UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

DISEÑO DE LOS MÓDULOS BASE DE DATOS, GRÁFICO DE TENDENCIAS, ALARMAS Y EVENTOS PARA UN SISTEMA SCADA BASADO EN SOFTWARE LIBRE QUE SERÁ IMPLEMENTADO EN EL LABORATORIO DE CONTROL DE LA ESCUELA DE INGENIERÍA ELÉCTRICA DE LA UNIVERSIDAD DE LOS ANDES.

Br. Ana Beatriz Montaña Villegas

Mérida, Marzo, 2022

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

DISEÑO DE LOS MÓDULOS BASE DE DATOS, GRÁFICO DE TENDENCIAS, ALARMAS Y EVENTOS PARA UN SISTEMA SCADA BASADO EN SOFTWARE LIBRE QUE SERÁ IMPLEMENTADO EN EL LABORATORIO DE CONTROL DE LA ESCUELA DE INGENIERÍA ELÉCTRICA DE LA UNIVERSIDAD DE LOS ANDES.

Trabajo de Grado presentado como requisito parcial para optar al título de Ingeniero Electricista

Br. Ana Beatriz Montaña Villegas
Tutor(es): M.Sc Oscar Enrique Blanco Ortiz

Mérida, Marzo, 2022

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

DISEÑO DE LOS MÓDULOS BASE DE DATOS, GRÁFICO DE TENDENCIAS, ALARMAS Y EVENTOS PARA UN SISTEMA SCADA BASADO EN SOFTWARE LIBRE QUE SERÁ IMPLEMENTADO EN EL LABORATORIO DE CONTROL DE LA ESCUELA DE INGENIERÍA ELÉCTRICA DE LA UNIVERSIDAD DE LOS ANDES.

Br. Ana Beatriz Montaña Villegas

Trabajo de Grado, presentado en cumplimiento parcial de los requisitos exigidos para optar al título de Ingeniero Electricista, aprobado en nombre de la Universidad de Los Andes por el siguiente Jurado.

MSc. Francisco J. Viloria M.	Ing. David A. Quintero

Prof. (MSc, Oscar E. Blanco O.)

DEDICATORIA

A Dios todo poderos por permitirme cumplir esta meta.

A mis padres, no tengo palabras para expresarles mi agradecimiento y orgullo, ustedes fueron mi mayor motivación.

A mis Hermanos, gracias por siempre estar a mi lado en las buenas y en las malas, dándome todo su apoyo, motivación y cariño para seguir adelante.

A mis Tíos y Abuelos, gracias por siempre estar pendiente de mí, apoyándome en cada paso que doy y brindándome las palabras necesarias para seguir adelante.

A la Sr. Olga Gonzales, por acogerme en su hogar durante toda mi carrera y por su FE en mí.

A todos mis amigos, y compañeros con los que compartí tantas experiencias y siempre me brindaron su apoyo incondicional.

A la ilustre alma mater, Universidad de los Andes y en especial a la Escuela de Eléctrica de la Facultad de Ingeniería y a todos sus profesores, por ayudarme, enseñarme y guiarme a convertirme en lo que soy ahora.

Ana Beatriz Montaña Villegas. Diseño de los módulos base de datos, gráfico de tendencias, alarmas y eventos para un sistema SCADA basado en software libre que será implementado en el Laboratorio de Control de la Escuela de Ingeniería Eléctrica de la Universidad de Los Andes. Universidad de Los Andes. Tutor(es): MSc. Oscar E. Blanco O. Marzo, 2022.

RESUMEN

Los sistemas SCADA han sido un factor clave en el desarrollo de la automatización industrial, pues estos permiten la supervisión y control de procesos industriales de manera remota y gestionan la evolución de dichos procesos sin la intervención continua de un operador. Debido a que la mayoría de los sistemas actuales son propietarios, en la Escuela de Ingeniería Eléctrica de la Universidad de Los Andes se plantea la elaboración de un sistema SCADA basado en software libre y de código abierto el cual se desarrollará de manera modular de manera que permita en un futuro realizar modificaciones sin mayores restricciones. En el presente trabajo se implementará el desarrollo de los módulos de base de datos el cual permitirá almacenar toda la información del sistema de una forma organizada, contar con interfaces gráficas de usuarios que faciliten la creación, modificación y eliminación de variables y su configuración de alarmas, así como también módulos para la gestión de alarmas y eventos y un módulo que permita la visualización de gráficos de tendencias que existen en el proceso tanto en tiempo real como históricas.

Des*cript*ores: Sistemas SCADA, automatización industrial, supervisión, base de datos, interfaz gráfica de usuario, gráficos de tendencia.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El sistema de supervisión, control y adquisición de datos, (SCADA, *Supervisory Control And Data Adquisition* - por sus siglas en inglés) es un sistema capaz de obtener, procesar y mostrar información de distintos procesos industriales y controlar sus múltiples variables de forma automática desde la pantalla de un computador.

Los sistemas SCADA permiten la integración y comunicación entre los diferentes dispositivos y a su vez, efectúan la tarea de monitoreo de alarmas análogas o digitales, tratamiento de datos y control de proceso en tiempo real, también proveen toda la información que se genera en el proceso de una forma amigable con el usuario accediendo al historial de alarmas, variables de control y bases de datos relacionales. La función de un sistema SCADA es administrar la evolución y actividad de los procesos sin que se requiera la intervención continua de un operador humano.

La arquitectura de un SCADA debe ser abierta, de manera que permita a los usuarios modificar el software sin restricciones y así pueda crecer y adaptarse según las necesidades cambiantes de los distintos procesos. La ventaja del software libre es que mientras haya interesados en continuar el proyecto y gracias al trabajo cooperativo, este seguirá desarrollándose. Mientras que un software propietario, si los promotores deciden abandonar el proyecto, su desarrollo no continuará.

El presente trabajo de grado se centra en la descripción de las herramientas a utilizar durante el desarrollo del sistema, posteriormente en la elaboración e implementación de una base de datos relacional para el almacenamiento de datos presentes en el SCADA ULA, así como también la creación de las distintas variables y sus alarmas, el desarrollo del gestor de alarma, generación de eventos, y un sistema de tendencias para representar gráficamente con información en tiempo real e histórica el comportamiento de los datos.

Este trabajo está dividido en cinco capítulos los cuales se muestran a continuación.

En el capítulo I se ilustra el planteamiento del problema, los objetivos, la justificación, el alcance y limitaciones para el desarrollo de los módulos del sistema SCADA.

En el capítulo II se exponen los antecedentes relacionados con los sistemas SCADA y se definen los conceptos necesarios para entender el sistema, los programas y herramientas necesarias para el desarrollo cada uno de los módulos.

En el capítulo III se realiza la comparación y elección de software libre más adecuado para el desarrollo de los distintos módulos del sistema.

En el capítulo IV se presenta la metodología implementada para el desarrollo de los distintos módulos y los pasos a seguir para la construcción del software.

El capítulo V contiene el análisis de los resultados obtenidos de la implementación del software y su funcionalidad, así como la integración con el módulo de Interfaz Humano Maquina (HMI)

Finalmente, se presentan las conclusiones y recomendaciones del presente trabajo de grado.

CAPÍTULO I

GENERALIDADES

En este capítulo se describen los motivos por el cual se ejecutará el presente trabajo, además, también se explicarán los objetivos perseguidos para el desarrollo modular del SCADA-ULA

1.1 PLANTEAMIENTO DEL PROBLEMA

En los últimos años, la automatización de procesos ha evolucionado de una manera significativa a nivel mundial, pasando a ser una herramienta indispensable en las industrias a la hora de competir en el mercado globalizado, generando beneficios como reducción de costos, aumento de productividad, confiabilidad y rendimiento, sustituyendo los procesos manuales con aplicaciones de software tales como los sistemas SCADA. Antes de la pandemia del COVID-19, se sabían los beneficios de la automatización y muchas empresas contaban con esta para mejorar la eficiencia y productividad, sin embargo, la necesidad de trabajar de manera remota aumento considerablemente la digitalización de procesos industriales, convirtiéndose en un elemento necesario para la sobrevivencia de las empresas ya que la automatización de procesos le permite mejorar la experiencia de sus clientes y empleados.

A nivel mundial, se está observando un posicionamiento cada vez más importante del software libre, en Venezuela, la mayoría de las empresas funcionan de manera manual y/o bajo software propietario, una de las consecuencias de los software propietarios es que, debido al alto costo de sus licencias para la implementación de los sistemas SCADA, se han visto afectadas tanto las pequeñas y medianas industrias (PYMES) como las instituciones educativas a la hora de enseñar al estudiantado su aplicación y uso en el sector industrial.

La Escuela de Ingeniería Eléctrica de la Universidad de Los Andes, cuenta con un Laboratorio de Control automático de procesos que es utilizado con fines educativos, el mismo presente

algunas deficiencias debido a la antigüedad de los equipos con que cuenta, así mismo, cuenta con una serie de Controladores Lógicos Programables (PLC), que son usados en las prácticas de laboratorio, sin embargo, no ha sido posible implementar un sistema SCADA motivado a que no se tiene un software con las licencias necesarias que permitan su uso. En este sentido, el presente trabajo de investigación, se centra en el diseño e implementación de un sistema SCADA basado en software libre, que aporten a los estudiantes un mayor conocimiento en el área de automatización y control industrial y permita mejorar el proceso de enseñanza y aprendizaje para los estudiantes de la carrera de Ingeniería Eléctrica a través de un software que no tenga restricciones de licencia y que pueda ser instalado en los equipos existentes en el laboratorio.

El sistema SCADA se desarrolla de manera modular, este tendrá grandes ventajas sobre el software propietario, ya que no requiere la compra de licencias lo cual permite reducir costos, beneficia la educación a distancia, y mejora las prácticas de Laboratorio de Control para que los estudiantes estén a la vanguardia tecnológica y cuenten con los conocimientos necesarios en dicha área.

El siguiente trabajo está orientado a desarrollar los módulos base de datos, gráfico de tendencias, alarmas y eventos de dicho sistema usando para ellos plataformas de desarrollo basadas en software libre. De esta manera, los estudiantes tendrán a su alcance una herramienta que les permitirá realizar determinadas prácticas para aprovechar los equipos que se encuentran en el Laboratorio de Control, así como también aplicar los conocimientos teóricos adquiridos en las diferentes clases impartidas por el facilitador.

1.2 JUSTIFICACIÓN

La evolución de la automatización ha permitido incorporar nuevas estrategias para hacer los procesos industriales más productivos, eficientes, fiables y rápidos. Para alcanzar este objetivo, utilizan software que permiten ejercer un control y supervisión de los elementos de campo de una manera más efectiva a medida que avanza el desarrollo tecnológico actual.

Los sistemas SCADA son empleados por industrias y empresas en el sector público y privado con el fin de monitorizar y controlar las distintas variables de procesos en tiempo real, dicho sistema permite la conexión de una gran cantidad de sensores, almacena grandes cantidades de

datos en bases de datos para su posterior análisis y uso en generación de gráficos de tendencias e históricos entre otros.

Debido al alto costo de las licencias propietarias, en la Escuela de Ingeniería Eléctrica de la Universidad de Los Andes se toma una alternativa de implementar un sistema SCADA basado en software libre que permita aprovechar los equipos disponibles en el laboratorio, mejorando las prácticas del Laboratorio de Control y ayudando a que el contenido académico esté más actualizado, beneficiando así a los estudiantes que cursan el Laboratorio de Control.

1.3 OBJETIVOS

1.3.1 Objetivo General

- Diseñar los módulos base de datos, gráficos de tendencia, alarmas y eventos a ser implementado a través del software SCADA-ULA en el Laboratorio de Control de la Escuela de Ingeniería Eléctrica de la Universidad de Los Andes.

1.3.2 Objetivos Específicos

- Analizar el funcionamiento de los módulos base de datos, gráficos de tendencia, alarmas y eventos para el sistema SCADA más adecuado.
- Seleccionar la herramienta de programación en software libre más adecuada para el diseño de los módulos base de datos, gráficos de tendencia, alarmas y eventos para sistemas SCADA.
- Diseñar los módulos base de datos, gráficos de tendencia, alarmas y eventos en software libre para desarrollo el sistema SCADA en el Laboratorio de Control de la escuela de ingeniería eléctrica
- Integrar los módulos desarrollados con el módulo HMI del sistema SCADA.

1.4 METODOLOGÍA

De acuerdo con las características de investigación y desarrollo que explica [1], Este trabajo de grado se concibe dentro de la siguiente modalidad: “Proyecto Factibles”. En este sentido [1], define el proyecto factible como un estudio “que consiste en la investigación, elaboración y

desarrollo de una propuesta de un modelo operativo viable para solucionar problemas, requerimientos o necesidades de organizaciones o grupos sociales".

De igual manera [2], considera que un proyecto factible está orientado a resolver un problema planteado o a satisfacer las necesidades en una institución. De acuerdo a las definiciones mencionadas previamente se deduce que, un proyecto factible consiste en un conjunto de actividades, cuya ejecución permitirá el desarrollo de un proyecto, que tendrá la finalidad de atender las necesidades que pueda tener una institución o un grupo social en un momento determinado.

Es decir, la finalidad del proyecto factible se basa en el diseño de una propuesta dirigida a resolver un problema o necesidad. En tal sentido este trabajo de grado permite dar solución a una problemática existente en el Laboratorio de Control de la faculta de Ingeniera Eléctrica, con el diseño de un proyecto viable que beneficiará tanto el laboratorio como a los estudiantes, debido a que maximizará el uso de los equipos de cómputo obsoletos que se encuentran en él laboraría, y a su vez permitirá reforzar las prácticas impartidas a los estudiantes.

1.5 ALCANCE

El sistema tendrá la capacidad de agregar, modificar y eliminar las distintas variables para cada uno de los procesos, se establece la comunicación con un cliente el cual envía constantemente datos al servidor, simulado a través de valores aleatorios, los cuales permite general alarmas y mostrar las gráficas en tiempo real. Será desarrollado bajo software libre, lo cual permitirá a los usuarios modificar el software sin restricciones de acuerdo a las necesidades cambiantes del sistema.

