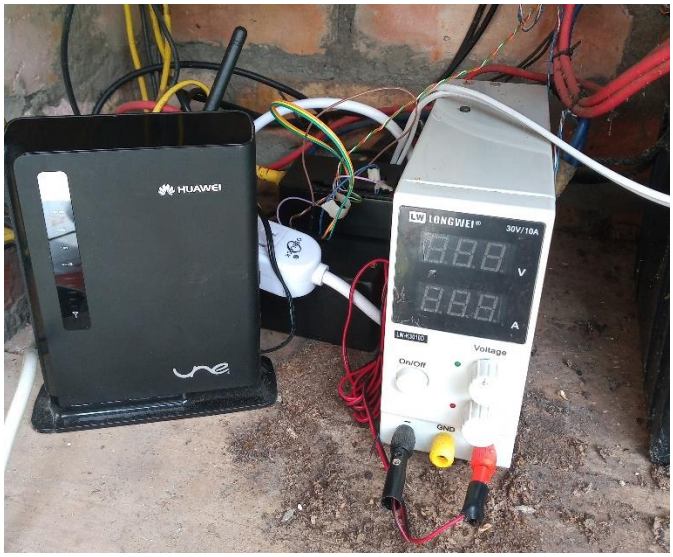

Figura 61. Sistema de adquisición de datos ubicado en el cuarto de control.

Fuente: Autor.

4.12 Montaje de la aplicación web completa en el servidor.

Para alojar la página web en un servidor, la Universidad de Pamplona, permitió el acceso a una máquina con conexión a internet permanente para allí almacenar la información que se publicará en el sitio web, la base de datos y manejar las peticiones HTTP que realizan los clientes.

Este computador está ubicado en el edificio Eduardo Cote Lamus (EC) de la sede principal de la Universidad de Pamplona y cuenta con el sistema operativo Ubuntu 20.04. Para utilizar esta máquina como servidor fue necesario instalar algunas librerías.

4.12.1 Instalación de NodeJS.

Esta dependencia permite ejecutar JavaScript del lado del servidor, se instaló de la siguiente manera en la terminal de Ubuntu:

- *curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -*
- *sudo apt-get install -y nodejs*

Adquiriendo la versión LTS 14.17.6 de Node.

4.12.2 Instalación de Mongo DB.

En este equipo se guardarán los datos enviados desde la Raspberry Pi 3B+, manejados por medio de Mondo DB. Esta base de datos en su versión 4.4.9, se adquirió por medio de los siguientes comandos ejecutados en la terminal de Ubuntu:

- *curl -fsSL https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add –*
- *apt-key list*
- *echo "deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list*
- *sudo apt update*
- *sudo apt install mongodb-org*

El servicio de la base de datos por defecto está inactivo, pero es necesario que esté funcionando en todo momento, para esto se realizó el siguiente proceso en la terminal de Ubuntu.

- *sudo systemctl start mongod.service*
- *sudo systemctl status mongod*
- *sudo systemctl enable mongod*

Con esta última línea de código se asegura que desde el arranque se inicie el servicio de mongo.

4.12.3 Instalación de UFW.

Uncomplicated FireWall (UFW, cortafuegos sin complicaciones) es un cortafuegos diseñado por Ubuntu, desarrollado para ser de fácil uso. Este firewall permite configurar las tablas de firewall nativas de Linux y habilitar o deshabilitar conexiones entrantes al servidor, aumentando la seguridad del mismo.

Por defecto, UFW se encuentra deshabilitado, se activó de la siguiente manera desde la terminal de Ubuntu:

- *sudo ufw status*
- *sudo ufw enable*
- *sudo ufw status*

En la última línea de cogido se verifica nuevamente el estado de firewall, el cual debe estar Activo.

4.12.4 Instalación de OpenSSH.

Secure Shell (SSH), es un protocolo seguro para el manejo remoto de servidores, routers y diferentes equipos. Permite administrar el dispositivo o servidor de red mediante un intérprete de órdenes de forma segura, sin algún entorno gráfico. Este protocolo permite comunicar maquinas sin el uso de contraseñas, sino por medio de la existencia de archivos cifrados, logrando mejorar la seguridad del servidor evitando el uso de contraseñas.

Se instalo y activó desde la terminal de Ubuntu de la siguiente manera:

- *sudo apt-get install openssh-server*
- *sudo systemctl status sshd*

Desde UFW se debió habilitar SSH para poder ser utilizada, así:

- *sudo ufw allow ssh*

4.12.5 Instalación de Nginx.

Es un software de servidor web, proxy inverso, libre y de código abierto. Permite direccionar un cliente, mediante un dominio, a una aplicación en el servidor, permitiendo tener diferentes aplicativos en el mismo servidor. Se instaló de la siguiente manera desde la terminal de Ubuntu:

- *sudo apt install nginx*

Se debió habilitar el perfil Nginx HTTP para permitir el tráfico web normal, no cifrado, por medio del puerto 80, de la siguiente manera:

- *sudo ufw allow 'Nginx HTTP'*

4.12.6 Instalación de PM2.

Es un gestor de procesos en producción para aplicaciones desarrolladas en NodeJS. Internamente cuenta con un balanceador de carga y permite mantener siempre activas las aplicaciones y volver a cargarlas logrando evitar tiempos de inactividad. Permite ejecutar un script sin tener una terminal abierta. Inicia la ejecución automática de los servicios del servidor en el caso que la maquina se reinicie. Se instaló en el servidor de forma global desde la terminal de la siguiente manera:

- *sudo npm install pm2 -g*

4.12.6 Dominio para el Fronted.

La máquina utilizada tiene asignada una IP pública y un subdominio que fue solicitado a la Universidad de Pamplona para este proyecto: *estación.unipamplona.edu.co*, esta es la dirección URL por la cual se accederá al sitio web creado.

4.12.7 Dominio para el Backend.

Debido a la necesidad de manejar un dominio diferente para el backend, se solicitó un dominio de forma gratuita en un sitio web con vigencia de un año (1): estacionunipamplona.edu.co, esta es la dirección a la cual *axios* hace las peticiones HTTP, es el *path* que se envía al crear el *SocketContext* y cada hilo de Python, en la Raspberry Pi 3B+, hace el envío de datos a esta dirección por medio de las peticiones HTTP.

4.12.8 Montaje de archivos al servidor.

Teniendo los archivos en la carpeta *aerogenerador*, en el equipo que funciona como servidor, se realizó el proceso para iniciar la página web. Desde la terminal de Ubuntu se realizó la instalación de todas las dependencias necesarias para el funcionamiento del backend, de la siguiente manera:

- *cd aerogenerador/backend*
- *npm i*

Esta instalación se realizó estando en el mismo nivel del archivo *package.json*, el cual contiene todas las dependencias a instalar. Para ejecutar el servidor del backend ser ejecuto la siguiente instrucción:

- *pm2 start src/index.js --name servidor*

Obteniendo por medio de PM2 un servicio en el puerto 8080 que ejecuta el archivo principal *index.js* bajo el nombre de *servidor*.

Los archivos en cuanto al fronted se montaron de la siguiente manera:

- *cd fronted*
- *npm i*
- *npm run build*

Con esta última instrucción se realiza la construcción final de la aplicación y se crea la carpeta *build* dentro del directorio *fronted*. Se realizó la instalación de *serve*, por medio del cual se sirven sitios estáticos, de la siguiente manera, de forma global:

- *sudo npm i -g serve*

Para subir este el fronted por medio de PM2, se ejecutó la siguiente instrucción:

- *pm2 serve build 3000 --spa --name cliente*

Obteniendo un servicio en el puerto 3000 referente al fronted con el nombre *cliente*.

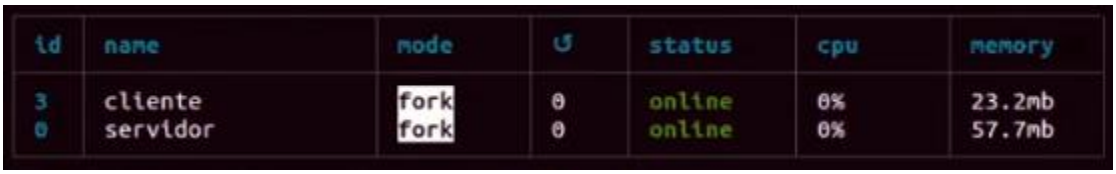

Figura 62. Dos servicios funcionando desde PM2 en la terminal de Ubuntu del servidor.

Fuente: Autor.

4.12.9 Puesta en marcha de la aplicación web.

Para lograr dirigir a cada uno de los puertos dependiendo del dominio, fue necesario manipular el nginx, creando un archivo para el servidor del backend y otro para el del fronted, de la siguiente manera:

- *sudo nano /etc/nginx/sites-available/estacionbackend*

Creando un documento de texto para el servidor del backend con el siguiente contenido:

```
server{
        server_name estacionunipamplona.tk www.estacionunipamplona.tk;
        location / {
                proxy_pass http://localhost:8080;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }
}
```

En el cual se especificó el puerto al cual dirigir al cliente cada vez que ingrese al dominio dado.

Para el fronted se creó el siguiente documento:

- *sudo nano /etc/nginx/sites-available/estacion*

```
server{
        server_name estacion.unipamplona.edu.co www.estacion.unipamplona.edu.co;
        location / {
                proxy_pass http://localhost:3000;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }
}
```

En el cual se especificó el puerto al cual dirigir al cliente cada vez que ingrese al dominio dado. Esta dirección será por la cual ingresarán los clientes al sitio web.

Nginx es su configuración, carga los archivos presentes en la carpeta *sites-enabled*, por esto fue necesario copiar los dos archivos creados anteriormente a este directorio, así:

- *sudo cp /etc/nginx/sites-available/estacion /etc/nginx/sites-enabled*
- *sudo cp /etc/nginx/sites-available/estacionbackend /etc/nginx/sites-enabled*

Para recargar nginx y poder aplicar los cambios se ejecutó la siguiente instrucción:

- *sudo systemctl reload nginx*

Al recargar nginx se aseguró el acceso a la página web desde el dominio *estación.unipamplona.edu.co* con una comunicación activa con el backend.

4.13 Elaboración de histogramas.

4.13.1 Histograma de la velocidad del viento.

En Python, se realizó un algoritmo para determinar el histograma de frecuencia de la velocidad del viento en un momento especifico. Por medio de una petición *POST*, se hace el envío del parámetro *fecha*, cuando se quieren obtener los datos de un día especifico, o se envían los parámetros *fechaIni* y *fechaFin*, cuando los datos que se quieren obtener están en un rango de días especifico.

Las direcciones de la API a las que se realizan estas peticiones son:

- */api/dia-especifico* , para los datos de un día.
- */api/semana-especifica* , para los datos de un rango de días.

Como respuesta del servidor, se obtiene un objeto JSON, del cual se extraen los valores de velocidad del viento y, basado en la cantidad de datos tomados en el lapso de tiempo, se calcula por medio de la regla de Sturges la cantidad de intervalos para el análisis de los datos.

$$k = 1 + 3.322 * \log_{10}(N)$$

Ecuación 3. Regla de Sturges.

En la ecuación anterior:

- *k* es el número de clases o intervalos.
- *N* es el número de muestras o datos.
- *Log* es el logaritmo base 10.

Se aplica aproximación común para tener un número entero de intervalos. Haciendo uso de la librería *matplotlib*, se crea un histograma en barras enviando los datos de la velocidad del viento y la cantidad de intervalos necesaria, obteniendo un histograma que muestra la frecuencia de un valor de velocidad de viento en un tiempo dado. Este código se encuentra en el anexo 20.

4.13.2 Histograma de la potencia producida.

Al igual que para el histograma de velocidad de viento, por medio de una petición *POST* se hace el envío del parámetro *fecha*, cuando se quieren obtener los datos de un día especifico, o se envían los parámetros *fechaIni* y *fechaFin*, cuando los datos que se quieren obtener están en un rango de días especifico.

Las rutas de la API que reciben estas peticiones son las mismas que para el histograma de velocidad de viento. Del JSON que se recibe como respuesta del servidor, se extraen los valores de la potencia producida y se calcula la cantidad de intervalos por medio de la fórmula de Sturges.

Por medio de matplotlib, se realiza el grafico del histograma de la potencia producida enviando los datos recibidos del servidor y la cantidad de intervalos calculada, como se ve en el anexo 21.

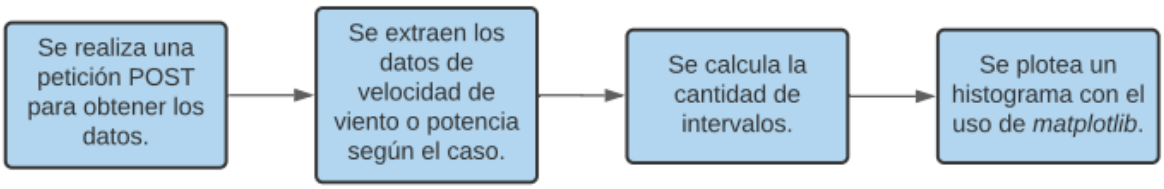

Figura 63. Diagrama de bloques para la creación del histograma de velocidad de viento o potencia producida, en Python.

Fuente: Autor.

4.13.3 Histograma de Temperatura.

Se creó un script de Python, el cual realiza una petición POST al servidor, enviando como parámetro *fecha*, cuando se quieren obtener los datos de un día especifico, o se envían los parámetros *fechaIni* y *fechaFin*, cuando los datos que se quieren obtener están en un rango de días especifico.

Las direcciones de la API que reciben estas peticiones son:

- */api/dia-especifico-temperatura* , para los datos de un día.
- */api/semana-especifica-temperatura* , para los datos de un rango de días.

Del objeto JSON que se recibe como respuesta se extraen los valores correspondientes a la temperatura y se calcula la cantidad de intervalos por medio de la fórmula de Sturges. Dando uso a la librería matplotlib, se grafica el histograma de frecuencia de temperatura en un tiempo específico (anexo 22).

4.13.4 Histograma de Humedad.

Al igual que para la realización del histograma de temperatura, se realiza una petición POST al servidor, enviando como parámetro *fecha*, cuando se quieren obtener los datos de un día especifico, o se envían los parámetros *fechaIni* y *fechaFin*, cuando los datos que se quieren obtener están en un rango de días especifico. Las direcciones de la API son las mismas que se describen en la realización del histograma de temperatura.

Del objeto JSON que se obtiene como respuesta desde el servidor, se extraen los datos de Humedad y se calcula, con la fórmula de Sturges, la cantidad de intervalos. Se genera el histograma por medio de matplolib enviando los datos de humedad en el tiempo dado y la cantidad de intervalos calculada. Este código se encuentra en el anexo 23.

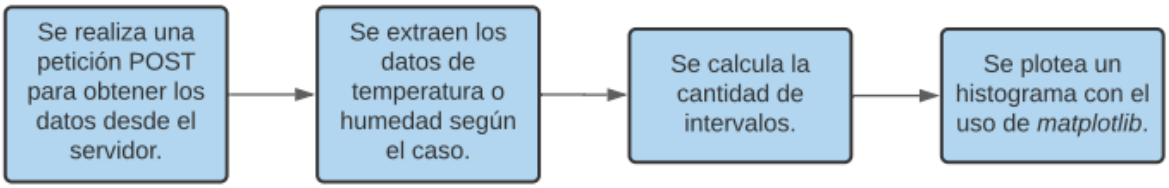

Figura 64. Diagrama de bloques para la creación del histograma de temperatura o humedad, en Python.

Fuente: Autor.

4.13.5 Histograma de velocidad de viento y potencia producida en la última hora.

Debido a que existe una ruta en la API que retorna los datos de la velocidad de viento y potencia producida en la última hora, se realizó en Python un script que realiza una petición GET a la dirección: */api/hora-ultima*, recibiendo un objeto JSON con los datos, ver anexo 24. Con la fórmula de Sturges se calcula la cantidad de intervalos y se crea el grafico enviando como parámetros a matplotlib los datos de velocidad de viento o potencia, según sea el caso, y la cantidad de intervalos.

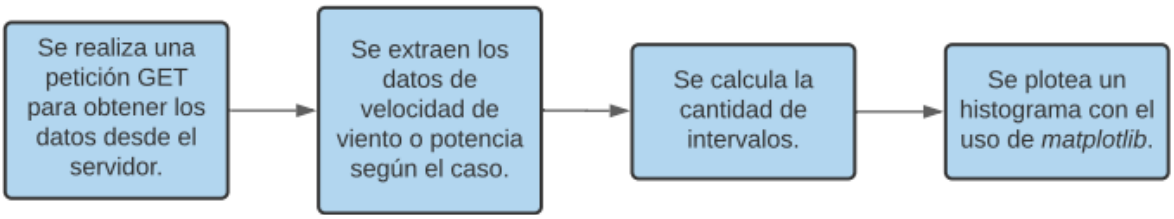

Figura 65. Diagrama de bloques para la creación del histograma de velocidad de viento o potencia producida en la última hora, en Python.

Fuente: Autor.

Capítulo 5. Resultados.

5.1 Servidor para el backend.

Creación de un servicio de API basado en NodeJS capaz de procesar peticiones HTTP, administrar base de datos y con una capa funcional de socket.io para la comunicación de datos en tiempo real con los clientes. Todo este servicio desarrollado con JavaScript. Se desarrolló con Express, debido a que NodeJS está enfocado en la ejecución de JavaScript del lado del servidor, pero no en la creación de sitios web, el framework Express esta propuesta con este fin.

5.2 Servidor para el fronted.

Se construyó una aplicación completa de React, con diferentes componentes renderizados al navegador, y con una comunicación por socket.io en tiempo real para recibir datos y pintarlos en un componente. Realiza peticiones HTTP para adquisición de información de una base de datos. El uso de React aumenta la velocidad de carga de la aplicación web ya que no es necesario recargar la página al navegar en las diferentes vistas, sino que se renderiza un componente especifico de React en cada caso.

5.3 Ensamble de servidores.

Se creó una comunicación entre dos servidores, uno para el backend y otro para el fronted, por medio de socket.io para enviar datos en tiempo real a los clientes y con la opción de realizar peticiones HTTP desde el fronted al backend. Cada servidor implementado en un puerto diferente, con un dominio distinto, pero en la misma máquina con relación constante entre los dos. El fronted cuenta con un monitoreo constante para la conexión con el servidor del backend, si se cae este servicio muestra este estado avisando al cliente, y como advertencia para revisión.