El sistema será implementado en el Laboratorio de Control de la Escuela de Ingeniería Eléctrica de la Universidad de Los Andes, con el fin de mejorar las prácticas de laboratorio y aprovechar tanto equipos existentes como los conocimientos teóricos impartidos por el facilitador.

CAPÍTULO II
MARCO TEÓRICO

Con las bases teóricas se pretende definir y explicar sistemáticamente algunos conceptos teóricos que giran en torno a la investigación.

2.1 ANTECEDENTES

Con objeto de proporcionar un mejor conocimiento en lo referente a estudios previos relacionados con la investigación se presentan los siguientes antecedentes:

Sánchez y Custodio [3], en su trabajo titulado "Servidor para un sistema de supervisión y control de procesos industriales bajo software libre" plantean el desarrollo de un sistema servidor para un SCADA basado en software libre, el cual permita la realización de reportes de las diferentes variables del proceso, así como también la configuración de las alarmas de una aplicación y la creación, edición y eliminación de los usuarios de un proceso determinado, la aplicación también les permitía la realización de historiales de forma gráfica. Los autores concluyeron que el servidor desarrollado para la supervisión y control de procesos industriales funciono operando de una forma integrada, permitiéndoles así verificar en correcto funcionamiento de cada módulo del sistema central

Abaffi y Lárez [4], en su trabajo presentado a la Universidad Nacional Experimental de Guayana titulado "Desarrollo de SCADA en una plataforma de software libre", se centraron en los elementos claves para el desarrollo de un sistema SCADA en software libre, realizado en un proceso particular para una línea de celdas de reducción, extrayendo los elementos que se pueden implementar en una arquitectura general y que deben ser considerados para el desarrollo de un SCADA en software libre, como lo son: el acceso en tiempo real a toda la información disponible en los controladores, información gráfica de todos los parámetros, cumplir con los requerimientos y seguimientos en tiempo real del sistema, y las operaciones realizadas, así como

también el almacenamiento en base de datos de histórico. Los autores plantean la separación en cuanto a software de una memoria compartida y una base de datos, donde el sistema buscará en la memoria compartida los valores instantáneos o pequeñas tendencias mientras que en la base de datos ubicará el historial con los detalles necesarios para su posterior análisis, lo cual le permitió satisfacer los requerimientos de rendimiento en tiempo real y de disponibilidad de información en cualquier momento.

E. Pérez [5], en su artículo "Los sistemas SCADA en la automatización industrial" se enfoca en la importancia de los sistemas SCADA como un aspecto fundamental de la automatización de los procesos de manufactura en la industria actual, los cuales permiten al ser humano interactuar con los procesos en los diferentes tipos de industria sin necesidad de asumir riesgos en la planta, ya que facilitan el control y toma de decisiones de manera remota desde una cabina de mando. Este tipo de software permite ilustrar gráficamente los procesos productivos en pantalla y crear alarmas y advertencias en tiempo real, para el manejo del proceso que se desea controlar. El autor concluye que gracias a la perspectiva que ofrecen los sistemas SCADA en todos los recursos de control e información, los ingenieros, supervisores, gerentes y operadores pueden visualizar e interactuar con los procesos mediante representaciones gráficas. Dicho software evita que el ser humano realice trabajos repetitivos que fácilmente puede realizar o supervisar una máquina o un software y de esta manera evitar grandes riesgos mediante la adopción de la tecnología que mejor se ajusta en cada caso.

2.2 MARCO CONCEPTUAL

2.2.1 SCADA

Los sistemas SCADA se tratan de una aplicación de software que se ejecutan sobre el control de un proceso, comunica los dispositivos de campo, permite el acceso a datos remotos y controla el proceso a distancia, proporcionando información en tiempo real [6].

SCADA son las siglas de *Supervisory Control And Data Adquisition*. Algunos autores lo definen como la tecnología que habilita la colección de datos de locaciones remotas, así como el envío de información a estas locaciones.

Un sistema SCADA permite que un operador, ubicado en una estación central a grandes distancias de la ubicación de los procesos industriales, pueda hacer ajustes o cambios en los controladores locales de los procesos. Tal es el caso de abrir o cerrar válvulas a distancias, conocer el estado de los interruptores de seguridad de un sistema, monitorear el estado de las alarmas del proceso y obtener información de las variables del proceso involucradas, [7].

Se da el nombre de SCADA cualquier software que permita el acceso a datos remotos de un proceso y permita, utilizando las herramientas de comunicación necesarias en cada caso, el control del mismo.

Atendiendo a la definición, se observa que no se trata de un sistema de control, sino de una utilidad software de monitorización o supervisión, que realiza la tarea de interfaces entre los niveles de control (PLC) y los de gestión a un nivel superior, [8].

- **Requisitos**

Según [7], un SCADA debe cumplir varios objetivos para que su instalación sea perfectamente aprovechada:

Deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa.

Deben comunicarse con total facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa (redes locales y de gestión).

Deben ser programas sencillos de instalar, sin excesivas exigencias de hardware y fáciles de utilizar, con interfaces amigables para el usuario.

- **Prestaciones**

Según [8], un software de supervisión SCADA debe estar en disposición de ofrecer las siguientes prestaciones:

Posibilidad de crear paneles de alarma, que exigen la presencia del operador para reconocer una parada o situación de alarma, con registro de incidencias.

Generación de históricos de señal de planta, que pueden ser volcados para su proceso sobre una hoja de cálculo.

Ejecución de programas que modifican la ley de control o incluso, el programa total sobre el autómata, bajo ciertas condiciones.

Posibilidad de programación numérica, que permite realizar cálculos aritméticos de elevada resolución sobre la Unidad Central de Procesamientos (CPU) del ordenador y no sobre la del autómata, menos especializado, entre otros.

2.2.2 Alarmas y Eventos

Según [7], Las alarmas se basan en la vigilancia de los parámetros de las variables del sistema. Son los sucesos no deseables, porque su aparición puede dar lugar a problemas de funcionamiento. Este tipo de sucesos requiere la atención de un operario para su solución antes de que se llegue a una situación crítica que detenga el proceso, o para poder seguir trabajando.

La norma ISA-18.2 define una alarma como: "Un medio audible y / o visible para indicar al operador un mal funcionamiento del equipo, la desviación del proceso, o una condición anormal que requiere una respuesta".

Una alarma según [9], es un anuncio al operador, normalmente por medios audibles y/o visuales, para alertarlo de alguna condición anormal de la operación que requiere de su intervención para ser corregida, y evitar así consecuencias negativas, tanto del proceso como de la seguridad de las personas, del medio ambiente y de los equipos.

En situaciones en las que se presenten varias alarmas simultáneamente, el o los operadores tendrán la potestad para decidir en qué orden las abordan, en función de las alarmas en cuestión, el estado operacional de la planta y su experiencia. Siempre teniendo en cuenta que la prioridad y el orden de aparición son factores importantes a la hora de tomar dicha decisión, o de interpretar correctamente que sucede y/o porque sucede.

Las situaciones normales, tales como puesta en marcha, paro, cambios de consignas de funcionamiento, consultas de datos, cuando un operador entra en el sistema, entre otros, serán los denominados eventos del sistema o sucesos.

Los eventos no requieren de la atención del operador del sistema, registran de forma automática todo lo que ocurre en el sistema. También será posible guardar estos datos para su consulta a posteriori, [9].

2.2.3 **Gestión y archivo de datos**

Se encarga del almacenamiento y procesamiento ordenado de los datos, pueden seleccionarse datos de planta para ser capturados a intervalos periódicos, como un registro histórico de actividad, o para ser procesados inmediatamente por alguna aplicación software para presentaciones estadísticas, análisis de calidad o mantenimiento. Esto último se consigue con un intercambio de datos dinámico entre el SCADA y el resto de aplicaciones que corren bajo el mismo sistema operativo.

Una vez procesados, los datos se presentan en forma de gráficas analógicas, histogramas, representación tridimensional, entre otros., que permiten después analizar la evolución global del proceso. [5]

2.2.4 Tendencias

Penin Define las tendencias como las utilidades que permiten representar de forma cómoda la evolución de variables del sistema, [8]. Las utilidades más generales son:

- Es posible representar varios valores de forma simultánea en una misma carta. La limitación del número de valores suele ser debida a su inteligibilidad.
- Representación en tiempo "casi real" de variables (*Real time trending*) o recuperación de variables almacenadas (*Historical Trending*).
- Visualización de valores.
- Desplazamiento a lo largo de todo el registro histórico

2.2.5 Base de datos

Se le denomina base de datos a los depósitos de información que incluyen datos referentes a diferentes temas y asuntos de diversas formas. Son datos correspondientes a un mismo contenido y almacenados metódicamente para después ser utilizados.

Una base de datos es un almacén que permite guardar grandes cantidades de información de forma organizada para que luego se puedan encontrar y utilizar fácilmente. Se define como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular.

Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que se quiere guardar en la tabla, cada fila de la tabla conforma un registro, [10].

Las diferentes categorías de bases de datos no son necesariamente excluyentes unas con otras, siendo su más grande diferenciación entre las relacionales y las no relacionales, [11].

- **Bases de datos relacionales**

  Una base de datos relacional es un conjunto de tablas de datos que contienen campos que sirven de nexo de unión (relación) y que permiten establecer múltiples combinaciones mediante la utilización de estos nexos. Las combinaciones posibles son prácticamente ilimitadas, sólo hay que configurar el método de búsqueda (el *query*) o el tipo de datos que se quiere consultar y aplicarlo a los datos, [8].

  El lenguaje predominante en estas bases de datos es el Lenguaje de consulta estructurado (SQL, *Structured Query Language* por sus siglas en ingles). Estas bases de datos permiten relacionar los elementos entre sí de manera muy sencilla y son recomendables cuando los datos a utilizar tienen un margen de error nulo y no requieren modificaciones constantes.

  Según [11], las bases de datos relacionales utilizan el modelo relacional y siempre es mejor usarlas cuando los datos que vas a utilizar son consistentes y ya tienen una estructura planificada.

  Las bases de datos relacionales funcionan bien con datos estructurados. Las organizaciones que tienen muchos datos no estructurados o semiestructurados no deberían considerar una base de datos relacional. Entre las bases de datos relacionales más conocidas se tienen:

  MySQL
  MicrosoftSQL Server
  Oracle *Data*base
  PostgreSQL
  IBM Bb2

- **Bases de datos no relacionales**

  A diferencia de las bases de datos relacionales, los datos de una base de datos NoSQL (*Not Only* SQL) son más flexibles en cuanto a consistencia de datos y se han convertido en una opción que intenta solucionar algunas limitaciones que tiene el modelo relacional. Este tipo de bases de datos es excelente para las organizaciones que buscan almacenar datos no estructurados o semiestructurados, [11].

  Una de las ventajas de las bases de datos NoSQL es que los desarrolladores pueden realizar cambios en la base de datos sobre la marcha, sin que ello afecte a las aplicaciones que la utilizan. Entre estas se encuentran:

  MongoDB
  Redis
  Apache Cassandra
  Apache CouchDB
  CouchBase

- **Bases de datos en las nubes**

  La principal característica de esta categoría es que las bases de datos se entregan como un servicio desde la nube, por lo que su correcta creación, mantenimiento y escalabilidad son competencia del proveedor de este servicio. Este tipo de bases de datos ha crecido exponencialmente con la era de internet y la infraestructura como servicio (IaaS, *Infrastructure as a Service*). Ejemplo:

  Google Firebase
  Microsoft Azure SQL Database
  Amazon Relational Database Service
  Oracle Autonomous Database

2.2.6 Sistema Gestor de Base de Datos (SGBD)

Los sistemas de gestión de base de datos (SGBD, en inglés, *data base management system*) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el

usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta [10].

Según [12], un SGBD relacional es un modelo de datos que facilita a los usuarios describir los datos que serán almacenados en la base de datos junto con un grupo de operaciones para manejar los datos.

En la actualidad, las tres grandes compañías que dominan el mercado de las bases de datos son IBM, Microsoft y Oracle. Como base de datos libres destacan MySQL, MariaDB, PostgreSQL y SQLite3.

- *MySQL*

  MySQL es un sistema gestor de bases de datos relacionales rápido, sólido y flexible. Es idóneo para la creación de bases de datos con acceso desde páginas web dinámicas, así como para la creación de cualquier otra solución que implique el almacenamiento de datos, posibilitando realizar múltiples y rápidas consultas. Está desarrollado en C y C++, facilitando su integración en otras aplicaciones desarrolladas también en esos lenguajes.

  Es un sistema cliente/servidor, por lo que permite trabajar como servidor multiusuario y de subprocesamiento múltiple, o sea, cada vez que se crea una conexión con el servidor, el programa servidor establece un proceso para manejar la solicitud del cliente, controlando así el acceso simultáneo de un gran número de usuarios a los datos y asegurando el acceso a usuarios autorizados solamente. Es uno de los sistemas gestores de bases de datos más utilizado en la actualidad, utilizado por grandes corporaciones como Yahoo! Finance, Google, Motorola, entre otras.

- *Microsoft SQL Server*

  SQL Server es un sistema gestor de base de datos relacionales producido por Microsoft. Es un sistema cliente/servidor que funciona como una extensión natural del sistema operativo Windows. Entre otras características proporciona integridad de datos, optimización de consultas, control de concurrencia y respaldo *(backup)* y recuperación.

  Es relativamente fácil de administrar a través de la utilización de un entorno gráfico para casi todas las tareas de sistema y administración de bases de datos. Utiliza servicios del sistema

operativo Windows para ofrecer nuevas capacidades o ampliar la base de datos, tales como enviar y recibir mensajes y gestionar la seguridad de la conexión. Es fácil de usar y proporciona funciones de almacenamiento de datos que sólo estaban disponibles en Oracle y otros sistemas gestores de bases de datos más caros [12].

- *PostgreSQL*

PostgreSQL es un poderoso sistema de base de datos relacional de objetos de código abierto que usa y extiende el lenguaje SQL combinado con muchas características que almacenan y escalan de manera segura las cargas de trabajo de datos más complicadas. Los orígenes de PostgreSQL se remontan a 1986 como parte del proyecto Postgres en la Universidad de California en Berkeley y tiene más de 30 años de desarrollo activo en la plataforma central.

Además de ser gratuito y de código abierto, PostgreSQL es altamente extensible. Por ejemplo, puede definir sus propios tipos de datos, crear funciones personalizadas, incluso escribir código desde diferentes lenguajes de programación sin volver a compilar su base de dato [13].

El código fuente se encuentra disponible para todos sin costo alguno. Está disponible para 34 plataformas con la última versión estable. Es totalmente compatible con ACID (Acrónimo de Atomicity, Consistency, Isolation and Durability; en español: atomicidad, consistencia, aislamiento y durabilidad).

PostgreSQL posee una integridad referencial e interfaces nativas para lenguajes como ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG; Python y Ruby. Funciona en todos los sistemas operativos Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Debido a la liberación de la licencia, PostgreSQL se puede usar, modificar y distribuir de forma gratuita para cualquier fin, ya sea privado, comercial o académico. [14]

Tamaño máximo de la base de datos	Ilimitado
Tamaño máximo de una tabla	32 terabytes
Tamaño máximo de un registro	1,6 terabytes
Tamaño máximo de una celda	1 gigabytes
Número máximo de columnas	En función del tipo de dato, de 250 a 1 600
Número máximo de filas	Ilimitado
Número máximo de índices	Ilimitado

Figura 2.1 Capacidad de Base de Datos PostgreSQL [12]

2.2.7 Lenguaje de Desarrollo SQL

La aparición del estándar por excelencia para la comunicación con bases de datos, SQL, permite una interfaz común para el acceso a los datos por parte de cualquier programa que se ciña al estándar SQL. El primer SQL aparece en 1986 bajo el nombre: ANSI X3.135-1986 [8]

El lenguaje SQL es el más universal en los sistemas de base de datos. Este lenguaje permite realizar consultas a bases de datos para mostrar, insertar, actualizar y borrar datos.

Según [15], SQL es un lenguaje de base de datos normalizado, utilizado por la gran mayoría de los servidores de bases de datos que manejan bases de datos relacionales u objeto-relacionales. En una base de datos relacional, los resultados de la consulta van a ser datos individuales, tuplas o tablas generados a partir de consultas en las que se establecen una serie de condiciones basadas en valores numéricos.

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

2.2.8 Lenguajes de Programación

Según [16], es un conjunto de comandos e instrucciones digitales que utilizan sintaxis específicas para crear aplicaciones informáticas. Estos lenguajes de programación se clasifican de la siguiente manera

Lenguaje de programación de alto nivel: Este tipo de lenguaje escribe códigos con palabras y símbolos de uso común en las conversaciones diarias. Es por eso que los códigos de lenguaje de programación de alto nivel son fáciles de leer y examinar.

Lenguaje de programación de bajo nivel: Este tipo de lenguaje de programación consta de lenguajes de máquina y lenguajes ensambladores con instrucciones peculiares. A diferencia del lenguaje de programación de alto nivel, escriben códigos complicados que son difíciles de leer. Un programa llamado ensamblador traduce el código ensamblador a código de máquina que la CPU entiende, pero ininteligible para los humanos.

2.2.9 *Frontend*

Frontend es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.

Para el desarrollo del *frontend* se debe tener conocimiento del lenguaje de marcas de hipertexto (HTML, *Hypertext markup language*) y las hojas de Estilo en Cascada (CSS, del inglés Cascading Style Sheets), los lenguajes de maquetación que permiten definir la estructura y estilos de una página web. También JavaScript, un lenguaje de programación para definir la lógica de la aplicación, recibir las solicitudes de los usuarios y enviárselos al b*ackend*. [17]

2.2.10 HTML

HTML podría ser traducido como lenguaje de formato de documentos para hipertexto. Es un lenguaje de programación que se utiliza para el desarrollo de páginas de Internet. El código HTML se crea a partir de etiquetas, también llamadas *tags*, que permiten conectar diversos conceptos y formatos. Para la escritura de este lenguaje, se crean etiquetas que aparecen especificadas a través de corchetes o paréntesis angulares “<>”. Entre sus componentes, los

elementos dan forma a la estructura esencial del lenguaje, ya que tienen dos propiedades (el contenido en sí mismo y sus atributos). [18]

Las partes principales del elemento son:

La etiqueta de apertura: consiste en el nombre del elemento encerrado por paréntesis angulares (< >) de apertura y cierre. Establece dónde comienza o empieza a tener efecto el elemento.

La etiqueta de cierre: es igual que la etiqueta de apertura, excepto que incluye una barra de cierre (/) antes del nombre de la etiqueta. Establece dónde termina el elemento.

El contenido: este es el contenido del elemento.

El elemento: la etiqueta de apertura, más la etiqueta de cierre, más el contenido equivale al elemento.

Los atributos contienen información adicional acerca del elemento que se quiere en el contenido real del elemento.

Un atributo debe tener siempre:

Un espacio entre este y el nombre del elemento (o del atributo previo, si el elemento ya posee uno o más atributos).

El nombre del atributo, seguido por un signo de igual (=).

Comillas de apertura y de cierre, encerrando el valor del atributo.

Los atributos siempre se incluyen en la etiqueta de apertura de un elemento, nunca en la de cierre. [19]

En este lenguaje no existen reglas para especificar los nombres de las etiquetas que se utilizarán al programar, por eso se dice que es un sistema de formato abierto.

Por otra parte, cabe destacar que el HTML permite ciertos códigos que se conocen como *script*s, los cuales brindan instrucciones específicas a los navegadores que se encargan de procesar el lenguaje. Entre los *script*s que pueden agregarse, los más conocidos y utilizados son JavaScript y CSS [18]

Figura 2.2 HTML [18]

2.2.11 CSS

Mientras que el HTML estructura el documento e indica a los navegadores cuál es la función de un elemento concreto, el CSS da instrucciones al navegador sobre cómo debe mostrar un elemento concreto: estilo, espaciado y posición. Si el HTML son los puntales y los ladrillos que forman la estructura de una casa, el CSS es el yeso y la pintura que la decoran.

Esto se consigue al utilizar un sistema de reglas que dicen qué elementos de HTML deben tener estilos añadidos, y en cada regla enumeran las propiedades (por ejemplo, color, tamaño, tipo de letra, etc.) de aquellos elementos HTML que quieren manipular y los valores nuevos que les quieren aplicar. [20]

Figura 2.3 CSS [20]

2.2.12 JavaScript

JavaScript es un lenguaje de secuencias de comandos que te permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes, entre otros. JavaScript fue creado por la compañía de software *Netscape Corporation* para que fuese colocado en su navegador 2.0 y que, gracias a su simplicidad, aún continúa siendo una de las herramientas de gran utilidad, para la creación de páginas web que posean algo más que texto. [21].

JavaScript es la tercera pieza fundamental del desarrollo web *frontend*, junto con los lenguajes HTML y CSS. Cada uno de estos tres lenguajes tiene una función muy concreta en el desarrollo web:

- El HTML se utiliza para conformar el esqueleto y la estructura de los contenidos de una página web.
- El CSS define el estilo y la apariencia web.
- JavaScript rompe con la esteticidad del HTML y permite crear elementos dinámicos e interactivos, mejorando ampliamente la interacción de los usuarios con una página web.

Es importante dejar claro, que JavaScript no es del todo un lenguaje de programación, sino más bien un lenguaje de *script* (rutinas o guiones). Por lo tanto, es más parecido a los macros de los procesadores de hojas de cálculo o texto. Sería imposible ejecutar un programa completo con JavaScript.

Los JavaScript ayudan a mejorar la gestión cliente/servidor; entre sus funciones básicas se encuentran: abrir y cerrar ventanas; cambios eficaces en una página (en lo que respecta a su contenido y aspecto; desarrollo de cadenas de texto; procedimientos aritméticos. Dado que su misión es extender el HTML, JavaScrip*t* es un lenguaje que contempla ciertas restricciones, que, de manera indirecta, terminan por brindarle seguridad al usuario, [21].

JavaScript ya no es solo un lenguaje de *scripting* del lado del cliente porque tecnologías como Node.js le permiten realizar operaciones del lado del servidor. Node.js depende de los marcos del lado del servidor llamados Express.js para crear una plataforma que permite a los desarrolladores escribir códigos que se ejecutan en el servidor. Una vez que *Express.Js* está en funcionamiento con Node.js, los desarrolladores pueden usar JavaScript como lenguaje de desarrollo de *frontend* y *backend*. También ofrece una interfaz de programación de aplicaciones (API) para crear varias aplicaciones, incluidas aplicaciones móviles, híbridas, web, de una sola página y de varias páginas, [16].

- Estructura básica de una función JavaScript

La estructura básica de una función JavaScript se puede dividir en tres partes:

Estructura: Las funciones en JavaScript se declaran mediante la palabra reservada *function*, seguida del nombre de la función. Después se añaden dos paréntesis () en cuyo interior irán los argumentos *(si procede)*. Por último, las llaves {} que se utilizan para encerrar todas las sentencias o instrucciones que se ordenan y se ejecutan en la función.

Nombre: el nombre de la función debe empezar por una letra minúscula y no puede contener espacios. No se pueden usar palabras reservadas para el lenguaje de programación que usemos.

Argumentos: las variables o constantes a utilizar. Los argumentos son una serie de valores que no se han definido dentro de la función, sino que se reciben de otra parte, de otra función u otro enunciado. Pueden ser números o cadenas de texto indistintamente. Se especifican entre los paréntesis () que van detrás del nombre de la función y van separados por comas. No todas las funciones necesitan argumentos para funcionar, pero son una más de las útiles ventajas que ofrecen las funciones.

Sentencias: las instrucciones de la función *(lo que se quiere hacer)*. Esta es la parte de la función donde se debe incluir las instrucciones y operaciones necesarias para el cometido de la función. Van justo después de los paréntesis encerradas entre llaves { } y separadas entre sí por el símbolo punto y coma *";"*. [22]

Figura 2.4 JavaScript [21]

2.2.13 *Chart.js*

Es una librería *open-source* creada para visualizar de manera simple e interactiva gráficos en el navegador web usando JavaScript. *Chart.js* es una librería sin dependencias para construir gráficos de 8 tipos distintos (gráficos lineales, gráficos de barras, gráficos de área, entre otros). La librería está modulada para no cargar los gráficos que no se van a utilizar. Es compatible con diseños *responsives*, adaptándose en base al ancho de cada dispositivo y puedes cambiar

fácilmente variables como el color o las animaciones para personalizar aún más la interfaz gráfica, [23] .

Figura 2.5 *Chart.js* [23]

2.2.14 *Backend*

El *backend* se refiere a los códigos de computadora que manejan las operaciones del lado del servidor, como la lógica del servidor, las funciones de la base de datos y muchas más. Cuando usa una aplicación, la gran mayoría de los datos que envía y recibe son administrados por el *backend* de la aplicación. Sin embargo, las funciones del *backend* son completamente invisibles para el usuario de la aplicación. [16]

Algunos de los lenguajes de programación para *backend* son Python, Node.js, PHP, Go, C++ y C#. Y así como en el *frontend*, todos estos lenguajes tienen diferentes *framework* que te permiten trabajar mejor según el proyecto que estás desarrollando, como Django, Flask, Express.js, Laravel, Ruby on Rails y ASP.Net, [17].

2.2.15 Python

Es un lenguaje de programación interpretado, multiparadigma y multiplataforma usado, principalmente, en *big data*, inteligencia artificial (AI), *data science*, *frameworks* de pruebas y desarrollo web. Esto lo convierte en un lenguaje de propósito general de gran nivel debido a su extensa biblioteca, cuya colección ofrece una amplia gama de instalaciones.

Python se gestó durante las vacaciones de navidad de 1989, cuando el desarrollador holandés Guido Van Rossum decidió escribir un intérprete para el nuevo lenguaje de *scripting* que venía trabajando. Su amplia experiencia en la implementación del sistema ABC —un lenguaje de programación interactivo, estructurado y de alto nivel— se sumó a su iniciativa por crear un

lenguaje más sencillo, intuitivo y potente. Así, en 1991, nació Python conocido en la actualidad como el sucesor del lenguaje ABC [24]

El lenguaje Python se caracteriza por ser simple, rápido y tener una curva de aprendizaje amigable y corta. Está desarrollado bajo una licencia de código abierto, aprobada por la Interconexión de sistemas abiertos (OSI, *Open System Interconnection*), lo que lo hace de libre uso y distribución, incluso para uso comercial. La licencia de Python es administrada por *Python Software Foundation*, [24], [25].

De acuerdo con [16], los desarrolladores de *backend* aprovechan sus códigos nítidos y altamente legibles para crear *script*s funcionales para manejar las asignaciones de *backend*.

Figura 2.6 Python [24]

CARACTERISTICAS

- Relativamente fácil de aprender: uno de los beneficios de Python es el estilo de programación similar al inglés que lo hace altamente legible. Por lo tanto, programar y leer códigos Python es relativamente fácil para programadores nuevos y experimentados.
- Bibliotecas enormes: Python disfruta del soporte de bibliotecas enormes que reducen la necesidad de escribir códigos manualmente. Algunas bibliotecas contienen códigos que mejoran tareas como correo electrónico, navegación, asignaciones de bases de datos, pruebas unitarias y mucho más.
- Rentable: además de que Python es una plataforma de código abierto de descarga gratuita, también ofrece numerosas herramientas y recursos gratuitos que mejoran los proyectos de desarrollo de aplicaciones.
- Códigos integrables**:** con la regla “Escribe una vez, ejecútalo en cualquier lugar” (WORA, del inglés *Write Once Run Anywhere*), el código Python se puede incrustar en el código fuente de otros lenguajes como C++.