JavaScript fue el lenguaje utilizado para crear el servidor del fronted, del backend y la base de datos, creando un servicio Full Stack JS.

5.4 Base de datos completa.

Por medio de MongoDB, se creó una base de datos la cual contiene información de las variables medioambientales, así como los valores de producción energética de algunas semanas del aerogenerador AIR 40 presente en la sede social Villa Marina de la Universidad de Pamplona. Esta base de datos continúa llenándose por medio de la Raspberry Pi 3B+ y el sistema de adquisición de datos creado. Todos estos datos enviados y guardados en un formato JSON logrando que ser más ligeras las transacciones de datos, más entendible y autodescriptivo.

Desde MongoDB se realizaron las consultas con filtros específicos y muy poco código debido a la gran cantidad de operadores de consulta que tiene y a la gran documentación. Su utilización fue gratuita ya que es una base de datos de código abierto.

5.5 Sistema de visualización de variables en tiempo real.

Asegurando un acceso continuo a internet, se diseñó un sistema basado en Raspberry Pi 3B+, capaz de hacer lectura de variables por medio del lenguaje Python, ejecutando subprocesos independientes o hilos para cada sensor y enviar estos datos por medio de peticiones HTTP a un servidor en tiempo real, todo en un mismo script de ejecución.

Esta comunicación de datos es posible gracias a la capacidad de este modelo de Raspberry Pi de conectarse a internet. En el momento de hacer la lectura del valor de un sensor, este dato se publica en el fronted de la aplicación, solo dependiendo de la velocidad del internet en el lugar. No solo de variables digitales por medio de lo pines GPIO, sino variables análogas por medio del ADC MCP3008, ampliando mucho más la cantidad de sensores que se pueden conectar a este sistema. Esto se logó diseñando una PCB que aloja el ADC y permitiendo la conexión a los sensores utilizados.

Figura 66. Sistema total de adquisición de datos ubicado en el cuarto de control.

Fuente: Autor.

5.6 Gráfica de datos.

Para mostrar el comportamiento de la producción de energía y velocidad del viento respecto al tiempo, en el fronted se generan gráficas que representan los valores recibidos desde la base de datos. Se presentan automáticamente las gráficas de la ultima hora y del ultima día, pero con la opción de establecer la búsqueda solo para los datos de un día especifico como se ve en la siguiente imagen.

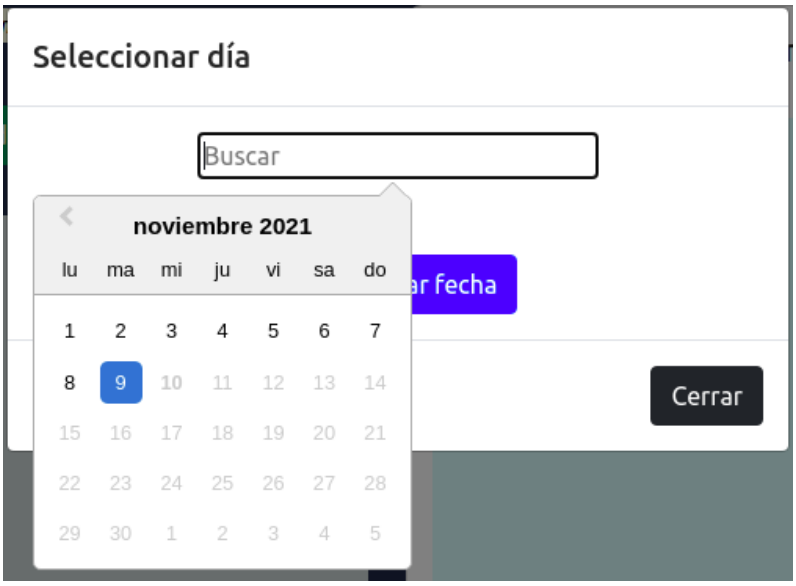

Figura 67. Modal para seleccionar el día de la búsqueda de datos.

Fuente: Autor.

También se puede elegir un rango especifico de búsqueda, un intervalo de días, una semana, un mes, etc., logrando pintar todos estos valores para analizar la producción en el tiempo establecido.

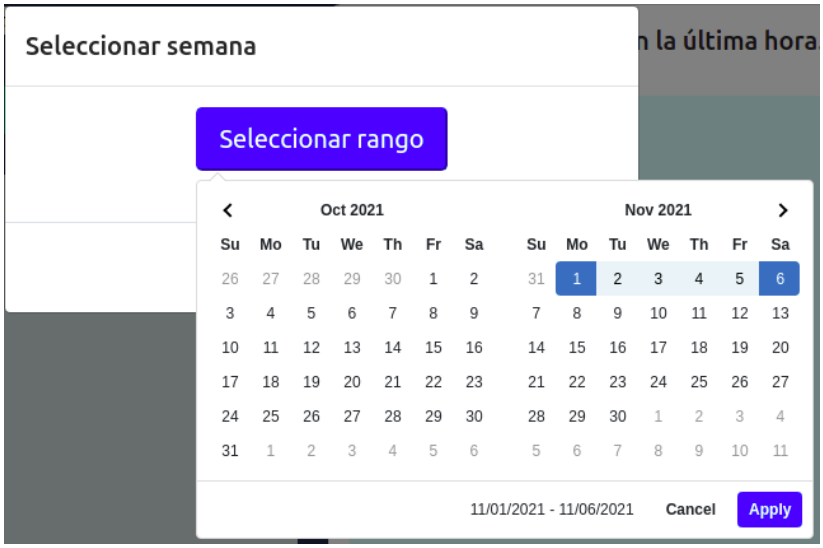

Figura 68. Modal para seleccionar el rango de días para la búsqueda de datos

Fuente: Autor.

5.7 Caracterización aerogenerador.

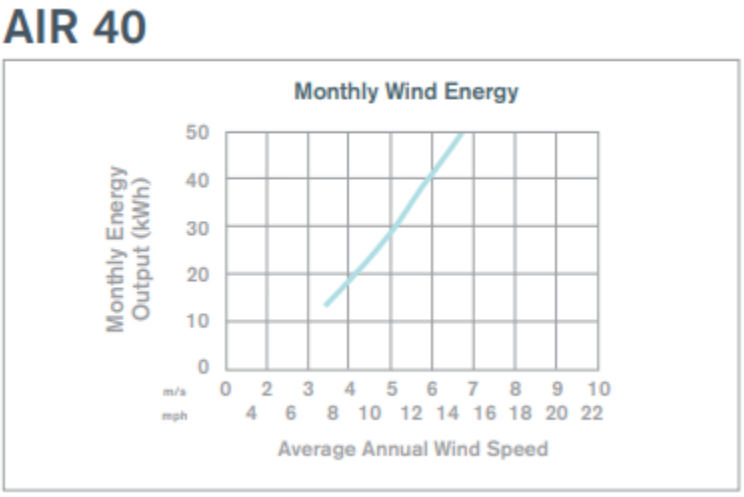

Figura 69. Curva de funcionamiento AIR 40.

Fuente: https://www.technosun.com/descargas/SOUTHWEST-AIR-40-ficha-EN.pdf

De la curva de funcionamiento del aerogenerador AIR40, figura 67, presente en su hoja de características, se generó la ecuación que describe el comportamiento de la curva, obteniendo como resultado:

$$y = 1.1241x2 + 0.0373x - 0.1218$$

Ecuación 4. Ec. Polinomial del aerogenerador AIR40.

Donde *x* representa la velocidad del viento y, *y* la potencia en Kilovatio por hora (kWh). Dando como resultado una línea de tendencia cuadrática y un coeficiente de determinación (R^2) igual a 0.999.

Figura 70. Curva de funcionamiento aerogenerador AIR 40 generada en Excel.

Fuente: Autor.

5.8 Análisis de la velocidad del viento y producción energética en la sede social Villa Marina.

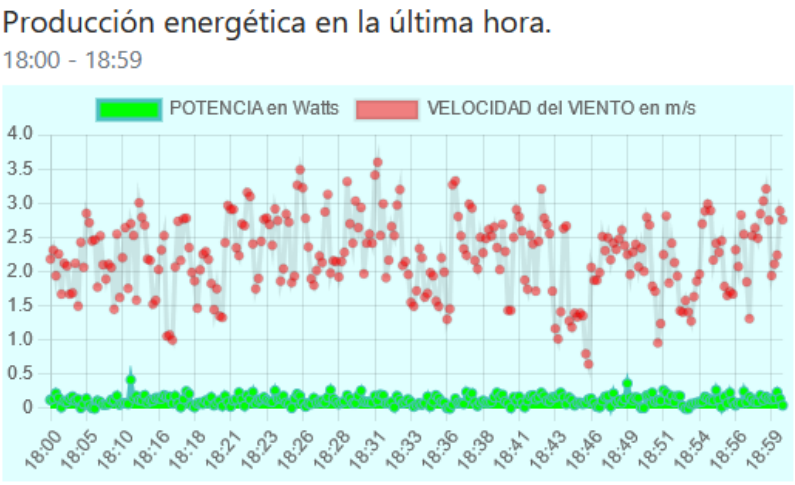

Figura 71. Potencia generada y velocidad del viento en una hora dada.

Fuente: Autor.

De los datos presentados en la gráfica anterior, disponibles en la página web desarrollada, se observa la constante variabilidad en la velocidad del viento, pero con un valor máximo muy pequeño. Según la curva de funcionamiento y la hoja de características del aerogenerador AIR40, se necesita una velocidad de 3.1m/s para tener una producción de energía un poco considerable, ya que se especifica que con una velocidad media de 5.4m/s se genera el voltaje para el cual fue diseñado el aerogenerador, que para este caso es de 12 V.

Figura 72. Histograma de potencia producida en una hora, generado en Python.

Fuente: Autor.

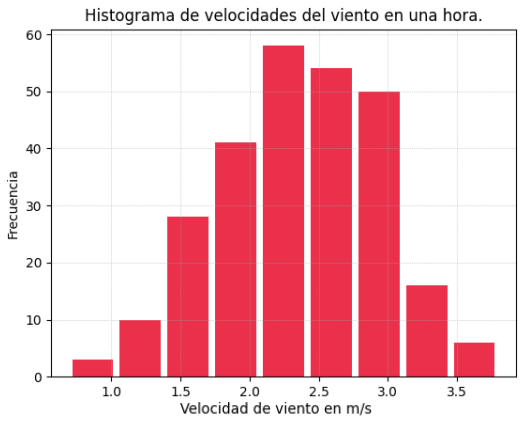

Figura 73. Histograma de velocidades del viento en una hora, generado en Python.

Fuente: Autor.

Para el análisis de una hora, se presenta el mayor porcentaje de velocidades de viento en un rango de 1.5 m/s a 3 m/s.

Para la misma hora, se generó el histograma de frecuencias de la potencia producida por el aerogenerador, en el cual se refleja la poca cantidad de energía que el generador entrega, con el mayor porcentaje de potencias entre 0W a 0.2W.

En todas las horas no se presentan las mismas velocidades de viento, debido a la temperatura y la dirección del viento respecto al aerogenerador, por esto se realizó una ilustración del comportamiento de la velocidad del viento y la potencia producida en un día.

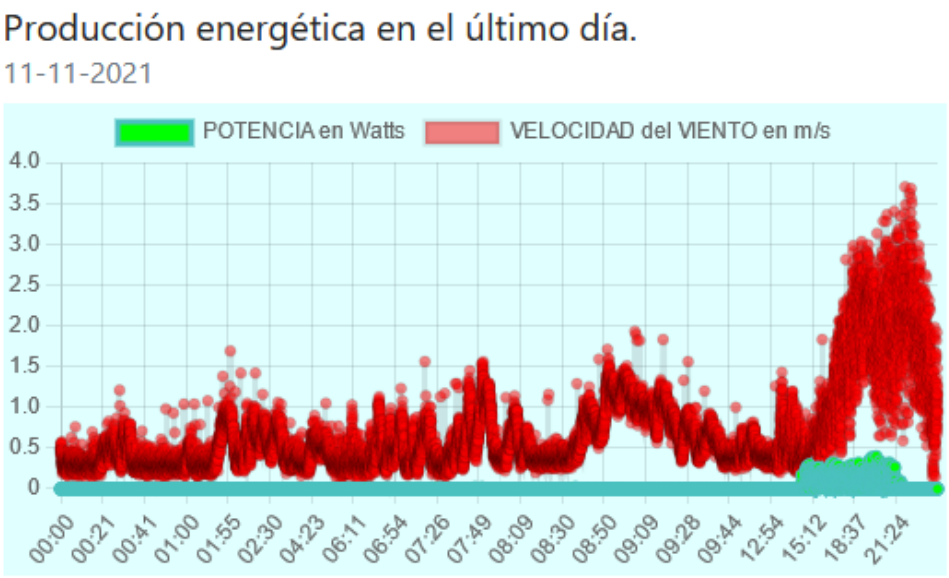

Figura 74. Potencia generada y velocidad del viento en un día dado.

Fuente: Autor.

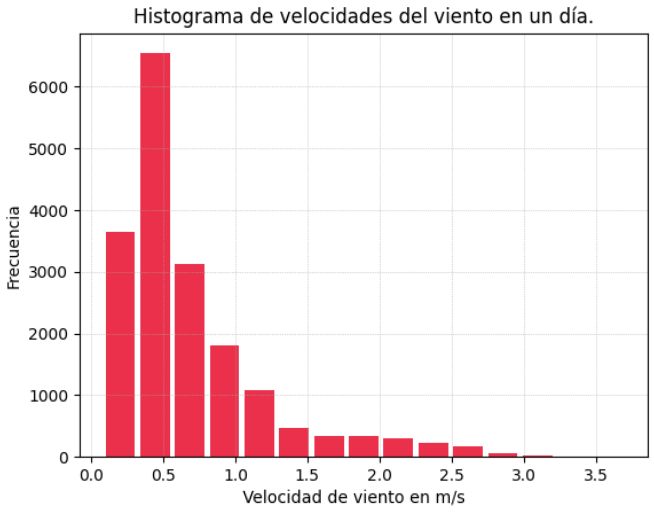

Figura 75. Histograma de velocidades de viento en un día, generado en Python.

Fuente: Autor.

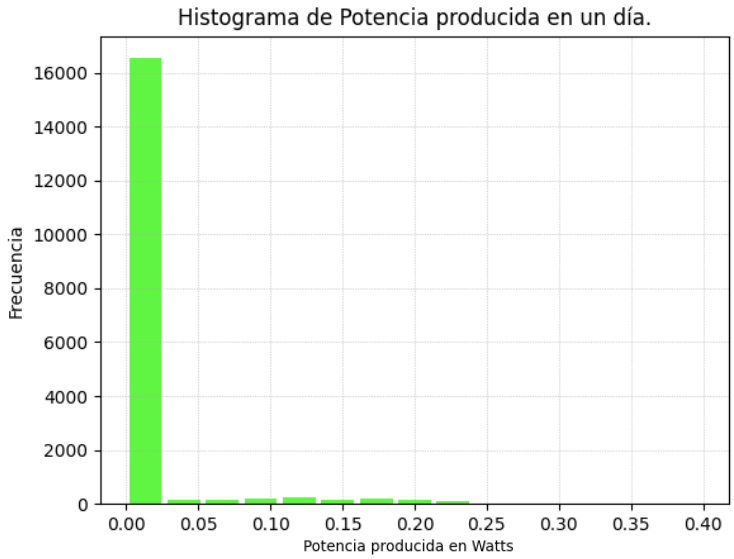

Figura 76. Histograma de potencia producida en un día, generado en Python.

Fuente: Autor.

El mayor porcentaje de velocidades de viento está en un rango de 0.25 m/s a 0.8 m /s, evidenciando lo que se presenta en la figura 72, en la cual se reflejan aumentos de la velocidad del viento en las horas de la tarde, con un máximo de 3.7 m/s, pero según el histograma generado con estos mismos datos, el aumento en ese tiempo de la velocidad del viento no es relevante para la presentada durante un día entero.

Este comportamiento se refleja en la producción de energía en ese día, ya que el mayor porcentaje de potencia que se produjo esta entre 0W y 0.025W, convirtiendo un muy bajos los aportes de energía en los momentos que el aerogenerador está en funcionamiento respecto a los datos de un día entero.

Para el mismo día se analiza el comportamiento de la temperatura y humedad, de la gráfica obtenida por la página web:

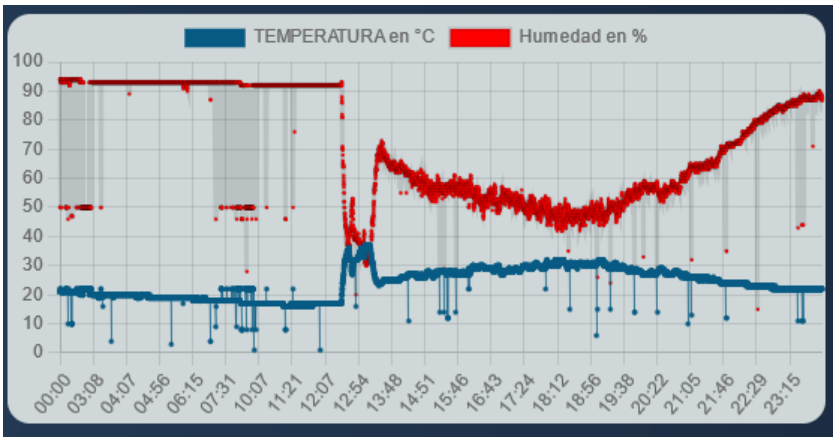

Figura 77. Temperatura y humedad en un día dado.

Fuente: Autor.

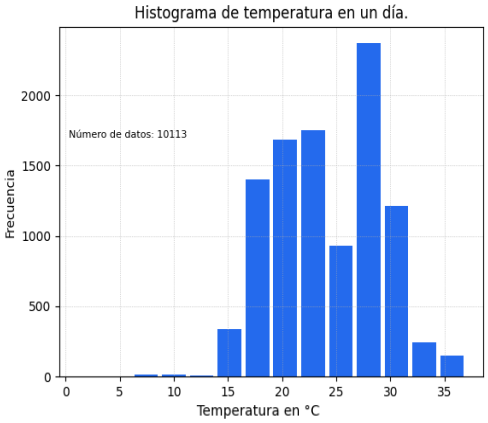

Figura 78. Histograma de temperatura en un día, generado en Python.