LIMITACIONES

- La ejecución del código Python se vuelve lenta cuando se interrumpe. Este déficit perjudica a todo el proyecto de desarrollo de aplicaciones.
- La capa de acceso a la base de datos de Python está menos desarrollada en comparación con otros lenguajes de programación *backend*.
- Depende en gran medida de bibliotecas y marcos de trabajo de terceros.

2.2.16 Node.js

Node.js es un entorno de ejecución de JavaScript multiplataforma de código abierto que se utiliza para ejecutar código JavaScript fuera de un navegador web. Es un excelente marco web para principiantes porque funciona muy bien para aplicaciones con uso intensivo de datos, como aplicaciones de transmisión y en tiempo real, y Node.js facilita el inicio de la construcción del *backend*. [26]

Figura 2.7 Node.js [26]

CARACTERISTICAS

- Tiene JavaScript incorporado, un lenguaje sencillo de aprender y estructurado.
- Es *OpenSource* por lo que, de manera libre, se pueden escoger módulos de sus librerías que faciliten el trabajo. Además, la comunidad de programadores en Node.js crece cada día siendo cada vez más grande.
- Actualmente es una de las plataformas de software más utilizadas superando a entornos y lenguajes conocidos como PHP o C.
- Tiene un menor tiempo de ejecución frente a otros sistemas. Esto lo consigue gracias a los eventos asíncronos y a que no ejecuta el código líneo a línea, sino que procesa lo que puede en cualquier momento.

LIMITACIONES

- Es muy diferente a otros lenguajes más estructurados, esto hace que el tiempo de aprendizaje pueda alargarse para programadores acostumbrados a otros sistemas. Sin embargo, resultará sencillo para los que ya conozcan el lenguaje JavaScript.
- Por su carácter formado a partir de los eventos asíncronos, no tiene la misma potencia de cálculo frente a otros lenguajes como, por ejemplo, Java.

2.2.17 C++

C++ es un lenguaje de programación que proviene de la extensión del lenguaje C para que pudiese manipular objetos. Fue diseñado a mediados de los años 80 por el danés Bjarne Stroustrup. Su intención fue la de extender el lenguaje de programación C para que tuviese los mecanismos necesarios para manipular objetos. Por lo tanto, C++ contiene los paradigmas de la programación estructurada y orientada a objetos, por lo que se le conoce como un lenguaje de programación multiparadigma.

A C++ primero se le conoció como "C con clases". Luego se cambió a C++ que significa "incremento de C", dando a entender que se trata de una extensión del lenguaje de programación C. [27]

Figura 2.8 C++

CARACTERISTICAS

- Compatibilidad con bibliotecas.
- Orientado a objetos: El foco de la programación está en los objetos y la manipulación y configuración de sus distintos parámetros o propiedades.
- Rapidez: La compilación y ejecución de un programa en C++ es mucho más rápida que en la mayoría de lenguajes de programación.

- Compilación: En C++ es necesario compilar el código de bajo nivel antes de ejecutarse, algo que no ocurre en otros lenguajes.
- Punteros: Los punteros del lenguaje C, también están disponibles en C++.
- Didáctico: Aprendiendo programación en C++ luego es mucho más fácil aprender lenguajes como Java, C#, PHP, JavaScript, etc.

LIMITACIONES

- Es un lenguaje muy amplio (con muchos años y muchas líneas de código), tiene que tener una compilación por plataforma y su depuración se complica debido a los errores que surgen.
- El manejo de librerías es más complicado que otros lenguajes como Java o .Net
- Su curva de aprendizaje muy alta.
- La capacidad de C++ para interactuar con el hardware también es un defecto con la tecnología y su b*ackend*. Un usuario malintencionado puede aprovechar esta laguna para interactuar con el hardware del sistema.

2.2.18 PHP

La sigla PHP identifica a un lenguaje de programación que nació como *Personal Home Page* (PHP) *Tools*. Fue desarrollado por el programador de origen danés Rasmus Lerdorf en 1994 con el propósito de facilitar el diseño de páginas web de carácter dinámico.

El acrónimo recursivo, sin embargo, en la actualidad está vinculado a PHP, (*Hypertext Pre-Processor)*. El lenguaje es desarrollado hoy en día por *The PHP Group* aunque carece de una normativa formal. La *Free Software Foundation*, por lo tanto, considera la licencia PHP como parte del software libre. El lenguaje PHP suele procesarse directamente en el servidor, aunque también puede usarse a través de software capaz de ejecutar comandos y para el desarrollo de otra clase de programas.

Actualmente el PHP suele incrustarse dentro del código HTML de las páginas web y ejecutarse desde un servidor. Se estima que PHP está presente en más de veinte millones de webs y en cerca de un millón de servidores. [28], [16]

Figura 2.9 PHP [28]

CARACTERISTICAS

- PHP es un lenguaje de programación multipropósito que es fácil de usar. Funciona perfectamente con una amplia gama de bases de datos y sistemas operativos. Los marcos modernos, una base de código masiva y la comunidad PHP activa son factores que impulsan la evolución continua de PHP.
- Versátil y de código abierto: hay muchas bibliotecas PHP gratuitas en línea que los desarrolladores pueden aprovechar para desarrollar rápidamente códigos de *backend*. Casi todos los sistemas operativos, como Windows y Linux, son compatibles con PHP. Además, PHP también puede lanzar aplicaciones en cualquier servidor web.
- Económico: PHP está disponible de forma gratuita y cuenta con el apoyo de una vibrante comunidad de desarrolladores.
- Fácil de usar: han surgido muchos marcos PHP que simplifican la programación, eliminando la necesidad de escribir códigos SQL tediosos. Algunos de ellos utilizan el sistema *Object Relational Mapping* (ORM, o mapeo objeto-relacional), que funciona como el modelo-vista-controlador (MVC) para escribir rápidamente funciones del lado del servidor.
- Excelente para principiantes: la simplicidad de PHP lo convierte en un lenguaje ideal para nuevos desarrolladores.
- Funciones de automatización: la función de *scripting* de PHP lo hace útil para crear automatización como autenticación, mapeo de URL, administración de sesiones y muchos más.
- Seguridad incorporada: aunque muchas personas piensan que PHP no es seguro, tiene muchas funciones de seguridad incorporadas que le permiten mitigar varias amenazas.

LIMITACIONES

- La influencia de PHP como tecnología de desarrollo está disminuyendo. Hoy en día, la gente apenas incluye PHP en sus habilidades de desarrollo.
- PHP no puede competir de manera eficiente con tecnologías de desarrollo modernas como Ruby y Python debido a las bibliotecas limitadas.
- Como plataforma de código abierto, PHP es susceptible al mal uso y a la creación de códigos con errores.

2.2.19 C# .NET

C# es un lenguaje de programación desarrollado por Microsoft, orientado a objetos, que ha sido diseñado para compilar diversas aplicaciones que se ejecutan en .NET *Framework.* Se trata de un lenguaje simple, eficaz y con seguridad de tipos. Las numerosas innovaciones de C# permiten desarrollar aplicaciones rápidamente y mantener la expresividad y elegancia de los lenguajes de estilo de C.

La sintaxis viene derivada de C y C+ + y utiliza el modelo de objetos de la plataforma .NET, muy parecido al de Java, aunque incluye mejoras propias de otros lenguajes. Como curiosidad, el nombre de este lenguaje fue inspirado por la escala musical. En ella, la letra C equivale a la nota musical *Do* y el símbolo # significa sostenido, lo que indica que es un semitono más alto. Así, C# sugiere que es superior a C y C++. [29], [16]

Figura 2.10 C# .NET [29]

CARACTERISTICAS

- Desarrollo multiplataforma: las aplicaciones con *backend* C-Sharp pueden ejecutarse en múltiples sistemas operativos, como MacOS, Windows, Linux, etc. Este artículo sobre marcos multiplataforma analiza más el desarrollo multiplataforma.

- Ventaja orientada a objetos: como lenguaje orientado a objetos, el código C# hace uso de clases y relaciones. Este enfoque permite la reutilización de fragmentos de código y una fácil resolución de problemas de código.
- Amplia compatibilidad: las aplicaciones C# presentan compatibilidad con versiones anteriores de sistemas heredados. Por ejemplo, las organizaciones que todavía usan versiones anteriores de marcos de programación encontrarían C# invaluable.
- Función de recolección de basura: C-Sharp tiene una función para eliminar toda la basura del sistema. Esta es una gran característica que acelera la ejecución del programa.

LIMITACIONES

- Como lenguaje de programación de alto nivel, el código C# no puede comunicarse directamente con el hardware.
- C# es muy rígido en comparación con otras tecnologías de *backend* porque solo se ejecuta en el marco .Net y solo se puede instalar y ejecutar desde una computadora con Windows.

2.2.20 *Framework* de Python

Un *framework* es una especie de esqueleto o estructura que le va a servir de base al programador para ordenar toda la información que implica desarrollar un determinado software. Cuando se crea una aplicación, se maneja una gran cantidad de librerías, ficheros de configuración, código fuente. Por eso, con un *framework* se puede estructurar todos esos datos, de manera que se facilite su implementación. Con él se evita repetir código, se puede separar los datos de la interfaz y reducir en gran medida el tiempo que se emplea para desarrollar.

Python cuenta con distintos *frameworks* disponibles. Entre los *frameworks* más utilizados en Python están Django y Flast que son los más reconocidos.

2.2.21 Django

Un proyecto de Django es una base de código de Python que contiene un archivo de configuración de Django. El administrador de Django puede crear un proyecto a través del comando *django-admin startproject NAME*. El proyecto normalmente tiene un archivo llamado *manage.py* en el nivel superior y un archivo de URL raíz llamado *urls.py*. *manage.py* es una versión específica del proyecto *Django-admin* y le permite ejecutar comandos de administración

en ese proyecto. Por ejemplo, para ejecutar su proyecto localmente, use *Python manage.py runserver*. Un proyecto se compone de aplicaciones Django. [30].

Una aplicación de Django es un paquete de Python que contiene un archivo de modelos (*Models.py* de forma predeterminada) y otros archivos, como URL y vistas específicas de la aplicación. Se puede crear una aplicación a través del comando *Django-admin startapp NAME* (este comando debe ejecutarse desde dentro del directorio de su proyecto). Para que una aplicación sea parte de un proyecto, debe incluirse en la *INSTALLED_APPS* lista en *settings.py*. Si se seleccionó la configuración estándar, Django viene con varias aplicaciones propias preinstaladas que se encargarán de cosas como la autenticación por usted. Las aplicaciones se pueden usar en múltiples proyectos de Django, [31].

Django-admin es una herramienta de línea de comandos que viene con Django. Viene con varios comandos útiles para comenzar y administrar un proyecto de Django. El comando es el mismo que *manage.py*, con la diferencia de que no es necesario que esté en el directorio del proyecto. Es necesario establecer *DJANGO_SETTINGS_MODULE* la variable de entorno. [31]

El Mapeo de Objeto Relacional (ORM, *Object Relational Mapping*) de Django recopila todos los modelos de base de datos definidos *models .py* y crea tablas de base de datos basadas en esas clases de modelo. Para hacer esto, primero, configure su base de datos modificando la *databases* configuración en *settings.py*. Luego, una vez que haya definido los modelos de su base de datos, ejecute Python *manage.py makemigrations* seguido de *Python manage.py migrate* para crear o actualizar el proyecto.

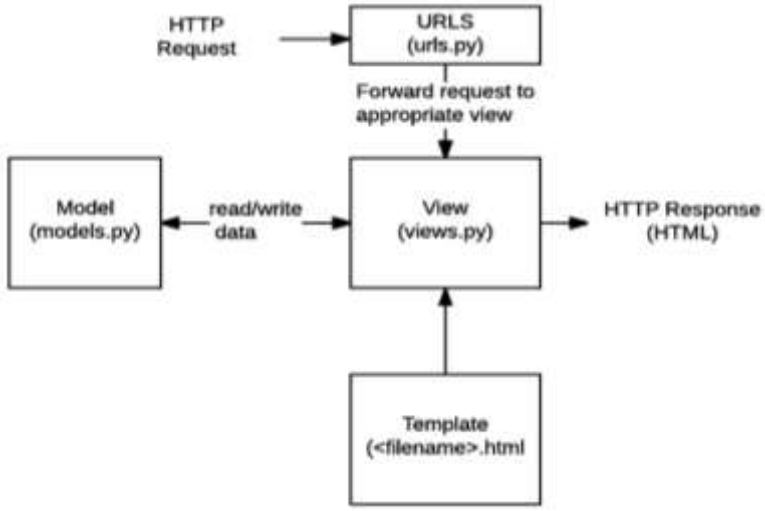

Figura 2.11 Diagrama de aplicación web en Django [30]

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en ficheros separados:

- *URLS:* ((Localizador de Recursos Uniforme, *Uniform Resource Locator*)) Aunque es posible procesar peticiones de cada URL individual vía una función individual, es mucho más sostenible escribir una función de visualización separada para cada recurso. Se usa un mapeador URL para redirigir las peticiones HTTP *(Hypertext Transfer Protocol)* a la vista apropiada basándose en la URL de la petición. El mapeador URL se usa para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la función de visualización como datos. [30]

- *Vista (View):* Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de modelos, y delegan el formateo de la respuesta a las plantillas (*templates*). [30]

- Modelo*s (Models):* Los modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos. Los modelos definen la estructura de los datos

almacenados, incluyendo los tipos de campos y posiblemente también su tamaño máximo, los valores por defecto, la lista de selección de opciones, texto de ayuda para documentación, etiquetas de texto para formularios, etc. [30]

- *Plantillas (Templates):* Una plantilla es un fichero de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real. Una vista puede crear dinámicamente una página usando una plantilla, rellenándola con datos de un modelo, [30].

- *Formularios:* Los formularios HTML se usan para recolectar datos de los usuarios para su procesamiento en el servidor. Django simplifica la creación, validación y procesamiento de los formularios, [30].

- *Autenticación y permisos de los usuarios:* Django incluye un sistema robusto de autenticación y permisos que ha sido construido con la seguridad en mente. [30]

- *Cacheo:* La creación dinámica de contenido es mucho más intensiva computacionalmente (y lenta) que un servicio de contenido estático. Django proporciona un cacheo flexible de forma que se puede almacenar todo o parte de una página renderizada para que no sea re-renderizada nada más que cuando sea necesario, [30].

- *Sitio de Administración*: El sitio de administración de Django está incluido por defecto cuando se crea una app usando el esqueleto básico. Esto hace que sea trivialmente fácil proporcionar una página de administración para que los administradores puedan crear, editar y visualizar cualquiera de los modelos de datos de su sitio, [30].

- *Serialización de datos:* Django hace fácil el serializar y servir tus datos como XML o JSON. Esto puede ser útil cuando se está creando un servicio web (un sitio web que sólo sirve datos para ser consumidos por otras aplicaciones o sitios, y que no presenta en pantalla nada por sí mismo), o cuando se crea un sitio web en el que el código del lado cliente maneja toda la renderización de los datos, [30].

• *Channels de Django: Channels* es un proyecto que toma Django y extiende sus capacidades más allá de HTTP, para manejar WebSockets, protocolos de chat, protocolos del Internet de las cosas (IoT, Int*ernet Of Things*) y más. Está construido sobre una especificación de Python llamada *ASGI*. Se basa en el soporte *ASGI* nativo disponible en Django desde v3.0 y proporciona una implementación propia para Django v2.2. Django todavía maneja HTTP tradicional, mientras que los canales le dan la opción de manejar otras conexiones en un estilo sincrónico o asincrónico, [21].

Django *Channles* permite emplear un servidor de tipo *ASGI* y con esto se puede manejar no solamente las peticiones de tipo HTTP, si no también protocolos que requieren de una comunicación de lado a lado, es decir, de tipo full-duplex de manera persistente abriendo un canal bidireccional para transmitir los mensajes empleando el protocolo llamado TCP (*Transmission Control Protocol*) de tal manera que esta comunicación full-duplex permite conectarnos entre el cliente y servidor de manera persistente, [32].

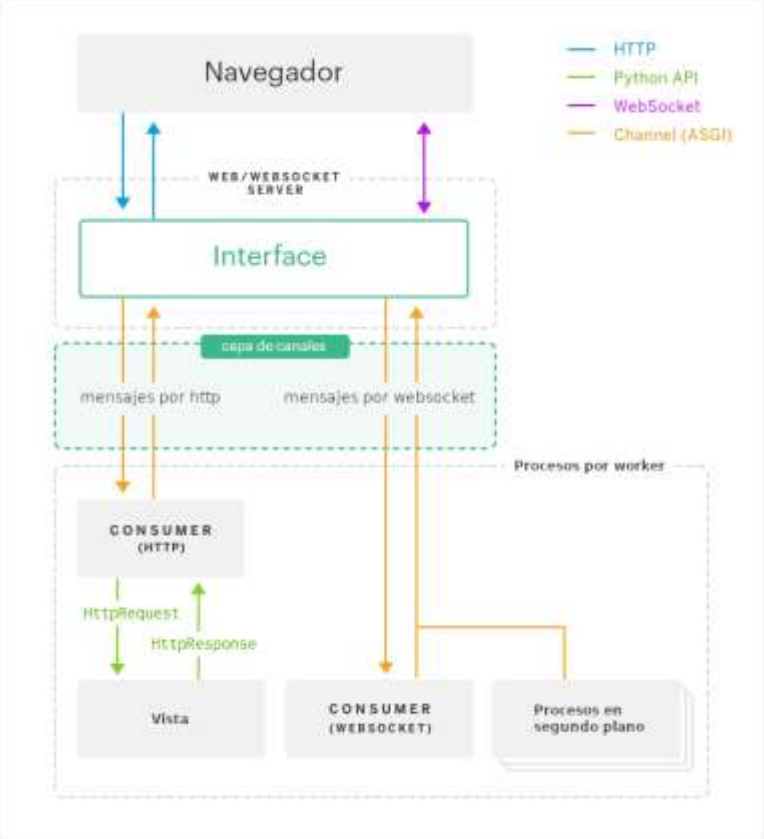

Figura 2.12 Capa de canales [32]

• *WSGI: conexión clásica*. Cuando una petición HTTP es enviada por un cliente mediante el navegador para el servidor web, Django maneja la petición pasando la misma a un objeto de tipo *HttpRequest* y lo manda de acuerdo a la vista según la ruta configurada, la vista procesa la petición y retorna un objeto de tipo *HttpResponse* y lo manda de vuelta al cliente. [32].

• *ASGI:* también conocido como interfaz de puerta de enlace de servidor asíncrono, está diseñado para desvincular las aplicaciones de c*hannels* de un servidor de aplicaciones especifico y proporcionar una forma común de escribir código de aplicaciones y middleware, diseñado tanto para ejecutarse de manera asíncrona como para admitir múltiples protocolos.

El *scope* define las propiedades de una conexión, como su IP remota (para HTML) o nombre de usuario y la duración de una conexión, este contiene detalles sobre la conexión específica *send*, un invocable asincrónico, que permite que la aplicación envíe mensajes de eventos al cliente, y *receive* un invocable asincrónico que permite a la aplicación recibir mensajes de eventos del cliente.

Esto no solo permite múltiples eventos entrantes y eventos salientes para cada aplicación, sino que también permite una corrutina (o micro hilos, otra forma de multitareas en Python) en segundo plano para que la aplicación pueda hacer otras cosas, [33].

• *WebSocket:* WebSocket es un protocolo de red basado en TCP que establece cómo deben intercambiarse datos entre redes. Puesto que es un protocolo fiable y eficiente, es utilizado por prácticamente todos los clientes. El protocolo TCP establece conexiones entre dos puntos finales de comunicación, llamados *Socket*s. De esta manera, el intercambio de datos puede producirse en las dos direcciones. [34]

La tecnología WebSocket permite abrir un canal de comunicación bidireccional entre el cliente y el servidor. El cliente puede enviar mensajes y recibir respuestas controladas por eventos sin tener que consultar al servidor. La ventaja de este intercambio es que se accede de forma más rápida a los datos. En otras palabras: la web que se solicita se muestra en tiempo real.

• *Consumers:* Un consumidor es la unidad básica del código de canales el cual consume eventos. Cuando ingresa una solicitud o un nuevo so*cket*, c*hannels* seguirá su tabla de

enrutamiento, encontrará el consumidor adecuado para esa conexión entrante y comenzará una copia de la misma. A diferencia de las vistas de Django, los consumidores son de larga duración, pero también pueden serlo de corta duración, ya que pueden atender solicitudes HTTP [35]

Para emplear el esquema de canales en la aplicación, se deben crear los *consumers*, estos *consumers* son los que van a manejar las peticiones de tipo asíncronos mediante el servidor *ASGI* específicamente la conexión mediante WebSockets.

Los consumidores hacen un par de cosas en particular:

1. Estructura su código como una serie de funciones que se llamarán cada vez que ocurre un evento, en lugar de hacer que escriba un bucle de eventos.
2. Le permite escribir código sincrónico o asincrónico y se ocupa de las transferencias y subprocesos.

Luego de crear el *consumers,* se debe crear el ruteo o enrutamiento correspondiente al mismo. [32]

• *Routing: channels* proporciona clases de enrutamiento (*Routing*) que le permiten combinar y apilar a sus consumidores para enviar en función de cuál es la conexión. Los enrutadores son en sí mismo aplicaciones *ASGI* válidas y es posible anidarlos. [35]

Los enrutadores de canales solo funcionan en el nivel de alcance, no en el nivel de eventos individuales, lo que significa que solo puede tener un consumidor para una conexión determinada. El enrutamiento consiste en determinar qué consumidor individual proporcionar una conexión, no cómo difundir eventos de una conexión entre varios consumidores [35].

2.2.22 Flask

Flask es un marco web, es un módulo de Python que te permite desarrollar aplicaciones web fácilmente. Tiene un núcleo pequeño y fácil de ampliar: Es un micro*framework* que no incluye un ORM (*Object-Relational Mapping,* o mapeo objeto-relacional) o características similares. Tiene muchas características interesantes como enrutamiento de URL, motor de plantillas. Es un marco de aplicación web *WSGI.*

Flask es un marco de aplicación web escrito en Python. Fue desarrollado por Armin Ronacher, quien dirigió un equipo de entusiastas internacionales de Python llamado Poocco. Flask se basa en el kit de herramientas *Werkzeg WSGI* y el motor de plantillas Jinja2, ambos proyectos de Pocco. [36]

- *WSGI:* La Interfaz de puerta de enlace del servidor web (Web Server Gateway Interface, *WSGI*) se ha utilizado como estándar para el desarrollo de aplicaciones web Python. *WSGI* es la especificación de una interfaz común entre servidores web y aplicaciones web.
- *Werkzeug:* es un conjunto de herramientas *WSGI* que implementa solicitudes, objetos de respuesta y funciones de utilidad. Esto permite construir un marco web sobre él. El marco Flask utiliza *Werkzeug* como una de sus bases.
- *Jinja2:* es un motor de plantillas popular para Python. Un sistema de plantillas web combina una plantilla con una fuente de datos específica para representar una página web dinámica.
- *Microframework:* Está diseñado para mantener el núcleo de la aplicación simple y escalable. En lugar de una capa de abstracción para el soporte de la base de datos, Flask admite extensiones para agregar tales capacidades a la aplicación.

2.2.23 Redis

Redis es el acrónico de Servidor de Diccionario Remoto *(Remote Dictionary Server)*, es un servidor de estructura de datos que implementa una gran cantidad de comandos y operaciones que puede realizar, proporcionando capacidades de canalización para ejecutar instrucciones por lotes. El servidor es compatible con varios tipos de valores, e incluye estructuras de datos más complejas que un almacén de clave-valores estándar.

Tiene licencia BSD, está escrito en código C optimizado y admite numerosos lenguajes de desarrollo, [37].

Puede ser usado para agregar un valor a una clave, aumentar el valor de un campo hash, guardar el conjunto de datos en un disco local, realizar operaciones bit a bit, asociar nuevas ranuras hash al nodo receptor, conectar clústeres de nodos, obtener información de depuración y mucho más. Las claves que almacena incluyen cadenas y conjuntos junto y listas.

El servidor puede conectarse a una variedad de clientes, admitiendo múltiples lenguajes de programación, incluidos C, C #, C ++, Erlang, Java, Lua, Matlab, Node.js, Perl, PHP, Python, Ruby, Scala, Scheme, Tcl, VCL.

Redis para Windows ofrece a los usuarios un equivalente con todas las funciones diseñado específicamente para funcionar en el sistema operativo de Microsoft. Su lista de comandos es tan rica como la del paquete Posix, lo que permite a los usuarios realizar una amplia variedad de operaciones.

2.2.24 Editor de Código

Para empezar a programar en los distintos lenguajes de programación, es necesario instalar un segundo programa en el equipo. Se hace referencia a un editor de código fuente, también conocido como Entorno de Desarrollo Integrado (IDE, *Integrated Development Environment*). Se trata de una herramienta diseñada para editar el código fuente de diversos lenguajes de programación como Python. Existen muchos editores que pueden utilizarse en todas las plataformas, en este caso se usará Visual Studio Code.

- Visual Studio Code

Es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

Visual Studio Code es un editor de código fuente compatible con varios lenguajes de programación y un conjunto de características que pueden o no estar disponibles para un lenguaje dado. [38]

CAPÍTULO III
HERRAMIENTA DE PROGRAMACIÓN

En el presente capítulo, se analizan y selecciona tanto la base de datos a utilizar como el lenguaje de programación y *framework* de desarrollo más adecuado, para el desarrollo del sistema SCADA de la Universidad de los Andes.

3.1 SELECCIÓN DE BASE DE DATOS

Existe diferentes tipos de bases de datos, habiendo una mayor diferenciación entre las relacionales y no relacionales, estas se organizan de forma diferente y trabajan con tipo de datos distintos, en caso de base de datos relacional se trabaja con el estándar SQL el cual es usado para actualizar y recuperar datos.

Para seleccionar entre una base de datos SQL o NoSQL se debe tener presente la relación entre los diferentes tipos de datos que se van a almacenar, si los datos no tienen relación y pueden vivir separados, la mejor opción es una base de datos NoSQL o no relacional, pero si en necesario mantener una relación entre la información almacenada, es recomendable una base de datos SQL o relacional.

Para efectos del presente trabajo, es necesario mantener una relación entre la información almacenada, por lo cual, se selecciona una base de datos SQL. Entre los distintos sistemas de gestión de bases de datos (SGBD) para una base de datos relacional se cuenta con las más grandes compañías como lo son IBM, Microsoft y Oracle las cuales son bases de datos propietario, mientras que de las bases de datos libres destacan MySQL, MariaDB, PostgreSQL y SQLite3.

Se realiza una tabla comparativa entre las bases de datos MySQL, SQL Server y PostgreSQL con el fin de seleccionar la base de datos más adecuada para el desarrollo del presente trabajo de grado.