Fuente: Autor.

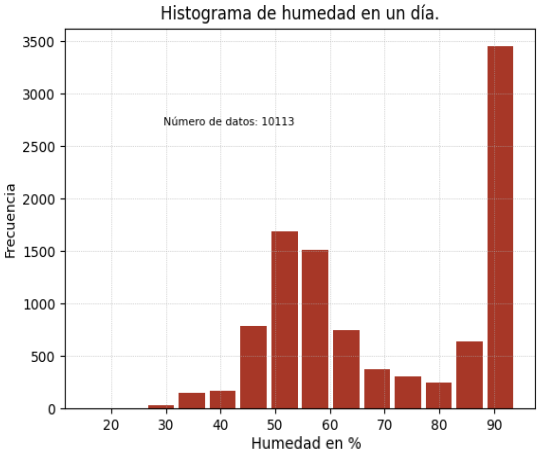

Figura 79. Histograma de humedad en un día, generado en Python.

Fuente: Autor.

Donde se presentan temperaturas iguales o superiores a los 30°C en horas de medio día, con una baja en la humedad relativa proporcional. El mayor porcentaje de temperatura se presenta entre valores de 16°C a 23°C. En el rango de horas entre medio día y las 20:00 pm, la temperatura es superior a los 23°C y es en ese mismo tiempo en que el aerogenerador produjo más potencia.

Para evidenciar el comportamiento de las variables medioambientales y eléctricas en un rango de días se tomó el intervalo del 11 de noviembre al 15 de noviembre del presente año.

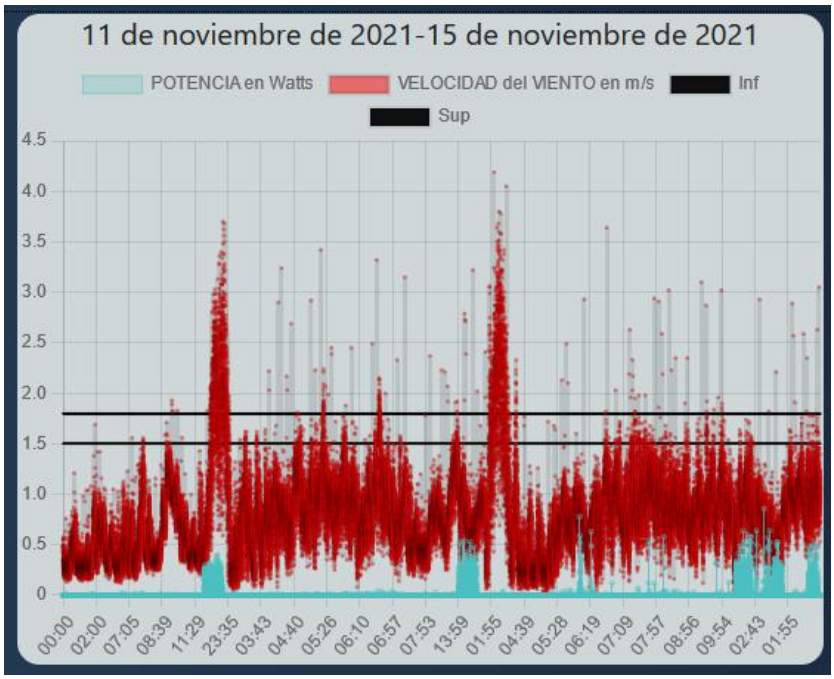

Figura 80. Potencia generada y velocidad del viento en un intervalo de días.

Fuente: Autor.

Como rango de inicio del aerogenerador se estableció de 1.6m/s a 1.8m/s, en el cual se evidencia que el generador produce energía. Para los valores menores a 1.6m/s, no es suficiente la velovidad de viento para iniciar el movimiento del aerogenerador. Los valores de potencia mas altos, en los días analizados, se consiguen entre las 10:00 am y las 3:00 pm. Se evidencia un comportamiento similar en los espectros de cada día.

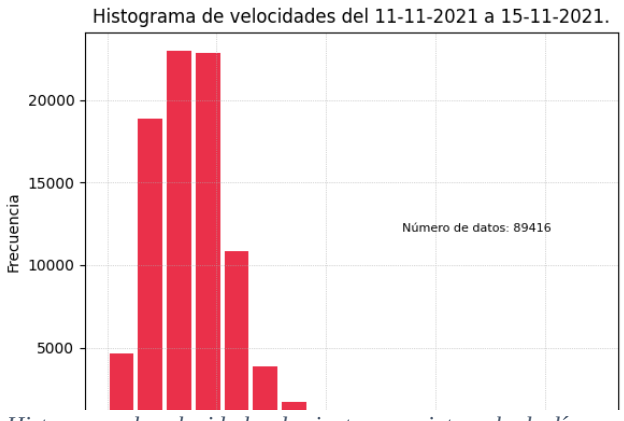

Figura 81. Histograma de velocidades de viento en un intervalo de días, generado en Python.

Fuente: Autor.

Durante los cinco días analizados se presenta un alto porcentaje de velocidades de viento entre los 0.25 m/s a 1.25 m/s. Los valores superiores a 1.8 m/s son los que generan potencia pero presentan poca frecuencia respecto a los demas valores medidos.

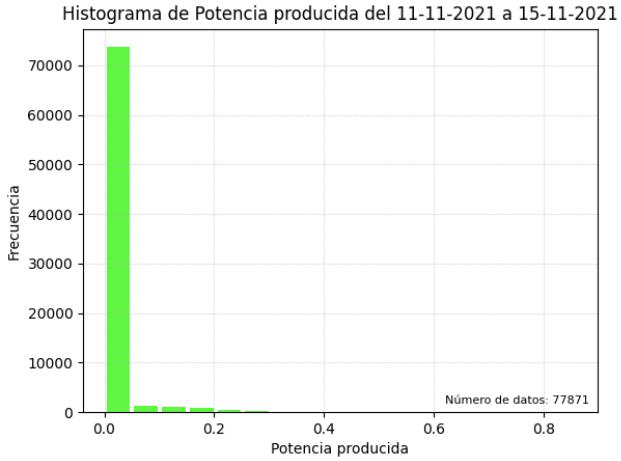

Figura 82. Histograma de potencia producida en un intervalo de días, generado en Python.

Fuente: Autor.

Debido al alto porcentaje de velocidades de viento bajas, aproximadamente el 93% de la potencia que se produce esta comprendida entre valores de 0W a 0.05W.

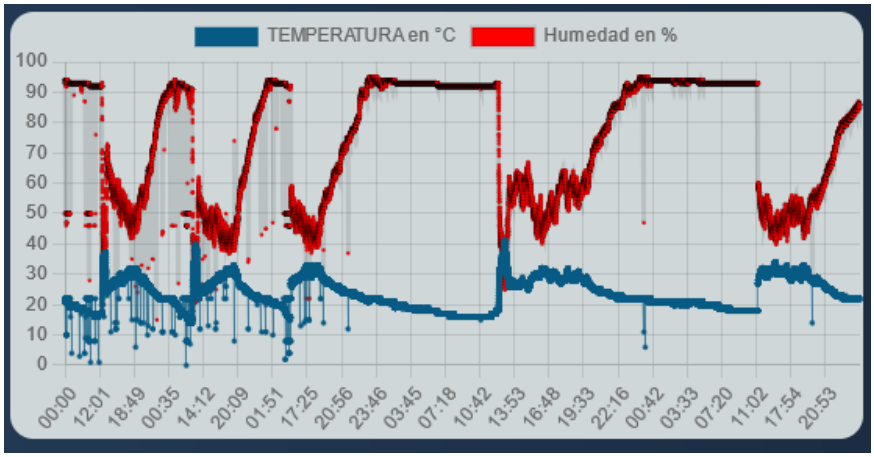

Figura 83. Temperatura y humedad en un intervalo de días.

Fuente: Autor.

El comportamiento de las variables de temperatura y humedad tiene un conducta periodica, ya que cada dia se presenta un comportamiento símilar en las dos curvas. Con temperaturas maximas de 30°C a 34°C entre las 11:00 am y las 3:00pm, y temperaturas mínimas de 14°C a 18°C en horas de la noche con una humedad superior al 90%.

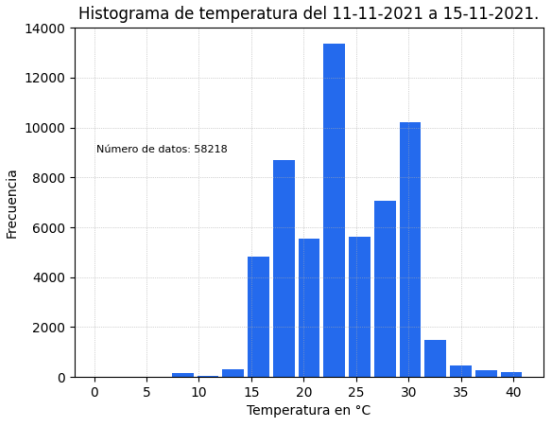

Figura 84. Histograma de temperatura en un intervalo de días, generado en Python.

Fuente: Autor.

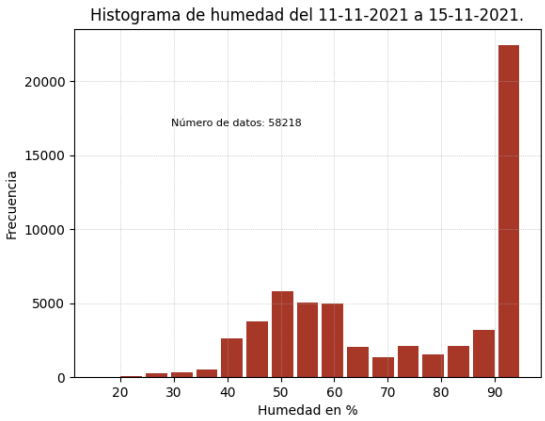

Figura 85. Histograma de humedad en un intervalo de días, generado en Python.

Fuente: Autor.

Las temperaturas mas presentes durantes estos días estan entre 15°C y 31°C. En temperaturas altas se presenta una humedad relativa baja, y en temperaturas bajas, como en horas nocturnas, se presentan valores de humedad altos.

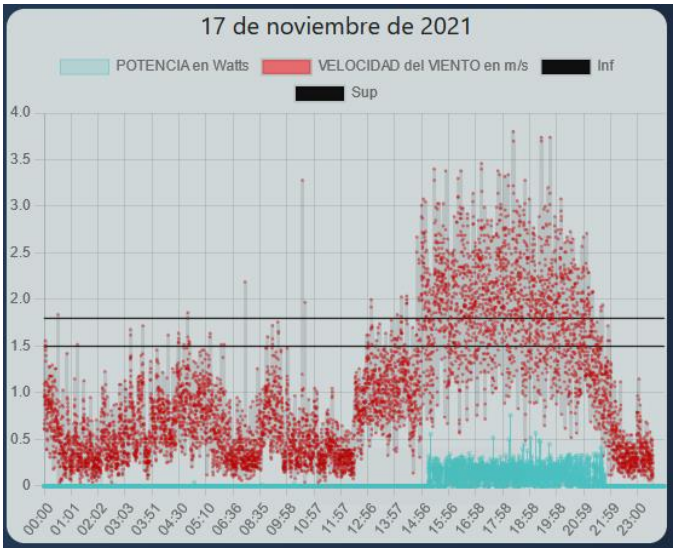

Figura 86. Potencia generada y velocidad del viento en un día dado con límite inferior definido.

Fuente: Autor.

Como se observa en la gráfica anterior, hasta las 21:00 pm se presentan velocidades de viento que hacen generar potencia al aerogenerador. Cuando se supera el límite de 1.8 m/s, empieza la generación de energía desde las 3:00 pm.

Capítulo 6. Conclusiones.

La velocidad promedio para generar el voltaje nominal del aerogenerador según la hoja de características del aerogenerador AIR40 es de 5.4m/s para producir 38kWh/mes, pero la velocidad del arranque es de 3.1m/s. El aerogenerador AIR40 empieza a producir con un rango de velocidad de 1.5 m/s a 1.8 m/s, aumentando el valor de voltaje producido. El mayor porcentaje de velocidad de viento en la sede social Villa Marina se presenta en velocidades de 0.25 m/s a 0.8 m/s, haciendo que la producción generada en velocidades superiores a 1.5m/s sea despreciable debido al porcentaje que esta representa en el total de las velocidades de un día. La producción de energía se presenta en el rango de horas de 10:00 am a 8:00 pm, con intervalos diferentes para cada día. En la gráfica de la producción energética de cada día, disponible en la página web, se pueden observar estos intervalos.

Con una torre base para un aerogenerador, de 10m a 30m de altura, se consiguen velocidades de 5.4m/s en promedio. Ya que el AIR40 en la sede social Villa Marina está en una base con altura de 5m, se reduce la posibilidad de acceso a la velocidad de viento que hará mover las palas del aerogenerador, reduciendo la producción de energía.

El sistema de adquisición de datos instalado, envía una gran cantidad de datos para Mongo DB, en promedio 21 datos por minuto de cada variable, que serían 1260 datos por hora, obteniendo en un día 30240 datos de cada variable. Aunque en la página web se muestran estos datos respecto al tiempo, el análisis es mejor realizarlo respecto a la frecuencia de estos datos con el algoritmo creado en Python para generar las figuras 72 y 73 como ejemplo. Si se reduce la cantidad de datos que se envían a la base de datos se pierde exactitud al graficas ya que la velocidad de viento y la producción energética varían con gran rapidez en el tiempo.

Aunque la producción energética del aerogenerador en la sede social Villa Marina es poca, esta es almacenada en un batería que se encuentra en el cuarto de control, y es complementada por un sistema de panel solar presente en la misma cabaña. Creando un sistema de producción de energía renovable más completo y robusto, evitando faltas de energía debido a la presencia o no de la velocidad de viento suficiente para mover el aerogenerador, supliéndolas con la producción del panel solar.

La temperatura medioambiental en la que se encuentra el sistema de generación de energía eólica en la sede social Villa Marina es en promedio de 29°C en horas diurnas, y de 18°C en horas nocturnas, con un cambio progresivo al transcurrir el tiempo. El aerogenerador está diseñado para funcionar en temperaturas de -10°C a 40°C, por ende, se encuentra en un ambiente propicio para su funcionamiento, evitando daños en sus partes internas y mal funcionamiento del mismo.

La humedad relativa y la temperatura ambiental medidas, tiene un comportamiento altamente proporcional ya que en los tiempos en los que la temperatura se mantiene estable, igual es el comportamiento de la humedad. Después de las 18:00 horas, la temperatura empieza a bajar y la humedad aumenta, durante todas las noches este comportamiento se presenta, llegando a

humedades de más del 90%, por esto un gran porcentaje de la humedad presente en el análisis de un día o varios está en un rango igual o superior a 90%.

El sistema de monitoreo creado para variables medioambientales, demostró ser confiable ya que los valores registrados fueron comparados con sensores análogos para cada variable, y los resultados coincidieron con los enviados en tiempo real a la página web. El sistema creado permitió monitoreo remoto y local ya que la Raspberry Pi cuenta con puerto HDMI para la conexión de un monitor para ver los datos adquiridos por los sensores.

Capítulo 7. Referencias.

[1] Sonia Montecinos, Danilo Carvajal. «Energías renovables: Escenario actual y perspectivas futuras». Universidad de la Serena. Chile. 2018.

[2] Metáfora Visual S.L. Organización cuidemos el planeta. (En línea). Definición de: Energías renovables (https://cuidemoselplaneta.org/energias-renovables/).

[3] Grupo de investigación XUÉ. Semillero de investigación Barión. «Potencial energético eólico para la Región Central». Universidad distrital Francisco José de Caldas. Colombia. 2020.

[4] Fabiola Castellano Gómez. «Estudio de impacto ambiental de un parque eólico en el municipio de Alhama de Murcia». Universidad Politécnica de Cartagena. España. 2018.

[5] C. J. I., A. Bretz, «Full Stack JavaScript development with mean, sitepoint », 2014.

[6] Joan Sebastián Aranda Balaguera. «Fortalecimiento del fronted y backend del sitio web www.vedetucarroya.com.co ». Universidad distrital Francisco José de Caldas. Colombia. 2018.

[7] «Front end development vs Back end development». 2020. Disponible en: https://www.coursereport.com/blog/front-end-development-vs-back-end-development-where-to-start

[8] «Empezando con Node.js». 2019. Disponible en: https://www.manual-informatica.com/

[9] Express, definición. Disponible en: https://expressjs.com/en/resources/glossary.html

[10] «¿Qué son las promesas?». Manz. 2021. Disponible en: https://lenguajejs.com/javascript/asincronia/promesas/

[11] «Generalidades del protocolo HTTP». MDN contributors. 2021. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Overview

[12] «Métodos de petición HTTP». MDN contributors. 2021 Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[13] «What is a database?». 2020. Disponible en: https://blog.airtable.com/what-is-a-database/

[14] A. M, M. Kaufmann. «Sql y NoSql databases». Spring Vieweg. 2019.

[15] «What is SQL database?». Joe Carder. 2020. Disponible en: https://www.openlogic.com/blog/what-sql-database

[16] «What is MongoDB? Introduction, architecture, features & example». 2021. Disponible en: https://www.guru99.com/what-is-mongodb.html

[17] «JSON-Notación de objetos de JavaScript». German Gracia. 2020. Disponible en: http://www.ggctools.com/learn/que-es-json/es

[18] Deimar A. «¿Qué es JSON?». 2020. Disponible en: https://www.hostinger.es/tutoriales/que-es-json

[19] Nora Corina Huacho Suntaxi, Jimmy Daniel Sañaicela Cueva. «Diseño e implementación de una aplicación web adaptativa (servidor de aplicaciones) que controle un sistema domótico cumpliendo el estándar ISO/IEC 25010 para reducir el consumo de energía eléctrica en el hogar». Universidad Politécnica Salesiana sede Quito. Ecuador. 2020.
[20] «What Socket.IO is». 2021. Disponible en: https://socket.io/docs/v4/index.html

[21] Joan Sebastián Aranda Balaguera. «Fortalecimiento del fronted y backend del sitio web www.vedetucarroya.com.co ». Universidad distrital Francisco José de Caldas. Colombia. 2018.

[22] «Conceptos básicos de HTML». MDN contributors. 2021. Disponible en: https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics#anatom%C3%ADa_de_un_elemento_html

[23] «Introducción y conceptos básicos CSS». Universidad Don Bosco. San Salvador. 2020.

[24] Emiliano Migliorata, Juan Ricardo Dahl. «Diseño e implementación de una aplicación web orientada a carpooling aplicando prácticas y tecnologías innovadoras maximizando la eficiencia y calidad del producto». Universidad Nacional del Centro de la Provincia de Buenos Aires. Argentina. 2020.

[25] «Como crear un proyecto de React.js desde 0 | create-react-app». 2020. Disponible en: https://dev.to/duxtech/como-crear-un-proyecto-de-react-js-desde-0-create-react-app-49f6

[26] Raspberry Pi 3B+, definición y especificaciones. Disponibles en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[27] Leandro Gastón Vazquez. «Uso de pines GPIO en Python». Universidad Nacional de la Plata. Argentina. 2021.

[28] Python, definición. Disponible en: https://www.python.org/about/

[29] María Paula Sánchez Fernández, Maicol Patiño Sierra, María Lucía Salazar Torres. Dhaily Zalenny Rico Torres, Karen Juliana Betancourt Ramírez. «Aerogenerador portátil de bicicleta para biciususarios de la ciudad de Bogotá». Universidad distrital Francisco José de Caldas. Colombia. 2020.

[30] Jonathan Xavier Ching Valle, Ángel Gerardo Figueroa Briones. «Diseño y construcción de un aerogenerador de eje vertical para un sistema de iluminación de emergencia con luces LED». Universidad católica de Santiago de Guayaquil. Ecuador. 2017.

[31] Fabian Danilo Fuentes Hernández. «Diseño de un aerogenerador como fuente principal de energía para un clúster de extracción petrolera en Rubiales de Puerto Gaitán». Fundación Universitaria de América. Colombia. 2020.

[32] Jehiner Andrés Rodríguez Loaiza, Daniel Felipe Loaiza Duque. «Diseño de un sistema con energía limpia para el gimnasio de la Universidad de Manizales». Universidad de Manizales. Colombia. 2018.

[33] Mauricio García. «React – Haciendo peticiones con Axios y Hooks». 2020. Disponible en: https://mauriciogc.medium.com/react-haciendo-peticiones-con-axios-y-hooks-27029dc36299

[34] Helder da Rocha. «Learn Chart.js: Create interactive visualizations for the Web with Chart.js 2». Packt. India. 2019.

[35] Óscar Ramírez Jiménez. «Python a fondo, domine el lenguaje del presente y del future». Primera edición. Marcombo. 2021.

[36] Andrés Parra Báez. «Diseño y simulación de un aerogenerador tripala en Boyacá, mediante dinámica de fluidos computacional». Universidad Santo Tomás. Colombia. 2021.

[37] Francisco Eraso Checa, Edisson Escobar Rosero, Diego Fernando Paz, Carlos Morales. «Metodología para la determinación de características del viento y evaluación del potencial de energía eólica en Túquerres - Nariño». Universidad distrital Francisco José de Caldas. Colombia. 2017.

[38] Juan Carlos Agotegaray, Andrea Pinzón, Emanuel Lera. «Automatización de un banco de ensayos de generadores eléctricos para aplicación en energía eólica de baja potencia». Universidad Nacional de General Sarmiento. Argentina. 2020.

[39] Jhon Alexander Guzmán Manrique. «Análisis de la distribución espacial del potencial eólico en el territorio colombiano». Universidad Nacional de Colombia. Colombia. 2021.

Anexos.

Anexo 1. Código en Python para leer el sensor DHT11.

```
import Adafruit_DHT
import asyncio

sensor = Adafruit_DHT.DHT11 #sensor DHT11

pin = 4 #Pin al que se conecto la salida del sensor

while True:
    humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)
    print('Temp={0:0.1f}*C Humedad={1:0.1f}%'.format(temperatura, humedad))
    print('')
```

Anexo 2. Código en Python para obtener la velocidad del viento.

```
import RPi.GPIO as GPIO #Para usar los puertos GPIO
import spidev
from time import sleep

GPIO.setmode(GPIO.BOARD)
spi = spidev.SpiDev()
spi.open(0,0)

#Función para hacer la lectura de un pin analogo
def analogRead(pin):
    spi.max_speed_hz = 1350000
    adc = spi.xfer2([1,(8+pin) << 4,0])
    lec = ((adc[1]&3) << 8) +adc[2]
    return lec

while (True):
    lectura = analogRead(0) #Se lee el sensor
    vol = (lectura*5.00)/1023 #Lectura en un rango de 0-5
    if vol < 0.408:
        vol = 0.4
    v=(20.25*vol)-8.1 #Ecuacion del anemometro
    v = round(v, 2) #Se redondea con 2 cifras
    print('Velocidad = ', v, 'm/s')
    sleep(1)
```

Anexo 3. Código en Python para lectura de voltaje.

```
import RPi.GPIO as GPIO
import spidev
from time import sleep

def analogRead(pin):
   spi.max_speed_hz = 1350000
   adc = spi.xfer2([1,(8+pin) << 4,0])
   lec = ((adc[1]&3) << 8) +adc[2]
   return lec

spi = spidev.SpiDev()
spi.open(0,0)

while(True):
   lectura = analogRead(1) #Lectura del voltaje
   voltaje = (lectura*5.00)/1023 #voltaje entre 0-5V
   voltaje = round (voltaje * 5.00, 1) #voltaje de 0-25V

   corriente = voltaje/100
   corriente = round(corriente, 2) #corriente con dos decimales

   potencia = round (voltaje*corriente, 2) #calculo de potencia

   print('Votaje = ', voltaje, ' V')
   print('Corriente = ', corriente, ' A')
   print('Potencia = ', potencia, ' W')

   print('')
   sleep(1)
```

Anexo 4. Código en Python para la ejecución de tres subprocesos.

```
import threading
import RPi.GPIO as GPIO
import spidev
import Adafruit_DHT

import time

sensor = Adafruit_DHT.DHT11
pin = 4

GPIO.setmode(GPIO.BOARD)
spi = spidev.SpiDev()
spi.open(0,0)
```

```
def sensor_tem_hum():
    while True:
        humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)
        humedad = round(humedad, 2)
        temperatura =  round(temperatura, 2)


        print("")
        print('Temp={0:0.1f}*C Humedad={1:0.1f}%'.format(temperatura, humedad))
        print('')

    return


def analogRead(pin):
    spi.max_speed_hz = 1350000
    adc = spi.xfer2([1,(8+pin) << 4,0])
    lec = ((adc[1]&3) << 8) +adc[2]
    return lec

def sensor_velocidad():
    while (True):
        lectura = analogRead(0)
        vol = (lectura*5.00)/1023
        if vol < 0.408:
            vol = 0.4
        v=(20.25*vol)-8.1
        v = round(v, 2)

        print('')
        print('Velocidad = ', v, 'm/s')
        print('')


        time.sleep(1)
    return

def sensor_voltaje():
    while (True):
        lectura = analogRead(1)
        voltaje = (lectura*5.00)/1023
        voltaje = round (voltaje * 5.00, 1)

        corriente = round (voltaje/100,3)

        potencia = round (voltaje*corriente, 2)
```

```
        print('')
        print('Votaje = ', voltaje, ' V')
        print('Corriente = ', corriente, ' A')
        print('Potencia = ', potencia, ' W')
        print('')

        time.sleep(1)
    return


while True:
    h_tem_hum = threading.Thread(name = "hil_tem_hum", target = sensor_tem_hum)
#subproceso para sensor DHT11
    h_velocidad= threading.Thread(name = "hil_velocidad", target = sensor_velocidad)
#subproceso para velocidad de viento
    h_voltaje= threading.Thread(name = "hil_voltaje", target = sensor_voltaje) #subproceso para
voltaje, corriente y potencia
    #Arranque de los subprocesos
    h_tem_hum.start()
    h_velocidad.start()
    h_voltaje.start()

    h_tem_hum.join()
```

Anexo 5. Código en Python para la ejecución de tres subprocesos con envío de datos al servidor.

```
import threading
import RPi.GPIO as GPIO
import spidev
import Adafruit_DHT
import requests
import json

import time

sensor = Adafruit_DHT.DHT11
pin = 4

GPIO.setmode(GPIO.BOARD)
spi = spidev.SpiDev()
spi.open(0,0)


def sensor_tem_hum():
    humedad_1=50
    temperatura_1=22
    while True:
```

```
        humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)
        humedad = round(humedad, 2)
        temperatura =  round(temperatura, 2)

        if(humedad>100):
            humedad=humedad_1
            temperatura=temperatura_1

        print("")
        params = {'temperatura': temperatura, 'humedad':humedad}
        json.dumps(params)
        response = requests.post('http://estacionunipamplona.tk/api/temp-hum', json = params)
        print(response.text)

        response1 = requests.post('http://estacionunipamplona.tk/api/temp-hum-db', json =
params)
        print(response1.text)



        temperatura_1=temperatura
        humedad_1=humedad

        print("")
    return


def analogRead(pin):
    spi.max_speed_hz = 1350000
    adc = spi.xfer2([1,(8+pin) << 4,0])
    lec = ((adc[1]&3) << 8) +adc[2]
    return lec

def sensor_velocidad():
    i=1
    vs=0
    while (True):
        lectura = analogRead(0)
        vol = (lectura*5.00)/1023
        if vol < 0.408:
            vol = 0.4
        v=(20.25*vol)-8.1
        v = round(v, 2)

        params = {'velocidad': v}
        json.dumps(params)
        response = requests.post('http://estacionunipamplona.tk/api/velocidad', json = params)
        print(response.text)
        print('')
```

```
      vs = vs + v

      if i==5:

         vs = round(vs/5, 2)
         params1 = {'velocidad': vs}

         json.dumps(params1)

         response1 = requests.post('http://estacionunipamplona.tk/api/velocidad-db', json =
params1)
         print(response1.text)
         print('')
         vs=0
         i=0

      i=i+1
      time.sleep(4.0)
   return

def sensor_voltaje():
   i=1
   vls=0
   cs=0
   ps=0
   voltaje_1=1
   corriente_1=0.07
   potencia_1=0.5
   while (True):
      lectura = analogRead(1)
      voltaje = (lectura*5.00)/1023
      voltaje = round (voltaje * 5.00, 1)

      corriente = round (voltaje/100,3) #RESISTENCIA DE LA CARGA

      potencia = round (voltaje*corriente, 2)

      if potencia > 50 :
         voltaje=voltaje_1
         corriente=corriente_1
         potencia=potencia_1

      print('')
      params = {'voltaje': voltaje, 'corriente': corriente, 'potencia': potencia}
      json.dumps(params)
      response = requests.post('http://estacionunipamplona.tk/api/vol-cor-pot', json = params)
      print(response.text)
```

```
        print('')

        vls = vls + voltaje
        cs = cs + corriente
        ps = ps + potencia

        if i==5:
            vls = round(vls/5, 2)
            cs = round(cs/5 ,4)
            ps = round(ps/5 ,2)
            params1 = {'voltaje': vls, 'corriente': cs, 'potencia': ps}
            json.dumps(params1)
            response1 = requests.post('http://estacionunipamplona.tk/api/vol-cor-pot-db', json =
params1)
            print(response1.text)
            print('')
            vls=0
            cs=0
            ps=0

            i=0
        i=i+1
        corriente_1=corriente
        voltaje_1=voltaje
        potencia_1=potencia
        time.sleep(4.0)
    return


while True:
    h_tem_hum = threading.Thread(name = "hil_tem_hum", target = sensor_tem_hum)
    h_velocidad= threading.Thread(name = "hil_velocidad", target = sensor_velocidad)
    h_voltaje= threading.Thread(name = "hil_voltaje", target = sensor_voltaje)

    h_tem_hum.start()
    h_velocidad.start()
    h_voltaje.start()

    h_tem_hum.join()
```

Anexo 6. Archivo del backend: index.js

```
//Se importa el servidor definido
const Server = require('./server');
//Se crea un servidor
const server = new Server();
```

```
module.exports = server;
//Se ejecuta el servidor creado
server.execute();
```

Anexo 7. Archivo del backend: server.js

```
//Se importa express para crear el servidor basado en NodeJS
const express = require('express');
const http = require('http'); //Para crear un servidor para peticiones
HTTP
const socketio = require('socket.io'); //Librería para el servidor con
sockets
const morgan = require('morgan');
const Sockets = require('./socket_server'); //configuraciones definidas
para el socket
const cors = require('cors');

class Server {
    constructor() {
        this.app = express();//Se ejecuta express para generar el
servidor
        this.PORT = 8080;//Número del puerto donde funciona el servidor
        this.server = http.createServer(this.app);

        //Configuracion de sockets
        this.io = socketio(this.server, {cors: {
                origin: "http://localhost:3000",
                methods: ["GET", "POST"],
                credentials: true
        }});

    }

    middlewares(){
        this.app.use(morgan('dev'));//informacion de las peticiones
        this.app.use(express.urlencoded({extended: false})); //para
entender input de formularios que reciba
        this.app.use(express.json()); // para entender objetos JSON que
reciba
        this.app.use( cors() ); //Para intercambiar datos entre
servidores
        this.app.use(function(req, res, next){
            res.header("Access-Control-Allow-Origin", "*");
            res.header("Access-Control-Allow-Headers", "Origin, X-
Requested-With, Content-Type, Accept");
            next();
        });
    }
    //Se definen las rutas por defecto de la aplicación
    routes(){
        this.app.use('/api' , require('./routes/api'));
    }
    //Se importa la configuración de la base de datos
    database(){
        require('./database');
```

```
    }
    //Se ejecuta la configuración del socket
    configurarSockets(){
        this.sockets = new Sockets(this.io);
    }

    execute(){
        this.middlewares();
        this.routes();
        this.database();
        this.configurarSockets();
        //Mensaje en consola cuando el servidor este funcionando
        this.server.listen(this.PORT, () => {
            console.log('Servidor en el puerto: ', this.PORT);
        });
    }

}
//Se exporta el servidor creado
module.exports = Server;
```

Anexo 8. Archivo del backend para configurar el socket: socketServer.js

```
//Se crea la clase para configurar el socket
class Sockets {
    constructor(io) {
        this.io = io;
        this.socketEvents();
    }
    //Se define el evento por defecto al conectar un cliente al socket
    socketEvents(){
        //On connection
        this.io.on('connection', (socket) => {
            console.log('new client connected');
            socket.emit('conectado', 'I am connected with the
backend');
        });

    }
    //Se definen los eventos para emitir desde el socket
    eventos(mensaje, datos){
        if (mensaje == 'datos-temp-hum'){
            this.io.sockets.emit('datos-temp-hum', datos);
        }

        if (mensaje == 'datos-velocidad'){
            this.io.sockets.emit('datos-velocidad', datos);
        }

        if (mensaje == 'datos-vol-cor-pot'){
            this.io.sockets.emit('datos-vol-cor-pot', datos);
        }
    }
```

```
}
//Se exporta la clase
module.exports = Sockets;
```

Anexo 9. Conexión y configuración de la base de datos: database.js

```
const mongoose = require('mongoose'); //modulo para conectarse a
mongoDB

const URI = 'mongodb://localhost/datos-monitoreo'; //dirección de la
base de datos y nombre

//modula para conectarse
mongoose.connect(URI, {
    useCreateIndex: true,
    useNewUrlParser: true,
    useFindAndModify: false,
    useUnifiedTopology: true
})
    .then(db => console.log('DB está conectada'))//mensaje por consola
al conectar la base de datos
    .catch(err => console.error(err));
//Se exporta la configuracion de la base de datos
module.exports = mongoose;
```

Anexo 10. Creación de la API.

```
const {Router} = require('express');//módulo de express para rutas
const router = Router();//se ejecuta el módulo
//Se importan los controladores de rutas creados
const {getVelocidad, postVelocidad, postVelocidadDB} =
require('../controllers/velocidad.controller');
const {getTempHum, postTempHum, postTempHumDB} =
require('../controllers/temp-hum.controller');
const {getVolCorPot, postVolCorPot, postVolCorPotDB} =
require('../controllers/vol-cor-pot.controller');
const {getUltima} = require('../controllers/ultima.controller');
const {postDiaEspecifico} = require('../controllers/dia-especifico');
const {getDiaUltimo} = require ('../controllers/dia-ultimo');
const {postSemana} = require('../controllers/semana');
const {getHoraUltima} = require('../controllers/hora-ultima');
const {postDiaEspecificoTemperatura} = require('../controllers/dia-
especifico-temperatura');
const {postSemanaTemperatura} = require('../controllers/semana-
temperatura');


router.route('/')
    .get((req, res) => res.send('INICIO API EN EXPRESS'))
    .post()
//Controlador para publicar los datos de velocidad de viento en tiempo
real
router.route('/velocidad')
    .get(getVelocidad)
```

```
    .post(postVelocidad)
router.route('/velocidad-db')
    .post(postVelocidadDB)//Controlador para guardar los datos en la
base de datos

//Controlador para publicar los datos de temperatura y humedad en
tiempo real
router.route('/temp-hum')
    .get(getTempHum)
    .post(postTempHum)
router.route('/temp-hum-db')
    .post(postTempHumDB)//Controlador para guardar los datos en la base
de datos

//Controlador para publicar los datos de voltaje, corriente y potencia
en tiempo real
router.route('/vol-cor-pot')
    .get(getVolCorPot)
    .post(postVolCorPot)
router.route('/vol-cor-pot-db')
    .post(postVolCorPotDB)//Controlador para guardar los datos en la
base de datos

router.route('/ultimos-datos')
    .get(getUltima)
//Ruta para enviar los datos de la consulta de un día específico
router.route('/dia-especifico')
    .post(postDiaEspecifico)
//Ruta para enviar los datos del último día
router.route('/dia-ultimo')
    .get(getDiaUltimo)
//Ruta para enviar los datos de la consulta de un rango de días
especifico
router.route('/semana-especifica')
    .post(postSemana)
//Ruta para enviar los datos de la última hora
router.route('/hora-ultima')
    .get(getHoraUltima)
//Ruta para enviar los datos de temperatura y humedad de la consulta de
un día específico
router.route('/dia-especifico-temperatura')
    .post(postDiaEspecificoTemperatura)
//Ruta para enviar los datos de temperatura y humedad de la consulta de
un rango de días específico
router.route('/semana-especifica-temperatura')
    .post(postSemanaTemperatura)
//Se exporta las configuraciones dentro de la variable router
module.exports = router;
```

Anexo 11. Modelos de datos para Mongo DB.