Tabla 3.1 Cuadro comparativo de los SDBG

	MySQL	SQL Server	PostgreSQL
Empresa	*Sun Microsystem*	Microsoft	PostgreSQL *Global Development Group*
Características	- Está optimizado para equipos de múltiples procesadores. - Se puede utilizar como cliente-servidor o incrustado en aplicaciones. - Soporta múltiples métodos de almacenamiento de tablas, con prestaciones y rendimiento diferentes para poder optimizar el SGBD a cada caso concreto	- Soporte de transacciones. -Soporta procedimientos almacenados. - Entorno gráfico de administración, que permite el uso de comandos DDl y DML gráficamente. - Permite trabajar en modo cliente-servidor,	- Texto de largo ilimitado. -Integridad referencial. - Llaves primarias y foráneas. - Replicación asíncrona/síncrona. - Copias de seguridad Múltiples métodos de autenticación.
Idioma	C, C++	C, C++	C
Licencia	Libre a nivel de usuario, pero para las empresas que quieran incorporarlo en productos privados deben comprar la licencia	Privada	Libre
Lenguaje de Manipulación de Datos	SQL	SQL	SQL
Requisitos de Hardware y Software	- RAM 512 MB - Espacio disco duro 1 GB. - Sistema Operativo: Windows *Server*, Windows seven, Linux, Unix. - Arquitectura del Sistema 32/64 bit. Protocolo de Red TCP/IP.	-.NET 3.5 SPI -.NET *Framework* 4.1 -SQL *Server Native Client* -Protocolos de Red: Memoria compartida Canalizaciones con TCP/IP Memoria:512 MB/ 1 GB/ 4GB Procesador x86: 1.0 GB	- SO. Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS, Solaris, Tru64) y Windows. Espacio Disponible en disco: 70 MB (mínimo) Tamaño de Fichero: 48.45 MB (50,802,536 bytes)

Una de las primeras características a tomar en cuenta a la hora de seleccionar una base de datos es que esta sea de licencia libre, como se puede observar en la anterior tabla SQL Server es de licencia privada, mientras que MySQL es libre a nivel de usuario, pero paso a ser propiedad de Oracle en el año 2010, el cual es de licencia privada, este tiene licencia GPL la cual obliga a incluir el código fuente en su distribución por lo cual es imposible cambiar la licencia al programa, por otro lado, se tiene que PostgreSQL es totalmente libre y se puede modificar y distribuir de forma gratuita tanto para fines académicos como privados y comercial. Este cuenta con una licencia BSD en software libre y no es *copyleft* como la GPL por lo cual permite cambiar la licencia.

Una encuesta en de 2021 en Stack Overflow reveló que entre las cinco bases de datos preferidas por los desarrolladores destacan MySQL, PostgreSQL, SQLite, MongoDB y Microsoft SQL Server como se muestra en la siguiente figura.

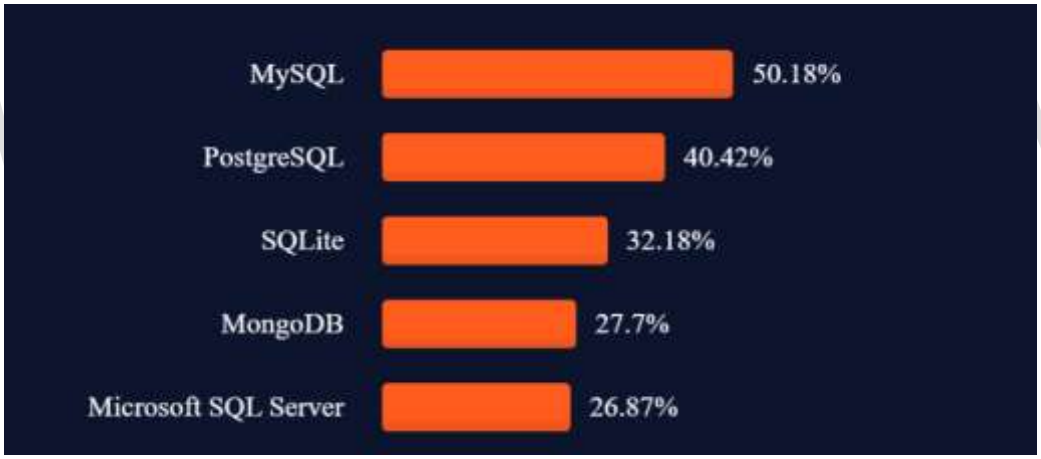

Figura 3.1 Preferencia de bases de datos en 2021 [39]

Luego de estudiar y analizar varias opciones para la base de datos se optó por seleccionar el motor de base de datos relacional PostgreSQL, el cual se adapta a necesidades y además es de licencia libre, también es gratuito, código abierto, altamente extensible y se ajusta al estándar SQL.

3.2 SELECCIÓN DE LENGUAJE DE PROGRAMACIÓN

Existen diversos lenguajes de programación de *backend* altamente competitivos, el presente trabajo se centra en los lenguajes de programación basados en código abierto, orientado a

objetos y que permitan la creación de páginas web. Se estudiaron los lenguajes como lo son Python, Node.js, C++, PHP y C#.NET, estos se comparan teniendo en cuenta sus características y limitaciones en la siguiente tabla.

Tabla 3.2 Cuadro comparativo para *backend*

LENGUAJE DE PROGRAMACIÓN	CARACTERÍSTICAS	LIMITACIONES
Python	- Fácil aprendizaje - Alta gama de bibliotecas - Rentable y de código abierto - Código intangible - Comunidad activa y amigable - Lenguaje orientado a objeto - Mayor seguridad. - Implementación web mediante *framework*.	- Depende en gran medida de bibliotecas y trabajos de terceros - Las variabilidades de las versiones pueden ocasionar errores en las versiones - Menor velocidad con respecto a otros programas
Node.js	- Fácil aprendizaje - Tiene incorporado JavaScript - Módulos de librerías - Orientado a objeto - Comunidad cada vez mayor - Menor tiempo de ejecución -	- Lenguaje estructurado diferente a otros. - No admite programación de subprocesos múltiples. - Hasta los momentos, es poco conocido.
C++	- Lenguaje robusto - Orientado a objeto - Más rápido que la mayoría de los lenguajes de programación	- Curva de aprendizaje alta - No es atractivo visualmente - No soporta creaciones de páginas web
PHP	- Multipropósito y fácil uso - Orientado al desarrollo de aplicaciones web - Versátil, código abierto y económico - Seguridad incorporada	- No puede competir con tecnologías de desarrollo modernas. - La influencia PHP como tecnología está disminuyendo
C#.NET	- Amplia compatibilidad con versiones anteriores - Desarrollo multiplataforma - Orientado a objetos	- No es posible comunicarse con el hardware - Es muy rígido, solo se ejecuta en el marco .Net usando Windows.

Según la encuesta de Stack Overflow en 2021 sobre los lenguajes de desarrollo más usados se encontró que JavaScript encabeza la lista por noveno año consecutivo, seguido de HTML/CSS, Python, SQL, Java y Node.js como se puede apreciar en la figura 3.2.

Una vez estudiados y analizados los distintos lenguajes de programación para el desarrollo de la aplicación SCADA-ULA, se optó por el lenguaje de programación Python el cual se adapta a las necesidades del sistema. Este es conocido por su fácil lectura de código, acceso a bibliotecas bien documentadas, su gran comunidad de usuarios y código repetible. Además, se cuenta con conocimiento previo en dicho lenguaje, lo cual facilita el desarrollo del sistema.

Desde que Python fue creado en 1991 por Guido Van Rossum, ha crecido hasta convertirse en uno de los lenguajes de programación multipropósito líderes en el mundo actual.

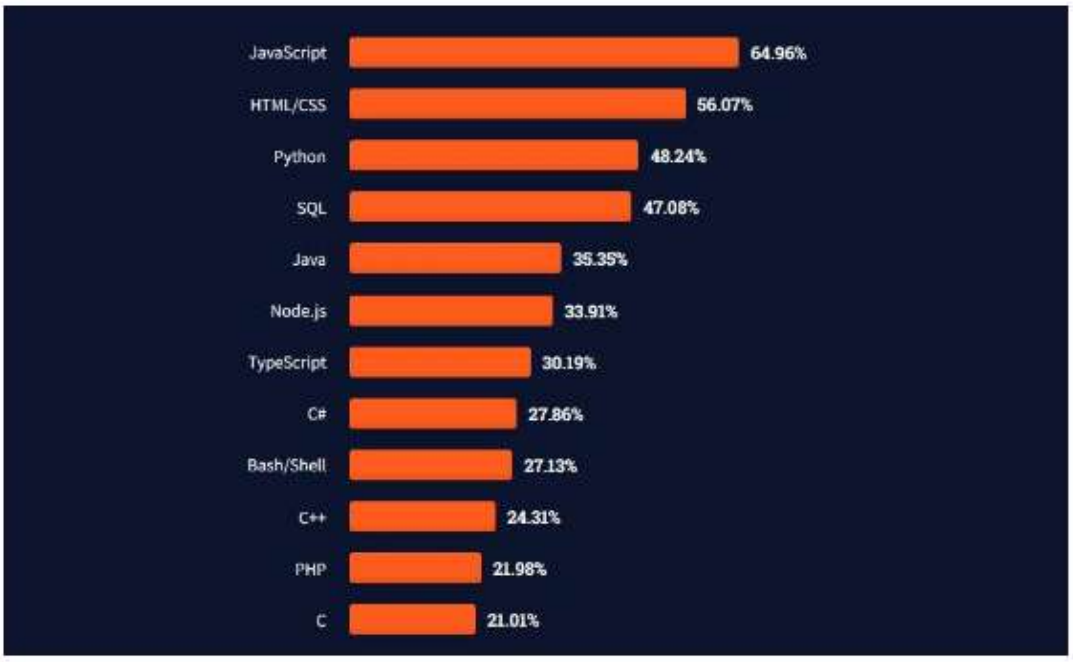

Figura 3.2 Lenguajes de programación más usados en 2021 [39]

La encuesta de Stack Overflow reveló que Python es el mejor lenguaje de programación de *backend* que los desarrolladores esperan aprender. Python ha encabezado esa categoría durante quinto años consecutivos. También clasificó tercero entre los lenguajes de desarrollo más queridos en existencia.

3.2 SELECCIÓN DE *FRAMEWORK*

Python cuenta con varios *framework* muy poderosos a la hora de crear aplicaciones, para efectos del presente trabajo, se analizan los más usados en la actualidad como lo son Django y Flaks, estos son comparados en la siguiente tabla.

Tabla 3.3 Cuadro comparativo de *framework*

DJANGO	FLASK
- Cuenta con sistema de autenticación de usuario - Trabaja bajo un patrón MVC que permite un desarrollo ágil y reutilizable. - Las API´s son mejores y se pueden convertir fácilmente en páginas HTML - Realiza solicitudes GET y POST fácilmente - Gran comunidad activa - Documentación de acceso gratuito - Utiliza ORM para asignar objetos a tablas de bases de datos - Ofrece formularios *Model*-based	- Usa opciones nativas para crear sistemas de autenticación - Condicionado para solicitudes GET y POST. - Mayor velocidad con respecto a Django - Tiene extensiones para agregar ORM, validación de formularios o manejo de cargas.

Para el desarrollo del sistema SCADA-ULA se seleccionó el *framework* de desarrollo web Django, el cual es, con diferencia el mayor *framework* web basado en Python, se apoya en una comunidad grade y activa la cual ayudan a aclarar las distintas dudas, su facilidad de convertir páginas web mediante HTML, trabaja bajo un patrón MVC y tiene un panel de administración para gestionar bases de datos.

CAPÍTULO IV
DESARROLLO DE MÓDULOS

En el presente capítulo se detallará la implementación de los módulos base de datos, gráficos de tendencia, alarmas y eventos del sistema SCADA, así como también las librerías utilizadas para el desarrollo de dichos módulos.

4.1 MÓDULO BASE DE DATOS

Para el desarrollo del módulo de base de datos se procedió a instalar PostgreSQL para Windows 7, una vez descargado y se procede a instalar y configurar definiendo la contraseña de administrador del servidor, se mantuvo el puerto TCP por defecto el cual es 5432.

Para la creación de la base de datos se usa la interfaz web para administrar PostgreSQL llamada *PGADMIN4*, se crea una base de donde se almacenarán todos los datos proporcionados por Django.

La conexión entre la base de datos y Python se realiza mediante la librería psycopg2.

- **Librería Psycopg2**

  Según [28], se define *psycopg* como el adaptador de base de datos PostgreSQL más popular para el lenguaje de programación Python. Sus principales características son la implementación completa de la especificación Python db api 2.0 y la seguridad de subprocesos (varios subprocesos pueden compartir la misma conexión).

  Fue diseñado para aplicaciones en gran medida multi-hilo que crean y destruyen gran cantidad de cursores y hacen que un gran número de concurrentes *insert* o *update*.

  Psycopg2 cuenta con cursores del lado del cliente y del lado del servidor, comunicaciones y notificaciones asincrónicas, soporte para copia. Muchos tipos de Python son compatibles desde el primer momento y se adaptan a los tipos de datos de PostgreSQL coincidentes; La

adaptación se puede ampliar y personalizar gracias a un sistema de adaptación de objetos flexible. Es compatible con Python 3.

Su instalación se realiza mediante el comando *pip install psycopg2*, este tiene la función de convertir las variables de Python en valores SQL

4.1.1 Modelo Entidad-Relación

Se define el diseño de modelo entidad-relación donde se representan de manera simplificada las tablas de datos de cada uno de los procesos a desarrollar.

Las variables del proceso 1, serán todas digital, por lo que no es necesario ingresar datos como el *SetPoint* y los límites para la generación de alarmas, dichos campos, son requeridos en los procesos 2 y 3 donde se incluyen variables análogas como lo son voltaje y temperatura.