Anexo 11.1 Modelo de datos para temperatura y humedad.

```
const {Schema, model}  = require('mongoose');//modulos para crear un
esquema y modelo de datos
//Se define un esquema de datos para guardar las variables temperatura
```

```
y humedad
const temp_hum_schema = new Schema({
    temperatura:{type: Number, required: true},
    humedad: {type: Number, required: true},
    date:{
        type: Date,
        default: Date.now //Se asigna la fecha por defecto del momento
en que se crea el dato.

    }
}, {
    timestamps: true
});
//Se exporta el modelo de dato
module.exports = model('temp_hum', temp_hum_schema);
```

Anexo 11.2 Modelo de datos para velocidad de viento.

```
const {Schema, model} = require('mongoose');//modulos para crear un
esquema y modelo de datos
//Se define un esquema de datos para guardar la variable velocidad de
viento
const velocidadSchema = new Schema({
    velocidad: Number,
    date:{
        type: Date,
        default: Date.now //Se asigna la fecha por defecto del momento
en que se crea el dato.

    }
}, {
    timestamps: true
});
//Se exporta el modelo de dato
module.exports = model('velocidad', velocidadSchema);
```

Anexo 11.3 Modelo de datos para voltaje, corriente y potencia.

```
const {Schema, model} = require('mongoose');//modulos para crear un
esquema y modelo de datos
//Se define un esquema de datos para guardar las variables voltaje,
corriente y potencia
const volCorPotSchema = new Schema({
    voltaje: Number,
    corriente: Number,
    potencia: Number,
    date:{
        type: Date,
        default: Date.now//Se asigna la fecha por defecto del momento
en que se crea el dato.

    }
}, {
    timestamps: true
});
//Se exporta el modelo de dato
module.exports = model('volCorPot', volCorPotSchema);
```

Anexo 12. Controladores para las rutas de la API.

Anexo 12.1 Controlador para la ruta */dia-especifico*.

```
//Recibe la petición POST para enviar datos de un día específico
const diaEspecificoCtrl = {};//Se crea el controlador
const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const volCorPotDB = require('../models/datos-vol-cor-pot');
const veloci = require('../models/datos-velocidad');

//Se crea un método en el controlador
diaEspecificoCtrl.postDiaEspecifico = async (req,res) => {
    console.log(req.body.params.fecha);
    const fecha = req.body.params.fecha;//se extrae la fecha recibida
en la petición
    //Se generan los límites de busqueda para la base de datos
    const fechaIni = "20"+fecha.charAt(2)+fecha.charAt(3)+"-
"+fecha.charAt(5)+fecha.charAt(6)+"-
"+fecha.charAt(8)+fecha.charAt(9)+"T00:00:00.000Z";
    const fechaFin = "20"+fecha.charAt(2)+fecha.charAt(3)+"-
"+fecha.charAt(5)+fecha.charAt(6)+"-
"+fecha.charAt(8)+fecha.charAt(9)+"T23:59:59.999Z";

    console.log("Fecha Dia inicial: " + fechaIni);
    console.log("Fecha Dia final: " + fechaFin);


    //Consulta a la base de datos para potencia
    const datosDiaPot = await volCorPotDB.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let potenciaDia = [];
    let horasDiaPot = [];
    let horaGeneralPot;
    let horaPot;
    for (let i = 0; i<datosDiaPot.length; i++){
        potenciaDia[i] = datosDiaPot[i].potencia;
        horaGeneralPot = JSON.stringify(datosDiaPot[i].date);
        horaPot =
horaGeneralPot.charAt(12)+horaGeneralPot.charAt(13)+':'+horaGeneralPot.
charAt(15)+horaGeneralPot.charAt(16);
        horasDiaPot[i] = horaPot;
    };

    //Consulta a la base de datos para velocidad del viento
    const datosDiaVel = await veloci.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
```

```
    });

    let velocidadDia = [];
    let horasDiaVel = [];
    let horaGeneralVel;
    let horaVel;
    for (let i = 0; i<datosDiaVel.length; i++){
        velocidadDia[i] = datosDiaVel[i].velocidad;
        horaGeneralVel = JSON.stringify(datosDiaVel[i].date);
        horaVel =
horaGeneralVel.charAt(12)+horaGeneralVel.charAt(13)+':'+horaGeneralVel.
charAt(15)+horaGeneralVel.charAt(16);
        horasDiaVel[i] = horaVel;
    };

    /*console.log("POTENCIA: ");
    console.log(datosDiaPot);
    console.log(potenciaDia);
    console.log(horasDiaPot);
    console.log("VELOCIDAD: ");
    console.log(datosDiaVel);
    console.log(velocidadDia);
    console.log(horasDiaVel);*/
    //Se crea el arreglo con los datos a enviar
    let envio = {potenciaDia, horasDiaPot, velocidadDia, horasDiaVel};
    //Se envía la respuesta al cliente
    res.send(envio);
};
//Se exporta el controlador
module.exports = diaEspecificoCtrl;
```

Anexo 12.2 Controlador para la ruta */dia-especifico-temperatura.*

```
//Recibe la petición POST para enviar datos de temperatura y humedad de
un día específico
const diaEspecificoTemperaturaCtrl = {};
const server = require('../index');
//Se requieren los modelos de datos de la base de datos
const tempHumDB = require('../models/datos-temperatura-humedad');
//Se crea un método en el controlador
diaEspecificoTemperaturaCtrl.postDiaEspecificoTemperatura = async
(req,res) => {
    const fecha = req.body.params.fecha; //se extrae la fecha recibida
en la petición
    //Se generan los límites de busqueda para la base de datos
    const fechaIni = "20"+fecha.charAt(2)+fecha.charAt(3)+"-
"+fecha.charAt(5)+fecha.charAt(6)+"-
"+fecha.charAt(8)+fecha.charAt(9)+"T00:00:00.000Z";
    const fechaFin = "20"+fecha.charAt(2)+fecha.charAt(3)+"-
"+fecha.charAt(5)+fecha.charAt(6)+"-
"+fecha.charAt(8)+fecha.charAt(9)+"T23:59:59.999Z";

    console.log("Fecha Dia inicial TEMP: " + fechaIni);
    console.log("Fecha Dia final TEMP: " + fechaFin);


    //Consulta a la base de datos para temperatura y humedad
```

```
    const datosDiaTemp = await tempHumDB.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let tempDia = [];
    let horasDiaTemp = [];
    let horaGeneralTemp;
    let horaTemp;
    let humDia = [];
    let horasDiaHum = [];
    let horaGeneralHum;
    let horaHum;
    for (let i = 0; i<datosDiaTemp.length; i++){
        tempDia[i] = datosDiaTemp[i].temperatura;
        horaGeneralTemp = JSON.stringify(datosDiaTemp[i].date);
        horaTemp =
horaGeneralTemp.charAt(12)+horaGeneralTemp.charAt(13)+':'+horaGeneralTe
mp.charAt(15)+horaGeneralTemp.charAt(16);
        horasDiaTemp[i] = horaTemp;
        humDia[i] = datosDiaTemp[i].humedad;
        horaGeneralHum = JSON.stringify(datosDiaTemp[i].date);
        horaHum =
horaGeneralHum.charAt(12)+horaGeneralHum.charAt(13)+':'+horaGeneralHum.
charAt(15)+horaGeneralHum.charAt(16);
        horasDiaHum[i] = horaHum;
    };

    /*
    console.log("TEMPERATURA: ");
    console.log(tempDia);
    console.log(horasDiaTemp);
    console.log("HUMEDAD: ");
    console.log(humDia);
    console.log(horasDiaHum);*/
    //Se crea el arreglo con los datos a enviar
    let envio = {tempDia, horasDiaTemp, humDia, horasDiaHum};
    //Se envía la respuesta al cliente
    res.send(envio);
};
//Se exporta el controlador
module.exports = diaEspecificoTemperaturaCtrl;
```

Anexo 12.3 Controlador para la ruta */dia-ultimo*.

```
//Recibe la petición GET para enviar datos del último día
let moment = require('moment'); //Para generar la fecha automaticamente
ya que no se recibe nada del cliente.
const diaUltimoCtrl = {};//Se crea el controlador

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const volCorPotDB = require('../models/datos-vol-cor-pot');
const veloci = require('../models/datos-velocidad');
```

```
//Se crea un método en el controlador
diaUltimoCtrl.getDiaUltimo = async (req,res) => {
    //Se resta un día a la fecha actual
    let fechaIni = moment().subtract(1,'days').format();
    let fechaFin = moment().subtract(1,'days').format();

    //Se generan los límites de busqueda para la base de datos
    fechaIni = "20"+fechaIni.charAt(2)+fechaIni.charAt(3)+"-
"+fechaIni.charAt(5)+fechaIni.charAt(6)+"-
"+fechaIni.charAt(8)+fechaIni.charAt(9)+"T00:00:00.000Z";
    fechaFin = "20"+fechaFin.charAt(2)+fechaFin.charAt(3)+"-
"+fechaFin.charAt(5)+fechaFin.charAt(6)+"-
"+fechaFin.charAt(8)+fechaFin.charAt(9)+"T23:59:59.999Z";

    console.log('Fecha DIA ULTIMO inicial: ' + fechaIni);
    console.log('Fecha DIA ULTIMO final: ' + fechaFin);

    //Consulta a la base de datos para potencia
    const datosDiaPot = await volCorPotDB.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let potenciaDia = [];
    let horasDiaPot = [];
    let horaGeneralPot;
    let horaPot;
    for (let i = 0; i<datosDiaPot.length; i++){
        potenciaDia[i] = datosDiaPot[i].potencia;
        horaGeneralPot = JSON.stringify(datosDiaPot[i].date);
        horaPot =
horaGeneralPot.charAt(12)+horaGeneralPot.charAt(13)+':'+horaGeneralPot.
charAt(15)+horaGeneralPot.charAt(16);
        horasDiaPot[i] = horaPot;
    };

    //Consulta a la base de datos para velocidad del viento
    const datosDiaVel = await veloci.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let velocidadDia = [];
    let horasDiaVel = [];
    let horaGeneralVel;
    let horaVel;
    for (let i = 0; i<datosDiaVel.length; i++){
        velocidadDia[i] = datosDiaVel[i].velocidad;
        horaGeneralVel = JSON.stringify(datosDiaVel[i].date);
        horaVel =
horaGeneralVel.charAt(12)+horaGeneralVel.charAt(13)+':'+horaGeneralVel.
charAt(15)+horaGeneralVel.charAt(16);
```

```
        horasDiaVel[i] = horaVel;
    };
    //Se crea el arreglo con los datos a enviar
    let envio = {potenciaDia, horasDiaPot, velocidadDia, horasDiaVel};
    console.log('Último día terminado');
    //Se envía la respuesta al cliente
    res.send(envio);
};
//Se exporta el controlador
module.exports = diaUltimoCtrl;
```

Anexo 12.4 Controlador para la ruta */hora-ultima*.

```
//Recibe la petición GET para enviar datos del último día
let moment = require('moment'); //Para generar la fecha automaticamente
ya que no se recibe nada del cliente.
const diaUltimoCtrl = {};//Se crea el controlador

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const volCorPotDB = require('../models/datos-vol-cor-pot');
const veloci = require('../models/datos-velocidad');

//Se crea un método en el controlador
diaUltimoCtrl.getDiaUltimo = async (req,res) => {
    //Se resta un día a la fecha actual
    let fechaIni = moment().subtract(1,'days').format();
    let fechaFin = moment().subtract(1,'days').format();

    //Se generan los límites de busqueda para la base de datos
    fechaIni = "20"+fechaIni.charAt(2)+fechaIni.charAt(3)+"-
"+fechaIni.charAt(5)+fechaIni.charAt(6)+"-
"+fechaIni.charAt(8)+fechaIni.charAt(9)+"T00:00:00.000Z";
    fechaFin = "20"+fechaFin.charAt(2)+fechaFin.charAt(3)+"-
"+fechaFin.charAt(5)+fechaFin.charAt(6)+"-
"+fechaFin.charAt(8)+fechaFin.charAt(9)+"T23:59:59.999Z";

    console.log('Fecha DIA ULTIMO inicial: ' + fechaIni);
    console.log('Fecha DIA ULTIMO final: ' + fechaFin);

    //Consulta a la base de datos para potencia
    const datosDiaPot = await volCorPotDB.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let potenciaDia = [];
    let horasDiaPot = [];
    let horaGeneralPot;
    let horaPot;
    for (let i = 0; i<datosDiaPot.length; i++){
        potenciaDia[i] = datosDiaPot[i].potencia;
        horaGeneralPot = JSON.stringify(datosDiaPot[i].date);
        horaPot =
```

```
horaGeneralPot.charAt(12)+horaGeneralPot.charAt(13)+':'+horaGeneralPot.
charAt(15)+horaGeneralPot.charAt(16);
        horasDiaPot[i] = horaPot;
    };

    //Consulta a la base de datos para velocidad del viento
    const datosDiaVel = await veloci.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let velocidadDia = [];
    let horasDiaVel = [];
    let horaGeneralVel;
    let horaVel;
    for (let i = 0; i<datosDiaVel.length; i++){
        velocidadDia[i] = datosDiaVel[i].velocidad;
        horaGeneralVel = JSON.stringify(datosDiaVel[i].date);
        horaVel =
horaGeneralVel.charAt(12)+horaGeneralVel.charAt(13)+':'+horaGeneralVel.
charAt(15)+horaGeneralVel.charAt(16);
        horasDiaVel[i] = horaVel;
    };
    //Se crea el arreglo con los datos a enviar
    let envio = {potenciaDia, horasDiaPot, velocidadDia, horasDiaVel};
    console.log('Último día terminado');
    //Se envía la respuesta al cliente
    res.send(envio);
};
//Se exporta el controlador
module.exports = diaUltimoCtrl;
```

Anexo 12.5 Controlador para la ruta */semana-especifica*.

```
//Recibe la petición POST para enviar datos de un rango de días
específico
const semanaCtrl = {};//Se crea el controlador

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const volCorPotDB = require('../models/datos-vol-cor-pot');
const veloci = require('../models/datos-velocidad');

//Se crea un método en el controlador
semanaCtrl.postSemana = async (req,res) => {
    const fechaInicial = req.body.params.fechaIni;//Se extrae la fecha
inicial enviada desde el cliente
    const fechaFinal = req.body.params.fechaFin;//Se extrae la fecha
final enviada desde el cliente

    //Se generan los límites de busqueda para la base de datos
    const fechaIni =
"20"+fechaInicial.charAt(2)+fechaInicial.charAt(3)+"-
"+fechaInicial.charAt(5)+fechaInicial.charAt(6)+"-
```

```
"+fechaInicial.charAt(8)+fechaInicial.charAt(9)+"T00:00:00.000Z";
    const fechaFin = "20"+fechaFinal.charAt(2)+fechaFinal.charAt(3)+"-
"+fechaFinal.charAt(5)+fechaFinal.charAt(6)+"-
"+fechaFinal.charAt(8)+(fechaFinal.charAt(9)-1)+"T23:59:59.999Z";

    console.log("Fecha Semana inicial: " + fechaIni);
    console.log("Fecha Semana final: " + fechaFin);

    //Consulta a la base de datos para potencia
    const datosSemanaPot = await volCorPotDB.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let potenciaSemana = [];
    let horasSemanaPot = [];
    let horaGeneralPot;
    let horaPot;
    for (let i = 0; i<datosSemanaPot.length; i++){
        potenciaSemana[i] = datosSemanaPot[i].potencia;
        horaGeneralPot = JSON.stringify(datosSemanaPot[i].date);
        horaPot =
horaGeneralPot.charAt(12)+horaGeneralPot.charAt(13)+':'+horaGeneralPot.
charAt(15)+horaGeneralPot.charAt(16);
        horasSemanaPot[i] = horaPot;
    };

    //Consulta a la base de datos para velocidad del viento
    const datosSemanaVel = await veloci.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let velocidadSemana = [];
    let horasSemanaVel = [];
    let horaGeneralVel;
    let horaVel;
    for (let i = 0; i<datosSemanaVel.length; i++){
        velocidadSemana[i] = datosSemanaVel[i].velocidad;
        horaGeneralVel = JSON.stringify(datosSemanaVel[i].date);
        horaVel =
horaGeneralVel.charAt(12)+horaGeneralVel.charAt(13)+':'+horaGeneralVel.
charAt(15)+horaGeneralVel.charAt(16);
        horasSemanaVel[i] = horaVel;
    };

    /*console.log("POTENCIA: ");
    console.log(datosSemanaPot);
    console.log(potenciaSemana);
    console.log(horasSemanaPot);
    console.log("VELOCIDAD: ");
    console.log(datosSemanaVel);
    console.log(velocidadSemana);
```

```
    console.log(horasSemanaVel);*/

    //Se crea el arreglo con los datos a enviar
    let envio = {potenciaSemana, horasSemanaPot, velocidadSemana,
horasSemanaVel};
    //Se envía la respuesta al cliente
    res.send(envio);
};
//Se exporta el controlador
module.exports = semanaCtrl;
```

Anexo 12.6 Controlador para la ruta */semana-especifica-temperatura*.

```
//Recibe la petición POST para enviar datos de temperatura y humedad de
un rango de días específico
const semanaTemperaturaCtrl = {};

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const tempHumDB = require('../models/datos-temperatura-humedad');

//Se crea un método en el controlador
semanaTemperaturaCtrl.postSemanaTemperatura = async (req,res) => {
    const fechaInicial = req.body.params.fechaIni;//Se extrae la fecha
inicial enviada desde el cliente
    const fechaFinal = req.body.params.fechaFin;//Se extrae la fecha
final enviada desde el cliente

    //Se generan los límites de busqueda para la base de datos
    const fechaIni =
"20"+fechaInicial.charAt(2)+fechaInicial.charAt(3)+"-
"+fechaInicial.charAt(5)+fechaInicial.charAt(6)+"-
"+fechaInicial.charAt(8)+fechaInicial.charAt(9)+"T00:00:00.000Z";
    const fechaFin = "20"+fechaFinal.charAt(2)+fechaFinal.charAt(3)+"-
"+fechaFinal.charAt(5)+fechaFinal.charAt(6)+"-
"+fechaFinal.charAt(8)+(fechaFinal.charAt(9)-1)+"T23:59:59.999Z";

    console.log("Fecha Semana inicial TEMP: " + fechaIni);
    console.log("Fecha Semana final TEMP: " + fechaFin);

    //Consulta a la base de datos para temperatura y humedad
    const datosSemanaTemp = await tempHumDB.find({
        $and: [
            {"date": {$gte: fechaIni}},
            {"date": {$lte: fechaFin}}
        ]
    });

    let tempSemana = [];
    let horasSemanaTemp = [];
    let horaGeneralTemp;
    let horaTemp;
    let humSemana = [];
    let horasSemanaHum = [];
    let horaGeneralHum;
    let horaHum;
```

```
    for (let i = 0; i<datosSemanaTemp.length; i++){
        tempSemana[i] = datosSemanaTemp[i].temperatura;
        horaGeneralTemp = JSON.stringify(datosSemanaTemp[i].date);
        horaTemp =
horaGeneralTemp.charAt(12)+horaGeneralTemp.charAt(13)+':'+horaGeneralTe
mp.charAt(15)+horaGeneralTemp.charAt(16);
        horasSemanaTemp[i] = horaTemp;
        humSemana[i] = datosSemanaTemp[i].humedad;
        horaGeneralHum = JSON.stringify(datosSemanaTemp[i].date);
        horaHum =
horaGeneralHum.charAt(12)+horaGeneralHum.charAt(13)+':'+horaGeneralHum.
charAt(15)+horaGeneralHum.charAt(16);
        horasSemanaHum[i] = horaHum;
    };


    /*console.log("TEMPERATURA: ");
    console.log(tempSemana);
    console.log(horasSemanaTemp);
    console.log("HUMEDAD: ");
    console.log(humSemana);
    console.log(horasSemanaHum);*/

    //Se crea el arreglo con los datos a enviar
    let envio = {tempSemana, horasSemanaTemp, humSemana,
horasSemanaHum};
    //Se envía la respuesta al cliente
    res.send(envio);
};
//Se exporta el controlador
module.exports = semanaTemperaturaCtrl;
```

Anexo 12.7 Controlador para las rutas */temp-hum* y */temp-hum-db*.

```
//Recibe la peticiónes POST enviadas desde la Raspberry para
temperatura y humedad
const tempHumCtrl = {};//Se crea el controlador

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const tempHum = require('../models/datos-temperatura-humedad');

tempHumCtrl.getTempHum = async (req,res) => {
    const tempHumDatos = await tempHum.find();
    res.json(tempHumDatos);
};

//Se crea un método en el controlador para enviar los datos por el
socket
tempHumCtrl.postTempHum = async (req,res) => {

    const {temperatura, humedad} = req.body;//Se extraen los datos
enviados desde la Raspberry
    //Se crea un objeto con los datos recibidos para enviar por el
socket
    const temp_hum_datos = {
```

```
        temperatura : temperatura,
        humedad: humedad
    }

    //Se emite el evento del socket con los datos
    server.sockets.eventos('datos-temp-hum', temp_hum_datos);

    console.log(temp_hum_datos);
    //Se envia una respuesta al cliente
    res.send('Temperatura y humedad publicadas');
};

//Se crea un método en el controlador para guardar los datos en la base
de datos
tempHumCtrl.postTempHumDB = async (req,res) => {

    const {temperatura, humedad} = req.body;//Se extraen los datos
enviados desde la Raspberry
    //Se crea un objeto con el modelo requerido
    const newDatos = new tempHum({
        temperatura: temperatura,
        humedad: humedad
    });
    await newDatos.save() //Se guarda el objeto en la base de datos

    console.log(newDatos);
    //Se envía la respuesta al cliente
    res.send('Temperatura y humedad guardadas');
};
//Se exporta el controlador
module.exports = tempHumCtrl;
```

Anexo 12.8 Controlador para las rutas */velocidad* y */velocidad-db.*

```
//Recibe la peticiónes POST enviadas desde la Raspberry para velocidad
de viento
const velocidadCtrl = {};//Se crea el controlador

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const veloci = require('../models/datos-velocidad');

velocidadCtrl.getVelocidad = async (req,res) => {
    const velocidadDato = await veloci.find();
    res.json(velocidadDato);
};

//Se crea un método en el controlador para enviar los datos por el
socket
velocidadCtrl.postVelocidad = async (req,res) => {
    const {velocidad} = req.body;//Se extraen los datos enviados desde
la Raspberry

    //Se crea un objeto con los datos recibidos para enviar por el
socket
    const velocidad_dato = {
```

```
        velocidad : velocidad
    }
    //Se emite el evento del socket con los datos
    server.sockets.eventos('datos-velocidad', velocidad_dato);

    console.log(velocidad_dato);
    //Se envia una respuesta al cliente
    res.send('Velocidad publicada');
};

//Se crea un método en el controlador para guardar los datos en la base
de datos
velocidadCtrl.postVelocidadDB = async (req,res) => {
    const {velocidad} = req.body;//Se extraen los datos enviados desde
la Raspberry
    //Se crea un objeto con el modelo requerido
    const newDato = new veloci({
        velocidad: velocidad,
    });
    await newDato.save() //Se guarda el objeto en la base de datos

    console.log(newDato);
    //Se envía la respuesta al cliente
    res.send('Velocidad guardada');
};
//Se exporta el controlador
module.exports = velocidadCtrl;
```

Anexo 12.9 Controlador para las rutas */vol-cor-pot* y */vol-cor-pot-db*.

```
//Recibe la peticiónes POST enviadas desde la Raspberry para voltaje,
corriente y potencia
const volCorPotCtrl = {};//Se crea el controlador

const server = require('../index');

//Se requieren los modelos de datos de la base de datos
const volCorPotDB = require('../models/datos-vol-cor-pot');

volCorPotCtrl.getVolCorPot = async (req,res) => {
    const volCorPotDatos = await volCorPotDB.find();
    res.json(volCorPotDatos);
};

//Se crea un método en el controlador para enviar los datos por el
socket
volCorPotCtrl.postVolCorPot = async (req,res) => {
    const {voltaje, corriente, potencia} = req.body;//Se extraen los
datos enviados desde la Raspberry
    //Se crea un objeto con los datos recibidos para enviar por el
socket
    const volCorPot_dato = {
        voltaje : voltaje,
        corriente : corriente,
        potencia : potencia
    }
```

```
    //Se emite el evento del socket con los datos
    server.sockets.eventos('datos-vol-cor-pot', volCorPot_dato);

    console.log(volCorPot_dato);
    //Se envia una respuesta al cliente
    res.send('Voltaje, corriente y potencia publicadas');
};

//Se crea un método en el controlador para guardar los datos en la base
de datos
volCorPotCtrl.postVolCorPotDB = async (req,res) => {
    const {voltaje, corriente, potencia} = req.body;//Se extraen los
datos enviados desde la Raspberry
    //Se crea un objeto con el modelo requerido
    const newDato = new volCorPotDB({
        voltaje : voltaje,
        corriente : corriente,
        potencia : potencia
    });
    await newDato.save() //Se guarda el objeto en la base de datos

    console.log(newDato);
    //Se envía la respuesta al cliente
    res.send('Voltaje, corriente y potencia guardadas');
};
//Se exporta el controlador
module.exports = volCorPotCtrl;
```

Anexo 13. Archivo principal del fronted: Index.js

```
//Archivo principal de ejecución
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import {MonitoreoApp} from "./MonitoreoApp";


ReactDOM.render(
    <MonitoreoApp />
  ,
  document.getElementById('root') //Se envía al componente id:root del
index.html
);
```

Anexo 14. Archivo del fronted: MonitoreoApp.js

```
//Se ejecuta la aplicación con capa de sockets
import React from "react";
import {SocketProvider} from './context/SocketContext'

import App from "./App";
export const MonitoreoApp = () => {
    return(
        <SocketProvider>
```

```
            <App />
        </SocketProvider>
    );
}
```

Anexo 15. Archivo del fronted: App.js

```
import React from "react";
import {BrowserRouter as Router, Route} from 'react-router-dom';//para
definir las rutas

import 'bootstrap/dist/css/bootstrap.min.css';//estilos de bootstrap
import './App.css';//estilos

//Componentes que se pintan en cada ruta
import Navigation from './components/navigation';
import Inicio from "./components/inicio";
import tiempo_real from "./components/tiempo_real";
import Estadisticas from "./components/estadisitcas";
import About from './components/about';

function App() {

    return (
        <>
            <Router>

                <Navigation/> {/*La navegación esta presente en todas
las rutas*/}

                {/*Se asigna un componente a cada ruta*/}
                <div className="container">
                    <Route path='/' exact component={Inicio}/>
                    <Route path='/tiempo-real'
component={tiempo_real}/>
                    <Route path='/estadisticas'
component={Estadisticas}/>
                    <Route path='/about' component={About}/>

                </div>
            </Router>
        </>

    );
}
//Se exporta la función App
export default App;
```

Anexo 16. Componentes de React.

Anexo 16.1 Creación del componente Navigation.

```
        import React, {Component} from 'react';
import {Link} from 'react-router-dom';
```

```
class Navigation extends Component {
    render() {
        return (
            <nav className="navbar navbar-expand-lg navbar-dark bg-
dark">
                <div className="container-fluid">
                    <Link className="navbar-brand" to="/" >
                          Monitoreo de la producción de energía eólica
                    </Link>      
                    <button className="navbar-toggler" type="button"
data-bs-toggle="collapse"
                            data-bs-target="#navbarNav" aria-
controls="navbarNav" aria-expanded="false"
                            aria-label="Toggle navigation">
                        <span className="navbar-toggler-icon"></span>
                    </button>
                    <div className="collapse navbar-collapse"
id="navbarNav">
                        <ul className="navbar-nav m-lg-auto">
                            <li className="nav-item">
                                <Link className="nav-link active"
to="/">Inicio</Link>
                            </li>
                            <li className="nav-item">
                                <Link className="nav-link active"
to="/tiempo-real">Tiempo real</Link>
                            </li>
                            <li className="nav-item">
                                <Link className="nav-link active"
to="/estadisticas">Estadísticas de producción</Link>
                            </li>
                            <li className="nav-item">
                                <Link className="nav-link active"
to="/about">Acerca de</Link>
                            </li>
                        </ul>
                    </div>
                </div>
            </nav>
        );
    }
}

export default Navigation;
```

Anexo 16.2 Creación del componente Carrusel.

```
        import React, {useState} from 'react';
import {
    Carousel,
    CarouselItem,
    CarouselControl,
    CarouselIndicators,
    CarouselCaption
} from 'reactstrap';
```

```
import aerogeneradorIMG from '../photos/aerogeneradorIMG.jpg';
import energiaEolicaIMG from '../photos/img1.jpg';
import uniPamplonaIMG from '../photos/unipamplonaIMG.jpg';
import villaMarinaIMG from '../photos/villa3.jpg';

const items = [
    {
        src: uniPamplonaIMG,
        altText: 'unipamplonaIMA',
        caption: 'Universidad de Pamplona',
        description: '',
        link: 'http://www.unipamplona.edu.co'
    },
    {
        src: energiaEolicaIMG,
        altText: 'energEolica',
        caption: <font color="white">
            Energía eólica
        </font>,
        description: <font color="black">
            Consiste en convertir la energía que produce el movimiento
de las palas de un aerogenerador impulsadas por el viento en energía
eléctrica.
        </font>,
        link: 'https://www.acciona.com/es/energias-renovables/energia-
eolica/'
    },
    {
        src: aerogeneradorIMG,
        altText: 'aerogeneradorAIR40',
        caption: 'Aerogenerador AIR 40',
        description: <font color="black">
            Este es un aerogenerador utilizado en aplicaciones
terrestres. Funciona de manera eficiente en una amplia gama de
velocidades de viento.
        </font>,
        link:
'https://www.primuswindpower.com/files/2213/8972/7061/Primus_Air_Manual
_Spanish.pdf'
    },
    {
        src: villaMarinaIMG,
        altText: 'villaMarina',
        caption: '',
        description: <font color="black">
            Ubicada en el Kilómetro 49 Vía Cúcuta - Pamplona, busca
promover, generar proyectos, programas y actividades con iniciativas de
carácter ambiental, con el  n de potencializar y desarrollar estilos de
vida saludables ademas de la protección de los recursos naturales y
acciones de cultura de paz y sana convivencia.
        </font>,
        link: 'http://www.unipamplona.edu.co/villamarina/'
    }
];

const Carrusel = (props) => {
```

```
    const [activeIndex, setActiveIndex] = useState(0);
    const [animating, setAnimating] = useState(false);

    const next = () => {
        if (animating) return;
        const nextIndex = activeIndex === items.length - 1 ? 0 :
activeIndex + 1;
        setActiveIndex(nextIndex);
    }

    const previous = () => {
        if (animating) return;
        const nextIndex = activeIndex === 0 ? items.length - 1 :
activeIndex - 1;
        setActiveIndex(nextIndex);
    }

    const goToIndex = (newIndex) => {
        if (animating) return;
        setActiveIndex(newIndex);
    }

    const slides = items.map((item) => {
        return (
            <CarouselItem
                onExiting={() => setAnimating(true)}
                onExited={() => setAnimating(false)}
                key={item.src}

            >
                <br/>
                <center>
                    <a href={item.link} target="_blank"
rel="noreferrer">
                        <img src={item.src} alt={item.altText}
width="90%" height="300px"/>
                    </a>
                </center>
                <CarouselCaption captionText={item.description}
captionHeader={item.caption}/>
            </CarouselItem>
        );
    });

    return (

        <Carousel

            activeIndex={activeIndex}
            next={next}
            previous={previous}
        >
            <CarouselIndicators items={items} activeIndex={activeIndex}
onClickHandler={goToIndex}/>
            {slides}
            <CarouselControl direction="prev" directionText="Anterior"
onClickHandler={previous}/>
```

```
            <CarouselControl direction="next" directionText="Siguiente"
onClickHandler={next}/>

        </Carousel>
    );
}

export default Carrusel;
```

Anexo 16.3 Creación del componente Inicio.

```
        import React from 'react';
import {Link} from 'react-router-dom';
import '../App.css';
import 'bootstrap/dist/css/bootstrap.min.css'
import Carrusel from './carrusel';
import Footer from "./footer";
import {Button} from "antd";

function Inicio() {

    return (
        <div className="inicio">

            <Carrusel/>
            <div> <hr/> </div>

            <div className="card">
                <div className="card-body text-center">
                    Este proyecto describe la implementación de un
monitoreo remoto de las características de
                    producción energética en la sede social Villa
Marina de la Universidad de Pamplona por medio del control
                    del aerogenerador AIR 40 el cual está ubicado en
ese lugar. Podrá ver los valores de la producción
                    energética y las variables mdioambientales en
tiempo real, pero también un informe estadístico
                    de esta producción en diferentes intervalos de
tiempo.
                </div>
            </div>

            <div> <hr/> </div>

            <div className="row justify-content-center">
                <div className="col-sm-4 text-center d-grid">
                    <button type="button" className="btn btn-outline-
info btn-lg me-md-2">
                        <Link to="/tiempo-real" style={{
textDecoration: 'none', color: 'white'}}>Sensores en tiempo real</Link>
                    </button>
                    <Button></Button>
                    <button type="button" className="btn btn-outline-
info btn-lg me-md-2">
                        <Link to="/estadisticas" style={{
textDecoration: 'none', color: 'white'}}>Estadísticas de
producción</Link>
```

```
                    </button>
                </div>
            </div>

            <div><hr/></div>

            <Footer/>

        </div>


    );
}

export default Inicio;
```

Anexo 16.4 Creación del componente Footer.

```
import React, {Component} from 'react';

class Footer extends Component {
    render() {
        return (
            <div>

                <div className="card-footer text-muted text-center">
                    Universidad de Pamplona, 2021
                </div>

                <div><hr/></div>
            </div>
        );
    }
}

export default Footer;
```