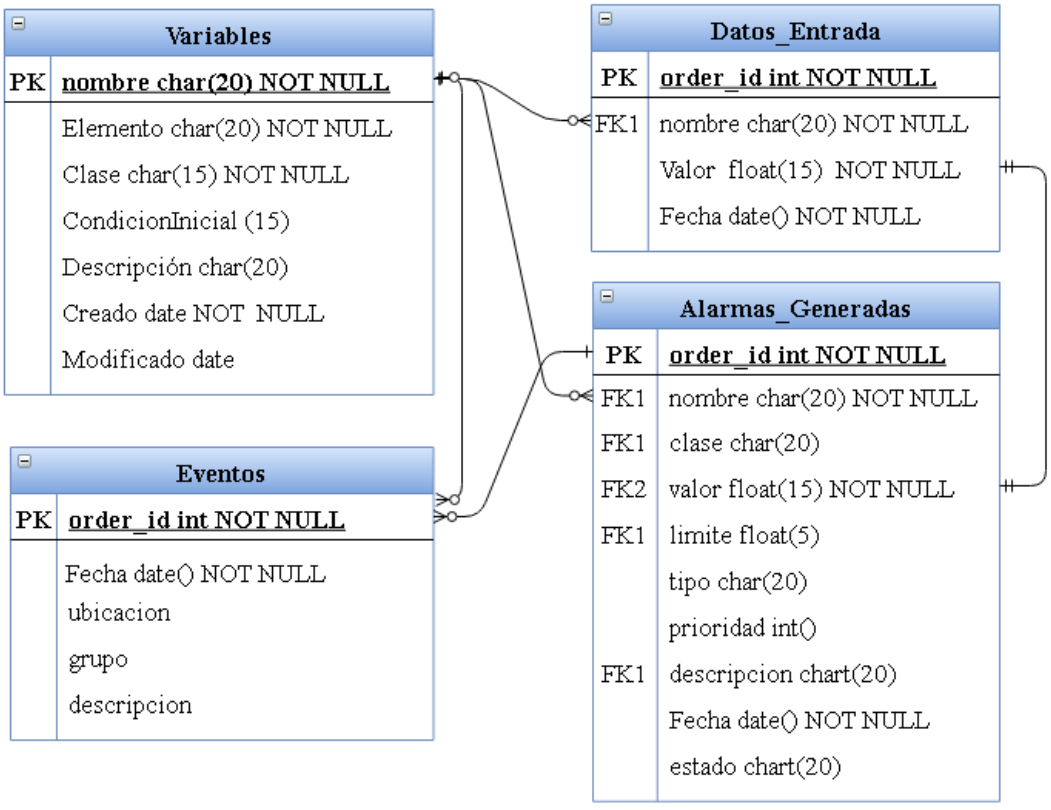

Figura 4.1 Diagrama entidad-relación perteneciente al proceso 1. Fuente: autor

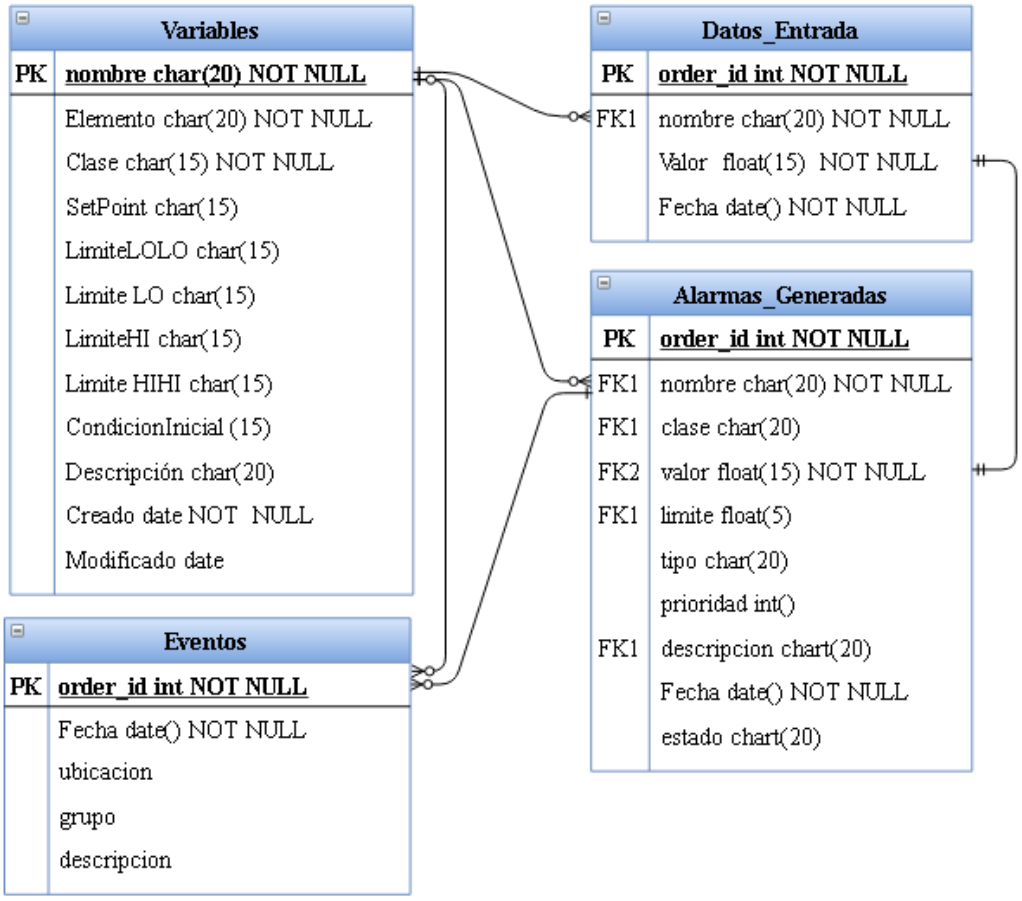

Figura 4.2 Diagrama entidad-relación perteneciente a los procesos 2 y 3. Fuente: autor

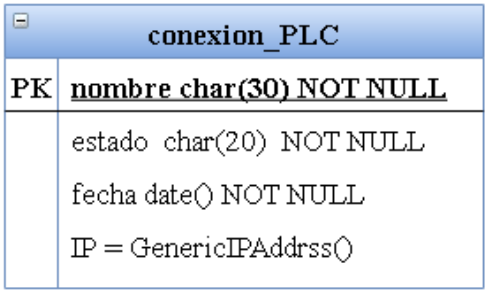

Figura 4.3 Diagrama entidad-relación de estado del PLC. Fuente: autor

Dentro del archivo *models .py* de cada aplicación se define las clases a través de código Python, las cuales son transformadas a código SQL para crear las tablas, relaciones e índices de la base de datos.

Se puede decir que la tabla *VariableProceso,* es la tabla principal de cada uno de los procesos, ya que las otras tablas dependen de ellas. En cada proceso, cada vez que se crea, modifica o elimina una variable se crea un registro en la tabla *Eventos* donde se indica que usuario a realizado dicha operación y que variable fue modificada. Para cada variable creada se almacenan los datos de entrada en la tabla *Datos_Entrada,* estos se verifican con el fin de comprobar que estén dentro de los límites establecidos, si se superan los límites, se rellena automáticamente los campos de la tabla *Alarma_Generada* para cada variable cuyos límites fueron superados, cada alarma tiene la opción de ser reconocida por un operador, al reconocer dicha alarma se genera un *evento* con el nombre del operador y el nombre de la alarma reconocida.

Los campos de la tabla *VariableProceso* están definidos como:

- Nombre: el nombre que se le asigna a cada variable para identificarla, dicho nombre es único para cada proceso.
- Elemento: se especifica a que elemento pertenece la variable, cada proceso contiene distintos elementos en los cueles se pueden encontrar elemento de voltaje, temperatura, interruptores, entre otros.
- Clase: indica si la variable es análoga o digital, dicho campo se crea automáticamente dependiendo del elemento al cual pertenece la variable.
- *SetPoint*: es el nivel óptimo en el cual se quiere que esté la medición de cada variable análoga, dicho campo no se toma en cuenta para las variables digitales como lo son los interruptores.
- Límites: los valores que deben ser superados para generar una alarma en las variables análogas, dichos límites definen si la alarma es moderada (si se superan los límites HI y LO) o es crítica (si supera los límites HIHI y LOLO).
- Condición inicial: este campo se toma en cuenta solo para las variables digitales, el cual indicará que estado tendrá la variable una vez iniciado el proceso.
- Descripción: permite agregar una pequeña descripción para mayor información respecto a la variable.
- Creado: fecha y hora en la que se crea la variable.

- Modificado: fecha y hora en la cual se realiza una modificación a la variable.

La tabla *Datos_Entrada* contiene los siguientes campos:

- Nombre: el nombre de la variable a la cual se le realiza la medición.
- Valor: el valor medido para dicha variable
- Fecha: fecha y hora en la que se efectúa la medición.

Es la tabla de *Alarmas_Generadas* se puede encontrar:

- Nombre: nombre de la variable que superó sus límites.
- Clase: de la tabla *Variable* se toma la clase a la cual pertenece dicha variable.
- Valor: es el valor que se obtuvo en la medición.
- Límite: el valor límite que superó la variable,
- Tipo: el tipo de alarma generada, está puede ser HI, HIHI, LO, LOLO o DSC.
- Prioridad: define la prioridad de la alarma, la cual puede ser "1" para alarmas críticas, "2" para alarmas moderadas "3" para alarmas digitales.
- Descripción: se toma de la tabla *Variable*
- Estado: indica si la alarma generada ha sido reconocida por el operador o no.
- Fecha: indica la fecha y hora en la cual ocurre la alarma.

La tabla *eventos* es única y almacena información de los tres procesos. Esta contiene los siguientes campos:

- Fecha: indica la fecha y hora en la cual ocurre el evento.
- Ubicación: se almacena el proceso al cual pertenece dicho evento
- Grupo: indica si el evento es por creación, modificación o eliminación de variable, por reconocimiento de Alarma.
- Descripción: contiene una especificación del usuario que generó dicho evento y que variable está involucrada.

Tanto la tabla *Datos_Entrada* como la de *Alarmas_Generadas* y *eventos* crean un ID automáticamente el cual es usado como clave primaria.

Se crea una tabla llamada *conexión_PLC*, la cual tendrá como función almacenar el estado de conexión o desconexión del PLC, esta tendrá 3 campos (PLC1, PLC2 o PLC3) creados

automáticamente una vez iniciado el sistema por primera vez, dicha tabla contiene los siguientes campos:

- Nombre: nombre del PLC según el proceso
- Estado: indica si el PLC está conectado o desconectado.
- Fecha: indica la fecha en la que se realiza el cambio de estado
- IP: se crea con el fin de ingresar la dirección IP que se conectará con el PLC.

La función de la tabla *conexión_PLC* es crear condicionales para conectar el PLC con el HMI, solo podrá estar un PLC activo el cual depende del proceso en el que se encuentre, si se conecta del proceso 1 y luego va a conectarse desde el proceso 2 se desconectará automáticamente el PLC del proceso 1.

4.2 CREACIÓN DE APLICACIÓN WEB EN DJANGO

Una vez instalado Django, desde la ventana de comando de Visual Studio Code, se ingresa el comando de la siguiente figura, dicha línea de código permite crear un nuevo proyecto.


```
django-admin startproject ScadaUla
```

Figura 4.4 Creación de nuevo proyecto en Django. Fuente: autor

Al ejecutar dicho comando crea un directorio automáticamente el cual contiene los siguientes archivos

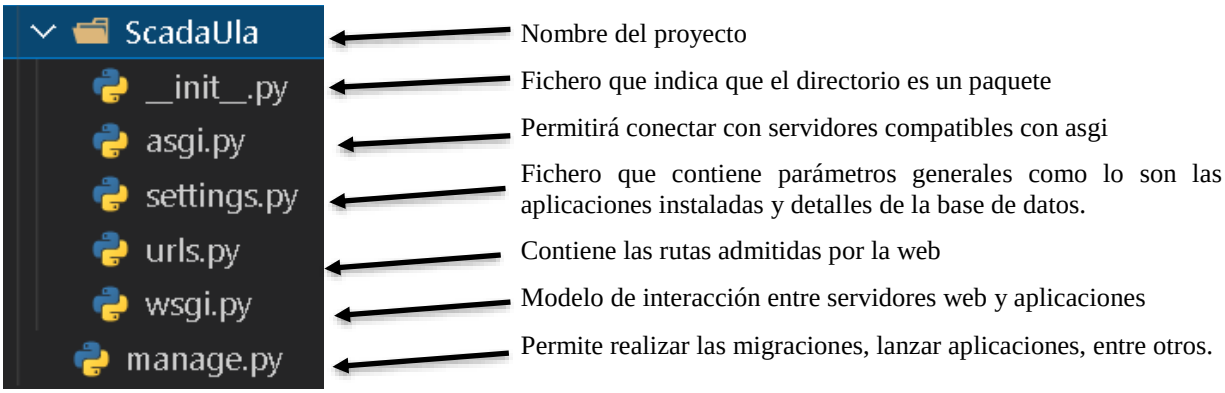

Figura 4.5 Archivos principales de proyecto en Django. Fuente: autor

Para crear una aplicación, se debe ingresar al directorio donde se encuentra el *magage.py* del proyecto creado anteriormente, se ejecuta el código de la figura 4.6, esté crea automáticamente otra carpeta que contiene la estructura descrita en la figura 4.7.

```
Desktop\scada2.0\1\scadaula> Python manage.py startapp AppPrincipal
```

Figura 4.6 Creación de nueva aplicación en django

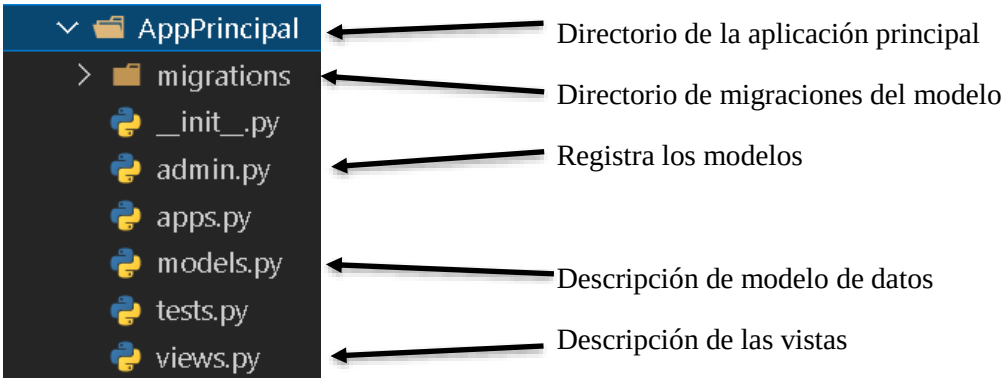

Figura 4.7 Archivos de aplicaciones en proyecto de Django. Fuente: autor

El fichero *settings.py* es único para cada proyecto y contiene todos los ajustes del sitio. Dentro de dicho fichero se deben registrar las aplicaciones de los proyectos dentro del apartado *INSTALLED_APPS*, así como también se debe agregar "*channels"* para indicarle a Django que se usaran canales.


```
INSTALLED_APPS = [
    'channels',
    'admin_interface',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'colorfield',
    'crispy_forms',
    'App0',
    'App1',
    'App2',
    'App3',
]
```

Figura 4.8 Aplicaciones instaladas en settings.py. Fuente: autor

Para iniciar el servidor se debe ejecutar el código de la figura 4.9, este permite mostrar el servidor web al cual se ingresa desde el navegador mediante la dirección *http://127.0.0.1:8000*

en la página de inicio de Django. Una vez en dicha página se confirma que se instaló y configuró la aplicación correctamente.

```
\scadaula> Python manage.py runserver
```

Figura 4.9 Iniciar el servidor. Fuente: autor

Python trae por defecto una interfaz de administración automática la cual permite interactuar con los datos almacenados en la base de datos, dicha interfaz se limita a la herramienta de gestión interna de una organización, más no está diseñado para construir todo su *frontend*.