Anexo 16.5 Creación del componente About.

```
import React, {Component} from 'react';
import 'bootstrap/dist/css/bootstrap.min.css'
import logo from '../photos/logoUP.png';
import '../App.css';
import Footer from "./footer";

class About extends Component {
    render() {
        return (
            <div>

                <div className="card text-center about">

                    <div className="card-header">
                        Monitoreo remoto de los parámetros energéticos
del aerogenerador AIR 40, usando herramientas web modernas y sistemas
embebidos.
                    </div>
```

```
                    <div className="card-body">
                        <h5 className="card-title">Tesis de pregrado
por el título de ingeniero electrónico.</h5>
                        <br/>
                        <p className="card-text">Este proyecto describe
la implementación de un monitoreo remoto de las características de
producción energética en la sede social Villa Marina de la Universidad
de Pamplona por medio del control del aerogenerador AIR 40 el cual está
ubicado en ese lugar. Podrá ver los valores de la producción energética
y las variables medioambientales en tiempo real, pero también un
informe estadístico de esta producción en diferentes intervalos de
tiempo.</p>
                        <br/>
                        <p style={{textAlign: 'justify'}}>Este sitio
web fue desarrollado desde cero con el objetivo de mostrar en él los
resultados obtenidos en el proyecto,
                        visualizar los valores de los sensores en
tiempo real ubicados en Villa Marina, por medio del sistema embebido
Raspberry Pi 3B+
                        y tambien dar un estudio de la producción de
energía por medio de gráficos estadísticos para caracterizar la
viabilidad del lugar para la producción de
                        energía eólica en mayor cantidad.</p>
                        <br/>

                        <div className="row justify-content-center">

                            <div className="card" style={{'width':
'30rem'}}>
                                <div className="card-body">
                                    <h5 className="card-
title">Desarrollado por:</h5>
                                    <p className="text-lg-start">Diego
Fernando Esquivel Rangel.</p>
                                    <p className="text-lg-
start"><i><font size="2">Estudiante de Ingeniería
Electrónica.</font></i></p>
                                    <p className="text-lg-
start"><i><font size="2">Universidad de Pamplona.</font></i></p>
                                    <p className="text-lg-start">
                                        <svg
xmlns="http://www.w3.org/2000/svg" width="50" height="40"
fill="currentColor"
                                             className="bi bi-envelope-
fill" viewBox="0 0 16 16">
                                            <path
                                                d="M.05 3.555A2 2 0 0 1
2 2h12a2 2 0 0 1 1.95 1.555L8 8.414.05 3.555zM0 4.697v7.104l5.803-
3.558L0 4.697zM6.761 8.83l-6.57 4.027A2 2 0 0 0 2 14h12a2 2 0 0
1.808-1.144l-6.57-4.027L8 9.586l-1.239-.757zm3.436-.586L16
11.801V4.697l-5.803 3.546z"/>
                                        </svg>


diego.esquivel@unipamplona.edu.co

           
```

```
; d9815d@gmail.com
                                    </p>
                                    <p className="text-lg-start"><font
size="2">Pamplona, Norte de Santander, Colombia.</font></p>

                                </div>
                            </div>

                            <div className="card" style={{'width':
'30rem'}}>
                                <div className="card-body">
                                    <h5 className="card-
title">Información relacionada:</h5>
                                    <p className="text-lg-
end"><i>Fecha:
                                        <font size="2"> Proyecto
realizado en el primer semestre del 2021.</font></i></p>
                                    <p className="text-lg-
end"><i>Director de tesis:
                                        <font size="2"> MSc. Luis
Alberto Muñoz Bedoya.</font></i></p>
                                    <p className="text-lg-
end"><i>Codirector de tesis:
                                        <font size="2"> Ing. Jose
Daniel Ramirez Corzo.</font></i></p>
                                    <p className="text-lg-
end"><i>Programa académico:
                                        <font size="2"> Ingeniería
electrónica.</font></i></p>
                                    <p className="text-lg-
end"><i>Facultad:
                                        <font size="2"> Ingenierías y
arquitectura.</font></i></p>
                                    <p>
                                        Universidad de Pamplona.
                                        <img src={logo} alt="logoUP"
width="20%" height="100px"/>
                                    </p>
                                    <p><font size="2">Pamplona, Norte
de Santander, Colombia.</font></p>

                                </div>
                            </div>

                        </div>
                    </div>


                </div>

                <Footer/>

            </div>
        );
    }
}
```

```
export default About;
```

Anexo 16.6 Creación del componente Tiempo real.

```
import React, {useContext, useEffect, useState} from 'react';
import {SocketContext} from "../context/SocketContext";
import ReactPlayer from "react-player";
import '../App.css';

import varAmbientalesIMA from '../photos/varAmbientalesIMG.png';
import varElectricasIMA from '../photos/varElectricasIMG.png';
import Footer from "./footer";

import estacionFoto2 from '../photos/estacionFoto2.jpg';
import estacionFoto3 from '../photos/estacionFoto3.jpg';


function TiempoReal() {

    const {socket, online} = useContext(SocketContext);

    const [datosTH = {
        temperatura: 0,
        humedad: 0
    }, setDatosTH] = useState();

    const [datosVel = {
        velocidad: 0
    }, setDatosVel] = useState();

    const [datosVolCorPot = {
        voltaje: 0,
        corriente: 0,
        potencia: 0
    }, setDatosVolCorPot] = useState();

    useEffect(() => {
        socket.on('conectado', (data) => {
            console.log(data);
        })
        socket.on('datos-temp-hum', (data) => {
            console.log(data);
            setDatosTH(data);
        })
        socket.on('datos-velocidad', (data) => {
            console.log(data);
            setDatosVel(data);
        })
        socket.on('datos-vol-cor-pot', (data) => {
            console.log(data);
            setDatosVolCorPot(data);
        })

    }, [socket])

    return (
```

```
        <div className="tiempo-real">
            <hr/>
            <div className="row justify-content-center">
                <div className="col-sm-4 text-center">
                    <p className="bg-warning">
                        Servidor en tiempo real
                        {
                            online
                                ? <span className="text-success">
conectado</span>
                                : <span className="text-danger">
desconectado</span>
                        }
                    </p>
                </div>
            </div>

            <div>
                <div className="card-group">
                    <center>
                        <img src={varAmbientalesIMA} className="bg-
secondary"
                             alt="variablesAmbientalesIMA"
height="180px" style={{'borderRadius':'5px'}}></img>
                    </center>

                    <div>  </div>

                    <div className="card">
                        <div className="card-body">
                            <h5 className="card-title">Temperatura:
</h5>
                            <p className="card-text text-center" >
                                <font size="7">
{datosTH['temperatura']} °C </font>
                            </p>
                        </div>
                    </div>
                    <div className="card">
                        <div className="card-body  bg-success">
                            <h5 className="card-title">Velocidad del
viento: </h5>
                            <p className="card-text text-center" >
                                <font size="7"> {datosVel['velocidad']}
m/s </font>
                            </p>
                        </div>
                    </div>
                    <div className="card">
                        <div className="card-body">
                            <h5 className="card-title">Humedad: </h5>
                            <p className="card-text text-center" >
                                <font size="7"> {datosTH['humedad']} %
</font>
                            </p>
                        </div>
                    </div>
```

```
                </div>

            </div>

            <hr/>

            <div className="card-group">
                <center>
                    <img src={varElectricasIMA} className="bg-
secondary"
                         alt="variablesElectricasIMA" height="180px"
style={{'borderRadius':'5px'}}></img>
                </center>

                <div>  </div>

                <div className="card">
                    <div className="card-body">
                        <h5 className="card-title">Corriente: </h5>
                        <p className="card-text text-center" >
                            <font size="7">
{datosVolCorPot['corriente']} A </font>
                        </p>
                    </div>
                </div>
                <div className="card">
                    <div className="card-body bg-success">
                        <h5 className="card-title">Voltaje: </h5>
                        <p className="card-text text-center" >
                            <font size="7"> {datosVolCorPot['voltaje']}
V </font>
                        </p>
                    </div>
                </div>
                <div className="card">
                    <div className="card-body">
                        <h5 className="card-title">Potencia: </h5>
                        <p className="card-text text-center" >
                            <font size="7">
{datosVolCorPot['potencia']} W </font>
                        </p>
                    </div>
                </div>
            </div>

            <hr/>
            <div className="row">
                <div className="col-sm-4">
                    <img src={estacionFoto3} alt="estacionFoto1"
width="100%" height="200px" style={{'borderRadius':'10px'}}/>
                </div>
                <div className="col-sm-4" >
                    <ReactPlayer

url={'https://www.youtube.com/watch?v=peVwftkxDck&ab_channel=DiegoEsqui
vel'}
```

```
                        className='react-player'
                        width='100%'
                        height='100%'
                        playing
                        muted
                        loop
                    />

                </div>
                <div className="col-sm-4">
                    <img src={estacionFoto2} alt="estacionFoto1"
width="100%" height="200px" style={{'borderRadius':'10px'}}/>
                </div>

            </div>

            <hr/>

            <Footer/>


        </div>

    );
}

export default TiempoReal;
```

Anexo 16.7 Creación del componente Estadisticas.

```
import React, {Component} from 'react';
import Consulta from "./consulta";
import Ultimas from "./ultimas";
import Footer from "./footer";
import logo from "../photos/logoUP.png";

class Estadisitcas extends Component {
    render() {

        return (
            <div>

                <div className="row">

                    <div className="col-sm-6">
                        <Consulta/>
                        <br/>

                        <div className="row">
                            <div className="col-sm-3">

                            </div>
                            <div className="col-sm-6"
style={{"textAlign":"center"}}>
                                <img src={logo} alt="logoUP"
width="50%" height="150px"/>
                            </div>
```

```
                            <div className="col-sm-3">

                            </div>

                        </div>

                    </div>


                    <div className="col-sm-6">
                        <br/><br/><br/><br/><br/>
                        <Ultimas/>

                    </div>
                </div>




                <br/>
                <Footer/>

            </div>
        );
    }
}

export default Estadisitcas;
```

Anexo 16.8 Creación del componente Consulta.

```
import React, {Component} from 'react';
import funciones from './temperatura';
import 'bootstrap/dist/css/bootstrap.min.css'
import {Button, Modal, ModalHeader, ModalBody, ModalFooter} from
'reactstrap';
import DatePicker, {registerLocale} from 'react-datepicker';
import DateRangePicker from 'react-bootstrap-daterangepicker';
import 'bootstrap/dist/css/bootstrap.css';
import 'bootstrap-daterangepicker/daterangepicker.css';
import 'react-datepicker/dist/react-datepicker.css';
import 'react-date-range/dist/styles.css';
import 'react-date-range/dist/theme/default.css';
import {Chart} from 'react-chartjs-2';
import axios from "axios";
import es from 'date-fns/locale/es';
registerLocale("es",es);


class Consulta extends Component {

    myChart;
    myChartTemp;

    state = {
        fecha: new Date().toLocaleString('en-US', {timeZone:
'America/Bogota'}),
```

```
        fechaSemana: new Date().toLocaleString('en-US', {timeZone:
'America/Bogota'})
    }
    state = {
        abierto: false,
        abiertoSemana:false,
        loading: false,
        startDate : null,
        endDate : null,
        minDate : parseInt('03332200'),
        maxDate : parseInt('13332200'),
        /*rangePicker: {
            startDate: moment(),
            endDate: moment().add(7, "days")
        },*/
    }
    abrirModal = () => {
        this.setState({abierto: !this.state.abierto});
    }

    abrirModalSemana = () => {
        this.setState({abiertoSemana: !this.state.abiertoSemana});
    }

    onChange = fecha => {
        this.setState({fecha: fecha});
    }

    onChangeSemana = fecha => {
        this.setState({fechaSemana: fecha});
    }

    semanaInicial;
    semanaFinal;

    generarGraficaDia = (datos) => {
        let ctx =
document.getElementById("graficaPotencia").getContext('2d');
        if(this.myChart){
            this.myChart.destroy();
        };

        let limite = [];
        let limiteSup = [];
        for(let i=0; i<datos.data.horasDiaPot.length; i++ ){
            limite[i] = 1.5
            limiteSup[i] = 1.8
        }

        this.myChart = new Chart(ctx, {
           type: 'line',
           data: {
               labels: datos.data.horasDiaPot,
               datasets: [{
                   //type: 'bar',
                   label: 'POTENCIA en Watts',
                   borderWidth: 0.5,
```

```
                    //fill: true,
                    pointRadius: 1.5,
                    pointHoverRadius: 1,
                    backgroundColor: "rgba(75,192,192,0.2)",
                    borderColor: "rgba(75,192,192,1)",
                    data: datos.data.potenciaDia
                },{
                    label: 'VELOCIDAD del VIENTO en m/s',
                    backgroundColor: 'rgba(255,0,0,0.5)',
                    fill: false,
                    pointRadius: 1.2,
                    pointHoverRadius: 1,
                    hoverBackgroundColor: 'rgba(255,180,0,0.5)',
                    data: datos.data.velocidadDia
                },{
                    label: "Inf",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limite
                },{
                    label: "Sup",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limiteSup
                }]
            },
             options: {
                 responsive: true
             }

        });
        console.log(datos.data);
    }

    generarGraficaSemana = (datos) => {
        let ctx =
document.getElementById("graficaPotencia").getContext('2d');
        if(this.myChart){
            this.myChart.destroy();
        };

        let limite = [];
        let limiteSup = [];
        for(let i=0; i<datos.data.horasSemanaPot.length; i++ ){
            limite[i] = 1.5
            limiteSup[i] = 1.8
        }

        this.myChart = new Chart(ctx, {
            type: "line",
            data: {
                labels: datos.data.horasSemanaPot,
                datasets: [{
```

```
                    //type: 'bar',
                    label: 'POTENCIA en Watts',
                    borderWidth: 0.5,
                    //fill: true,
                    pointRadius: 1.5,
                    pointHoverRadius: 1,
                    backgroundColor: "rgba(75,192,192,0.2)",
                    borderColor: "rgba(75,192,192,1)",
                    data: datos.data.potenciaSemana
                },{
                    //type: 'line',
                    label: 'VELOCIDAD del VIENTO en m/s',
                    backgroundColor: 'rgba(255,0,0,0.5)',
                    fill: false,
                    pointRadius: 1.2,
                    pointHoverRadius: 1,
                    hoverBackgroundColor: 'rgba(255,180,0,0.5)',

                    data: datos.data.velocidadSemana
                },{
                    label: "Inf",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limite
                },{
                    label: "Sup",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limiteSup
                }]
            },
            options: {
                responsive: true
            }

        });
        console.log(datos.data);
    }


    seleccionarFecha = (fecha) => {
        console.log(fecha);
        const options = {weeday: 'long', year:'numeric', month:'long',
day:'numeric'};
        const diaPost = async() => {
            axios.post('http://www.estacionunipamplona.tk/api/dia-
especifico', {
                params: {
                    fecha: fecha
                }
            })
                .then(res => {
                    this.generarGraficaDia(res);
```

```
                })
                .catch(err => {
                    console.log(err);
                })
            let titulo = fecha.toLocaleDateString('es-Es', options);
            document.getElementById("tituloGrafica").innerHTML =
titulo;
        };
        diaPost();
        funciones.Temperatura(fecha);

    }

    seleccionarFechaSemana = (fechaIni, fechaFin) => {
        const options = {weeday: 'long', year:'numeric', month:'long',
day:'numeric'};
        const semanaPost = async() => {
            axios.post('http://www.estacionunipamplona.tk/api/semana-
especifica', {
                params: {
                    fechaIni: fechaIni,
                    fechaFin: fechaFin
                }
            })
                .then(res => {
                    this.generarGraficaSemana(res);
                })
                .catch(err => {
                    console.log(err);
                })
            let titulo = fechaIni.toLocaleDateString('es-Es',
options)+'-'+fechaFin.toLocaleDateString('es-Es', options);
            document.getElementById("tituloGrafica").innerHTML =
titulo;
        };

        semanaPost();
        funciones.TemperaturaSemana(fechaIni, fechaFin);

    }

    fechaMaxima = (fecha) => {

        fecha.setDate(fecha.getDate() - 1);

        return fecha;
    }

    render() {
        const modalStyle={
            position:"absolute",
            top:'50%',
            left:'50%',
            transform: 'translate(-50%, -50%)',

        }
```

```
        return (
            <div>
                <br/>
                <p><small><i><mark>Puede buscar los datos de un día
específico, una semana o un rango determinado de días y se generará una
gráfica.</mark></i></small></p>
                <div className="dropdown"
style={{"textAlign":"center"}}>
                    <button
                        className="btn btn-success dropdown-toggle"
                        type="button"
                        id="dropdownMenuButton"
                        data-bs-toggle="dropdown"
                        aria-expanded="false"
                    >
                        Elegir busqueda
                    </button>
                    <ul className="dropdown-menu" aria-
labelledby="dropdownMenuButton">
                        <li><Button className="dropdown-item"
onClick={this.abrirModal} >Día</Button></li>
                        <hr className="dropdown-divider"/>
                        <li>
                            <Button className="dropdown-item"
onClick={this.abrirModalSemana}>Semana</Button>
                        </li>
                    </ul>
                </div>
                <br/>
                <div>


                </div>

                <div style={{background: '#cfd7d8',
'borderRadius':'15px'}}>
                    <h5 id="tituloGrafica"
style={{"textAlign":"center"}}>

                    </h5>
                    <canvas id="graficaPotencia" width="400"
height="300"></canvas>

                </div>

                <br/>

                <div style={{background: '#cfd7d8',
'borderRadius':'15px'}}>
                    <canvas id="graficaTemperatura" width="400"
height="200"></canvas>
                </div>



                <Modal isOpen={this.state.abierto}
style={{modalStyle}}>
```

```
                    <ModalHeader>
                        Seleccionar día
                    </ModalHeader>
                    <ModalBody className="text-center">
                        <DatePicker selected={this.state.fecha}
onChange={this.onChange} locale="es" dateFormat="dd 'de' MMMM 'de'
yyyy" placeholderText="Buscar" minDate={new Date("05 30 2021")}
maxDate={this.fechaMaxima(new Date())}  /*showTimeSelect*//>
                        <br/>
                        <br/>
                        <input type="button" value="Seleccionar fecha"
className="btn btn-primary"
onClick={()=>{this.seleccionarFecha(this.state.fecha)}}/>
                    </ModalBody>
                    <ModalFooter>
                        <Button color="dark"
onClick={this.abrirModal}>Cerrar</Button>
                    </ModalFooter>
                </Modal>

                <Modal isOpen={this.state.abiertoSemana}
style={{modalStyle}}>
                    <ModalHeader>
                        Seleccionar semana
                    </ModalHeader>
                    <ModalBody className="text-center">
                        <DateRangePicker
                            initialSettings={{ startDate: '10 30 2021',
endDate: this.fechaMaxima(new Date()) }}
                            onCallback={(start, end, label) => {
                                console.log(start._d);
                                this.semanaInicial = start._d;
                                console.log(end._d);
                                this.semanaFinal = end._d;
                            }}
                            onApply={event =>
this.seleccionarFechaSemana(this.semanaInicial, this.semanaFinal)}
                        >
                            <button className="btn-primary btn-
lg">Seleccionar rango</button>
                        </DateRangePicker>
                        <br/>
                        <br/>
                    </ModalBody>
                    <ModalFooter>
                        <Button color="dark"
onClick={this.abrirModalSemana}>Cerrar</Button>
                    </ModalFooter>
                </Modal>



            </div>
        );
    }
}
```

```
export default Consulta;
```

Anexo 16.9 Creación del componente Ultimas.

```
import React, {Component} from 'react';
import {Card, CardBody, CardTitle, CardText, CardSubtitle} from
"reactstrap";
import axios from "axios";
import moment from "moment";
import {Chart} from "react-chartjs-2";

class Ultimas extends Component {

    myChart;
    myChartHora;

    seleccionarFecha = (fecha) => {

        const diaGet = async() => {
            axios.get('http://www.estacionunipamplona.tk/api/dia-
ultimo', {
            })
                .then(res => {
                    this.generarGraficaDia(res);
                })
                .catch(err => {
                    console.log(err);
                })

        };
        diaGet();

    }

    generarGraficaDia = (datos) => {

        let ctx =
document.getElementById("graficaUltimoDia").getContext('2d');
        if(this.myChart){
            this.myChart.destroy();
        };

        let limite = [];
        let limiteSup = [];
        for(let i=0; i<datos.data.horasDiaPot.length; i++ ){
            limite[i] = 1.5
            limiteSup[i] = 1.8
        }

        this.myChart = new Chart(ctx, {
            type: "line",
            data: {
                labels: datos.data.horasDiaPot,
                datasets: [{
                    label: 'POTENCIA en Watts',
                    backgroundColor: "rgba(0,200,0,0.8)",
```

```
                    //fill: true,
                    borderWidth: 0.5,
                    pointRadius: 1.5,
                    pointHoverRadius: 1,
                    borderColor: "rgba(75,192,192,1)",
                    data: datos.data.potenciaDia
                },{
                    label: 'VELOCIDAD del VIENTO en m/s',
                    backgroundColor: 'rgba(255,0,0,0.5)',
                    hoverBackgroundColor: 'rgba(255,180,0,0.5)',
                    fill: false,
                    pointRadius: 1.2,
                    pointHoverRadius: 1,
                    data: datos.data.velocidadDia
                },{
                    label: "Inf",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limite,
                },{
                    label: "Sup",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limiteSup
                }]
            },
            options: {
                responsive: true
            }

        });
        console.log(datos.data);
    }

    seleccionarFechaHora = (fecha) => {
        const horaGet = async() => {
            axios.get('http://www.estacionunipamplona.tk/api/hora-
ultima', {
            })
                .then(res => {
                    this.generarGraficaHora(res);
                })
                .catch(err => {
                    console.log(err);
                })


        };
        horaGet();

    }

    generarGraficaHora = (datos) => {
```

```
        this.fechaHora = datos.data.fechaIni;
        let ctx =
document.getElementById("graficaUltimaHora").getContext('2d');
        if(this.myChartHora){
            this.myChartHora.destroy();
        };

        let limite = [];
        let limiteSup = [];
        for(let i=0; i<datos.data.horasDiaVel.length; i++ ){
            limite[i] = 1.5
            limiteSup[i] = 1.8
        }

        this.myChartHora = new Chart(ctx, {
            type: "line",
            data: {
                labels: datos.data.horasDiaVel,
                datasets: [{
                    label: 'POTENCIA en Watts',
                    backgroundColor: "rgba(0,200,0,0.8)",
                    //fill: true,
                    borderWidth: 0.5,
                    pointRadius: 1.5,
                    pointHoverRadius: 1,
                    borderColor: "rgba(75,192,192,1)",
                    data: datos.data.potenciaDia
                },{
                    label: 'VELOCIDAD del VIENTO en m/s',
                    backgroundColor: 'rgba(255,0,0,0.5)',
                    hoverBackgroundColor: 'rgba(255,180,0,0.5)',
                    fill: false,
                    pointRadius: 1.2,
                    pointHoverRadius: 1,
                    data: datos.data.velocidadDia
                },{
                    label: "Inf",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limite
                },{
                    label: "Sup",
                    type: 'line',
                    pointRadius: 0.1,
                    pointHoverRadius: 0.1,
                    backgroundColor: 'rgba(14,15,15,1)',
                    data: limiteSup
                }]
            },
            options: {
                responsive: true
            }

        });
        console.log(datos.data);
```

```
    }


    render() {
        let fecha = new Date();
        this.seleccionarFecha(fecha);
        this.seleccionarFechaHora(fecha);
        let fechaDia = moment().subtract(1,'days').format();
        fechaDia =
fechaDia.charAt(8).concat(fechaDia.charAt(9).concat('-
',fechaDia.charAt(5).concat(fechaDia.charAt(6).concat('-
20',fechaDia.charAt(2).concat(fechaDia.charAt(3))))));

        let fechaHora = moment().subtract(1,'hour').format();
        console.log(fechaHora);
        fechaHora =
fechaHora.charAt(11).concat(fechaHora.charAt(12).concat(':00 -
',fechaHora.charAt(11).concat(fechaHora.charAt(12).concat(':59'))))


        return (
            <div>
                <div className="col">
                    <div>
                        <Card style={{'borderRadius':'10px'}}>
                            <CardBody>
                                <CardTitle tag="h5">
                                    Producción energética en la última
hora.
                                </CardTitle>
                                <CardSubtitle className="mb-2 text-
muted" tag="h6">
                                    {fechaHora}
                                </CardSubtitle>
                                <CardText
style={{background:'#E0FFFF'}}>
                                    <canvas id="graficaUltimaHora"
width="800" height="400"></canvas>

                                </CardText>
                            </CardBody>
                        </Card>
                    </div>

                    <hr/>

                    <div>
                        <Card style={{'borderRadius':'10px'}}>
                            <CardBody>
                                <CardTitle tag="h5">
                                    Producción energética en el último
día.
                                </CardTitle>
                                <CardSubtitle className="mb-2 text-
muted" tag="h6">
                                    {fechaDia}
                                </CardSubtitle>
```

```
                                <CardText
style={{background:'#E0FFFF'}}>
                                    <canvas id="graficaUltimoDia"
width="800" height="400"></canvas>

                                </CardText>
                            </CardBody>
                        </Card>
                    </div>

                    <hr/>


                </div>



            </div>
        );
    }
}

export default Ultimas;
```

Anexo 16.10 Creación del componente Temperatura.

```
import React from 'react';
import axios from "axios";
import {Chart} from "react-chartjs-2";
let myChartTemp;

function Temperatura (fecha) {

    const temperaturaPost = async() => {
        axios.post('http://www.estacionunipamplona.tk/api/dia-
especifico-temperatura', {
            params: {
                fecha: fecha
            }
        })
            .then(res => {
                generarGraficaTemperatura(res);
                //this.generarGraficaDiaTemperatura(res);
            })
            .catch(err => {
                console.log(err);
            })
    };
    temperaturaPost();


    const generarGraficaTemperatura = (datos) => {
        let ctx =
document.getElementById("graficaTemperatura").getContext('2d');
        if(myChartTemp){
            myChartTemp.destroy();
        };
```

```
        myChartTemp = new Chart (ctx, {
            type: "line",
            data: {
                labels: datos.data.horasDiaTemp,
                datasets: [{
                    label: 'TEMPERATURA en °C',
                    borderWidth: 0.5,
                    pointRadius: 1.5,
                    pointHoverRadius: 1,
                    backgroundColor: "rgba(6,90,131,1)",
                    borderColor: "rgba(6,90,131,1)",
                    data: datos.data.tempDia
                },{
                    label: 'Humedad en %',
                    backgroundColor: 'rgba(255,0,0,1)',
                    fill: false,
                    pointRadius: 1.2,
                    pointHoverRadius: 1,
                    hoverBackgroundColor: 'rgba(255,180,0,0.5)',
                    data: datos.data.humDia
                }]
            },
            options: {
                responsive: true
            }

        });
        console.log(datos.data);
    }

        return (
            <div>


            </div>
        );

};


function TemperaturaSemana (fechaIni, fechaFin) {

    const temperaturaPost = async() => {
        axios.post('http://www.estacionunipamplona.tk/api/semana-
especifica-temperatura', {
            params: {
                fechaIni: fechaIni,
                fechaFin: fechaFin
            }
        })
            .then(res => {
                generarGraficaTemperaturaSemana(res);
                //this.generarGraficaDiaTemperatura(res);
            })
            .catch(err => {
                console.log(err);
```

```
            })
    };
    temperaturaPost();


    const generarGraficaTemperaturaSemana = (datos) => {
        let ctx =
document.getElementById("graficaTemperatura").getContext('2d');
        if(myChartTemp){
            myChartTemp.destroy();
        };

        myChartTemp = new Chart (ctx, {
            type: "line",
            data: {
                labels: datos.data.horasSemanaTemp,
                datasets: [{
                    label: 'TEMPERATURA en °C',
                    borderWidth: 0.5,
                    pointRadius: 1.5,
                    pointHoverRadius: 1,
                    backgroundColor: "rgba(6,90,131,1)",
                    borderColor: "rgba(6,90,131,1)",
                    data: datos.data.tempSemana
                },{
                    label: 'Humedad en %',
                    backgroundColor: 'rgba(255,0,0,1)',
                    fill: false,
                    pointRadius: 1.2,
                    pointHoverRadius: 1,
                    hoverBackgroundColor: 'rgba(255,180,0,0.5)',
                    data: datos.data.humSemana
                }]
            },
            options: {
                responsive: true
            }

        });
        console.log(datos.data);
    }


    return (
        <div>


        </div>
    );

};

const funciones = {Temperatura, TemperaturaSemana};
export default funciones;
```

Anexo 17. Creación del hook: useSocket.

```
import io from "socket.io-client";
import {useEffect, useMemo, useState} from "react";

export const useSocket = (serverPath) => {
    const socket = useMemo( () => io.connect(serverPath, {
        transports: ['websocket']
    }), [serverPath]);

    const[online, setOnline] = useState(false);

    useEffect(() => {
        setOnline(socket.connected);
    }, [socket])
    useEffect( () => {
        socket.on('connect', () => {
            setOnline(true);
        })
    }, [socket])
    useEffect( () => {
        socket.on('disconnect', () => {
            setOnline(false);
        })
    }, [socket])


    return {
        socket,
        online,
    }
}
```

Anexo 18. Creación del hook context.

```
import React, {createContext} from 'react';
import {useSocket} from "../hooks/useSocket";

export const SocketContext = createContext();

export const SocketProvider = ({ children }) => {

    const { socket, online } =
useSocket('http://www.estacionunipamplona.tk');

    return (
        <SocketContext.Provider value={{socket, online}}>
            { children }
        </SocketContext.Provider>
    )
}
```

Anexo 19. Configuración de index.html.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1"
/>
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="icon" href="./icono.png" />
    <!--
      manifest.json provides metadata used when your web app is
installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during
the build.
      Only files inside the `public` folder can be referenced from the
HTML.

      Unlike "/favicon.ico" or "favicon.ico",
"%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root
public URL.
      Learn how to configure a non-root public URL by running `npm run
build`.
    -->
    <title>Monitoreo remoto</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty
page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body>
tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn
build`.

    -->
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.
min.js" integrity="sha384-
IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
```

```
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/bootstrap.min
.js" integrity="sha384-
lpyLfhYuitXl2zRZ5Bn2fqnhNAKOAaM/0Kr9laMspuaMiZfGmfwRNFh8HlMy49eQ"
crossorigin="anonymous"></script>
  </body>
</html>
```

Anexo 20. Creación de histograma en Python de la velocidad de viento.

```
import requests
import json
import math
import matplotlib.pyplot as plt

from collections import Counter

#body = {'params' : {'fecha' : "2021-11-11T05:00:00.000Z"}}
body = {'params' : {'fechaIni' : "2021-11-11T05:00:00.000Z", 'fechaFin' : "2021-11-
15T05:00:00.000Z"}}
json.dumps(body)
response = requests.post('http://estacionunipamplona.tk/api/semana-especifica', json = body)

diccionario = response.json()
potenciaDia = diccionario['potenciaSemana']
horasDiaPot = diccionario['horasSemanaPot']
velocidadDia = diccionario['velocidadSemana']
horasDiaVel = diccionario['horasSemanaVel']

print(len(velocidadDia),' Cantidad de datos de velocidad del viento.')

datosImprimir = "Número de datos: {}".format(len(velocidadDia))

intervalos = round(1+(3.3*math.log(len(velocidadDia),10)))

plt.hist(x=velocidadDia, bins=intervalos, color='#EA304A', rwidth=0.85)
plt.title('Histograma de velocidades del 11-11-2021 a 15-11-2021.')
plt.xlabel('Velocidad del viento')
plt.ylabel('Frecuencia')
plt.grid(linestyle='dotted',linewidth=0.5)
plt.text(2.7,12000,datosImprimir,fontsize=8)

plt.show() #Se dibuja el histograma
```

Anexo 21. Creación de histograma en Python de la potencia producida.

```
import requests
```

```
import json
import math
import matplotlib.pyplot as plt

from collections import Counter

#body = {'params' : {'fecha' : "2021-11-11T05:00:00.000Z"}}
body = {'params' : {'fechaIni' : "2021-11-11T05:00:00.000Z", 'fechaFin' : "2021-11-
15T05:00:00.000Z"}}
json.dumps(body)
response = requests.post('http://estacionunipamplona.tk/api/semana-especifica', json = body)

diccionario = response.json()
potenciaDia = diccionario['potenciaSemana']
horasDiaPot = diccionario['horasSemanaPot']
velocidadDia = diccionario['velocidadSemana']
horasDiaVel = diccionario['horasSemanaVel']

print(len(velocidadDia),' Cantidad de datos de potencia.')

datosImprimir = "Número de datos: {}".format(len(potenciaDia))

intervalos = round(1+(3.3*math.log(len(potenciaDia),10)))

plt.hist(x=potenciaDia, bins=intervalos, color='#60F543', rwidth=0.85)
plt.title('Histograma de Potencia producida del 11-11-2021 a 15-11-2021')
plt.xlabel('Potencia producida')
plt.ylabel('Frecuencia')
plt.grid(linestyle='dotted',linewidth=0.5)
plt.text(0.62,1700,datosImprimir,fontsize=8)

plt.show() #Se dibuja el histograma
```

Anexo 22. Creación de histograma en Python de la temperatura.

```
import requests
import json
import math
import matplotlib.pyplot as plt
import pandas as pd

from collections import Counter

#body = {'params' : {'fecha' : "2021-11-11T05:00:00.000Z"}}
body = {'params' : {'fechaIni' : "2021-11-07T05:00:00.000Z", 'fechaFin' : "2021-11-
20T05:00:00.000Z"}}
json.dumps(body)
```

```
respuesta = requests.post('http://estacionunipamplona.tk/api/semana-especifica-temperatura',
json = body)
print(respuesta.text)

diccionario = respuesta.json()

temperaturaDia = diccionario['tempSemana']
horasDiaTemp = diccionario['horasSemanaTemp']
humedadDia = diccionario['humSemana']
horasDiaHum = diccionario['horasSemanaHum']

print(len(temperaturaDia),' Cantidad de datos de temperatura.')

datosImprimir = "Número de datos: {}".format(len(temperaturaDia))

intervalos = round(1+(3.3*math.log(len(temperaturaDia),10)))

plt.hist(x=temperaturaDia, bins=intervalos, color='#246AED', rwidth=0.85)
plt.title('Histograma de temperatura del 11-11-2021 a 15-11-2021.')
plt.xlabel('Temperatura en °C')
plt.ylabel('Frecuencia')
plt.grid(linestyle='dotted',linewidth=0.5)
plt.text(0.25,9000,datosImprimir,fontsize=8)
plt.show() #Se dibuja el histograma
```

Anexo 23. Creación de histograma en Python de la humedad.

```
import requests
import json
import math
import matplotlib.pyplot as plt
import pandas as pd

from collections import Counter

#body = {'params' : {'fecha' : "2021-11-11T05:00:00.000Z"}}
body = {'params' : {'fechaIni' : "2021-11-11T05:00:00.000Z", 'fechaFin' : "2021-11-
15T05:00:00.000Z"}}
json.dumps(body)
respuesta = requests.post('http://estacionunipamplona.tk/api/semana-especifica-temperatura',
json = body)
print(respuesta.text)

diccionario = respuesta.json()

temperaturaDia = diccionario['tempSemana']
horasDiaTemp = diccionario['horasSemanaTemp']
```

```
humedadDia = diccionario['humSemana']
horasDiaHum = diccionario['horasSemanaHum']

print(len(humedadDia),' Cantidad de datos de humedad.')

datosImprimir = "Número de datos: {}".format(len(humedadDia))

intervalos = round(1+(3.3*math.log(len(humedadDia),10)))

plt.hist(x=humedadDia, bins=intervalos, color='#A73727', rwidth=0.85)
plt.title('Histograma de humedad del 11-11-2021 a 15-11-2021.')
plt.xlabel('Humedad en %')
plt.ylabel('Frecuencia')
plt.grid(linestyle='dotted',linewidth=0.5)
plt.text(29.5,17000,datosImprimir,fontsize=8)

plt.show() #Se dibuja el histograma
```

Anexo 24. Creación de histograma en Python de la velocidad de viento y la potencia producida en la última hora.

```
import requests
import json
import math
import matplotlib.pyplot as plt

from collections import Counter

response = requests.get('http://estacionunipamplona.tk/api/hora-ultima')

diccionario = response.json()
potenciaDia = diccionario['potenciaDia']
horasDiaPot = diccionario['horasDiaPot']
velocidadDia = diccionario['velocidadDia']
horasDiaVel = diccionario['horasDiaVel']

print(len(velocidadDia),' Cantidad de datos de velocidad de viento.')

datosImprimir = "Número de datos: {}".format(len(velocidadDia))

intervalos = round(1+(3.3*math.log(len(velocidadDia),10)))

plt.hist(x=velocidadDia, bins=intervalos, color='#EA304A', rwidth=0.85)
plt.title('Histograma de velocidad de viento en una hora.')
plt.xlabel('Velocidad de viento en m/s')
plt.ylabel('Frecuencia')
```

```
plt.grid(linestyle='dotted',linewidth=0.5)
plt.text(2.6,27,datosImprimir,fontsize=8)

plt.show() #Se dibuja el histograma
```

```
import requests
import json
import math
import matplotlib.pyplot as plt

from collections import Counter

response = requests.get('http://estacionunipamplona.tk/api/hora-ultima')

diccionario = response.json()
potenciaDia = diccionario['potenciaDia']
horasDiaPot = diccionario['horasDiaPot']
velocidadDia = diccionario['velocidadDia']
horasDiaVel = diccionario['horasDiaVel']

print(len(velocidadDia),' Cantidad de datos de potencia.')

datosImprimir = "Número de datos: {}".format(len(potenciaDia))

intervalos = round(1+(3.3*math.log(len(potenciaDia),10)))

plt.hist(x=potenciaDia, bins=intervalos, color='#60F543', rwidth=0.85)
plt.title('Histograma de Potencia producida en una hora.')
plt.xlabel('Potencia producida en Watts')
plt.ylabel('Frecuencia')
plt.grid(linestyle='dotted',linewidth=0.5)
plt.text(2.6,27,datosImprimir,fontsize=8)

plt.show() #Se dibuja el histograma
```