Para iniciar sesión el usuario debe crear un superusuario el cual se realiza desde la consola mediante el comando descrito en la figura 4.10, dicho usuario el atributo *is_staff* establecido en *True***,** el cual indica que el usuario puede acceder al sitio como administrador y tener acceso a toda la plataforma sin asignarlo explícitamente.

```
\scadaula> Python manage.py createsuperuser
```

Figura 4.10 Creación de superusuario en Django. Fuente: autor

4.2.1 Models

En Django, *Models* contiene la información de las tablas de la base de datos, los atributos de ese modelo se convierten en columnas de esa tabla, esos atributos reciben el nombre de Django fields, estos convierten automáticamente la información a almacenar para que sea compatible con la base de datos a utilizar.

El ORM permite usar los datos persistentes en la base de datos como objetos de distintos tipos en Python, por ende, se pueden realizar cambios en el modelo como crear, leer, modificar o eliminar campos en la base de datos elegida. A la operación de sincronizar del modelo Python con el modelo físico de la base de datos se le da el nombre de migración.

Django trae incluido por defecto el gestor de base de datos SQLite3, pero debido a que en este proyecto se usará el motor de base de datos PostgreSQL se debe modificar la definición del archivo *settings.py* el apartado *DATABASE* el cual se sustituye por el código presente en siguiente figura.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'db_scada',
        'USER': 'postgres',
        'PASSWORD': 'password',
        'HOST': '127.0.0.1',
        'DATABASE_PORT': '5432',
    }
}
```

Figura 4.11 Configuración de PostgreSQL como gestor de base de datos. Fuente: autor

En el archivo *models .py* se crean las clases donde se almacenarán los datos correspondientes a cada tabla definida en el modelo entidad-relación para cada uno de los procesos a desarrollar los cuales tienen distintas variables a controlar.

En el ORM de Django cada tabla de la base de datos es representado por una clase Python y cada columna es representada por cada uno de los campos pertenecientes a la clase.

```
class VariableProceso1(models.Model):
    nombre=models.CharField(max_length=50, primary_key=True)
    elemento=models.CharField(max_length=20, choices=elem)
    clase=models.CharField(max_length=20, null=False)
    CondicionInical=models.CharField(max_length=15, null=True, blank=True, choices=dsc,
    descripcion=models.TextField(max_length=20, blank=True)
    creado= models.DateTimeField(auto_now_add=True)
    modificado= models.DateTimeField(auto_now_add=True)
```

Figura 4.12 Creación de la clase variables del proceso 1 en *models .py*. Fuente: autor

Las clases heredan de *Model* (Una clase provista por el ORM de Django) que lo convierte en un objeto mapeable para el ORM el cual tiene que cumplir una serie de características en los atributos de dicha clase.

El sistema de migración sincroniza Django *Models* con las tablas en SQL mediante la orden de la figura 4.13. este traduce el código de las clases del modelo en sentencias SQL.

```
scadaula> Python manage.py makemigrations
```

Figura 4.13 Migración de Django al lenguaje SQL. Fuente: autor

Makemigrations crea un archivo dentro de *migrations* que contiene una clase Python con el modelo. Para que este modelo se cree en la base de datos se debe ejecutar el comando de la

figura 4.14, dicho comando permite crear las tablas de las clases definidas en los modelos en la base de datos. Se debe tener en cuenta que es necesario realizar una migración cada vez que se realice un cambio en la base de datos.

```
\scadaula> Python manage.py migrate
```

Figura 4.14 Creación de tablas en la base de datos. Fuente: autor

Lo primero que se realiza en un proyecto Django es la creación de un mapeador url, una vista y una plantilla dentro de cada proyecto

4.2.2 Views

Las vistas (*views*) en Django, son un controlador que recogen las peticiones HTTP de una aplicación web, dichas peticiones pueden ser GET o POST. Las vistas son un enlace entre el *Models* y el *template*.

En las vistas se encuentra toda la lógica necesaria para devolver una respuesta. El archivo *views.py* importa el atajo de la función *render* que genera archivos HTML usando plantillas y datos.

A través del archivo *views*, es solicitada la información a los modelos, posteriormente son renderizados invocando la función *render* para crear o retornar una página HTML como respuesta, también se usan funciones como *redirect* para redirección a una página distinta una vez que se haya cumplido una función en específico.

Ejemplo, en el caso del inicio de sección, se verifica si el usuario está registrado mediante el método de autenticación, si lo está ingresa al sistema, sino lo retorna de nuevo a la página de ingreso

```python
from django.shortcuts import render, redirect

def Home(request):
    if request.user.is_authenticated:
        return render(request, "AppPrincipal/home.html")
    else:
        return redirect('login')
```

Figura 4.15 Archivo *views* perteneciente a la aplicación principal. Fuente: autor

4.2.3 Urls

Se crea un archivo *urls.py* el cual es enlazado mediante el método *include()* desde el urls principal ubicado a la altura del *settings.py* en el nuevo archivo creado dentro de la aplicación se encuentra el patrón url y las funciones de vista que serán llamadas desde el archivo *views.py*, cada una de las rutas está asociada a una vista. Es recomendable especificar el parámetro *name* en la url para identificar cada url de manera particular.

En la siguiente figura 4.16 se muestra la configuración del url perteneciente a la aplicación principal el cual permite ejecutar el HTML que contiene la interfaz de ingreso sistema.

```
from django.urls import path
from django.conf import settings
from . import views

urlpatterns = [
    path('', views.Home, name="home"),
]
```

Figura 4.16 Archivo url perteneciente a la aplicación principal. Fuente: autor

4.2.4 *Templates*

Estos deciden como se mostrará la información. En una aplicación web, el usuario interactúa con la página produciendo una petición HTML que es recogida por las vistas. Mediante las vistas se envía y recibe información de los modelos a través de formularios para crear, modificar o eliminar campos una vez validados o leer campos existentes en las tablas de dicho modelo, la información de los formularios se obtiene mediante el método GET o el método POST, esta es enviada a través de navegador.

En las plantillas se determina la estructura de un archivo HTML, estas generan la salida que irá al navegador. Django busca automáticamente las plantillas en el directorio llamado *template* de su aplicación. En el presente trabajo se genera una carpeta *template* para cada una de las aplicaciones, la cual contiene todos los archivos HTML necesarios para cada proceso. Al igual que una carpeta *static* en la cual se encuentran los archivos CSS y js que permiten darle estilo y características interactivas a la aplicación.

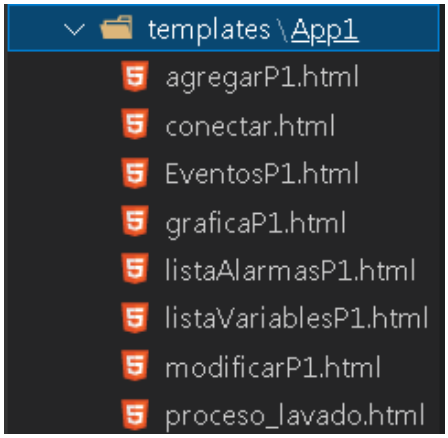

Figura 4.17 Arcchivos *Template* pertenecientes al Proceso 1. Fuente: autor

Para usar las plantillas se usas componentes visuales externos a Django como lo son el CSS y el *framework* bootstrap4 usado para aplicaciones *frontend*. En el presente trabajo se crea una plantilla base la cual tendrá la cabecera, el cuerpo y pie de página fijo, junto con secciones para navegar dejando la estructura de navegación igual para las distintas páginas a usar, en el cuerpo de la plantilla se establecen los bloques de contenidos los cuales se usan para representar las páginas derivadas de las plantillas.

4.2.5 *Message*

Se hace uso del marcador de mensajes *message* que viene integrado por defecto en Django y está basado en cookies y sesiones. Dicho marco de mensaje permite almacenar mensajes temporales en una solicitud y recuperarlos para mostrarlos en la siguiente solicitud. Cada mensaje está determinado por un nivel específico que determina su prioridad, (por ejemplo, *info, warning*, *error*, *questions*, entre otros).

Los mensajes se implementan a través de la clase *middleware* y vienen incluidos en las aplicaciones instaladas dentro del archivo *settings.py*.

Se hace uso de la librería *SweetAlert2* la cual es una librería Javascript que permite crear ventanas emergentes con un diseño profesional y de fácil implementación, la cual es compatible con la mayoría de los navegadores.

Para almacenar los mensajes temporales se debe agregar dentro del archivo *settings.py* el código de la figura 4.18, este se recomienda agregar en las primeras líneas del archivo.

```
MESSAGE_STORAGE="django.contrib.messages.storage.cookie.CookieStorage"
```

Figura 4.18 Activación de la librería SweetAlert2 en archivo settings.py. Fuente: autor

4.2.6 Forms

Los formularios forman una parte esencial en esta aplicación, ya que permite a los usuarios agregar información. En el presente trabajo se usan los formularios basados en modelos, los cual permite definir los campos que se muestran en el formulario mediante un *array* con el nombre de los campos.

En los formularios basado en clases no se tiene control de la presentación que realiza Django del formulario, por lo cual se deben personalizar los campos mediante *widget.attrs* y se especifica los campos que se desean mostrar en el formulario de la figura 4.19.

```
class Meta:
    model=VariableProceso1
    fields=[
        "nombre",
        "elemento",
        "CondicionInical",
        "descripcion",
    ]
```


Figura 4.19 Campos de formulario para el proceso 1. Fuente: autor

En el archivo HTML, cuando se va a trabajar con un formulario se debe agregar el tag *csrf-_toquen* para proteger al sistema de ataques.

Los *forms* permiten definir el tipo de campo y las validaciones que se pasan a los datos que introducen los usuarios, dicha validaciones pueden ser individuales o a grupos de campos.

Los *forms* cuentan con una función llamada *clean*, la cual es ejecutada cuando el usuario realiza un POST en el formulario, dicha función se usa para transformar la entrada recibida en el POST, por ejemplo, dependiendo de la selección del usuario al manipular una variable, se puede definir la clase de la variable dependiendo si es análoga o digital y reemplazar el campo perteneciente a la clase en el formulario, y así para otros campos.

4.3 MÓDULO DE ALARMAS

En el desarrollo del módulo de alarmas se trabaja con los canales de Django los cuales envían información al *consumers*, este permite tanto escribir código síncrono y/o asíncrono como realiza las transferencias y subprocesos para generar las alarmas en tiempo real.

4.3.1 *Channels*

Channels es el módulo que lleva a Django a una web en tiempo real, es en esencia una cola de tareas la cual escucha los mensajes enviados y delega una acción al *consumer*.

Para hacer uso de c*hannels* se debe instalar dicho módulo mediante el comando *pip install channels* en Windows, posterior a esto se debe agregar *"Channels"* en el área de INSTALLED_APPS ubicado en el archivo *settings.py*.

Los canales envuelven el soporte de vistas asíncrono nativo de Django, el cual permite manejar no solo HTTP sino protocolos de larga duración como lo es el Websocket, permitiendo elegir como se escribe el código el cual puede ser síncrono, asíncrono o una mezcla de ambos.

4.3.2 *Channels layer*

Las capas de canales (c*hannels layer*) son el mecanismo de transporte usado por *Channles* para el envío de mensajes hacia al consumidor. Se debe configurar una capa de canal que permita hablar entre diferentes instancias de una aplicación y ayude a la creación de una aplicación distribuida en tiempo real, sin necesidad de enviar todos los mensajes a través de una base de datos.

La capa de canal en memoria es utilizada para fines de realizar las pruebas de desarrollo local, dicha capa proporciona una copia automáticamente al consumidor mediante el comando *self.Channels_layer*. Se debe agregar el código mostrado a continuación (figura 4.20) en el archivo *settings.py* para hacer uso de la capa de canal en memoria.

```
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels.layers.InMemoryChannelLayer'}
}
```

Figura 4.20 Capa de canal en memoria. Fuente: autor

La configuración anterior es solo para fines de pruebas, para ejecución es necesario instalar Redis y el paquete de *Channels*-Redis el cual se instala mediante el comando *pip install Channels*-Redis desde la consola de comando.

Redis es un motor de base de datos muy eficiente, trabaja con datos volátiles, pues almacena su información en memoria, por lo que su acceso es casi instantáneo. Sin embargo, es posible volcar su información a un medio permanente como lo es PostgreSQL

Redis es la capa de canal oficial de Django, esta utiliza Redis como almacenamiento de respaldo, admite configuraciones de servidor único y fragmentadas, así como soporte grupal. Para usar esta capa de Django es necesario instalarla. Desde su página web.

Redis no es compatible con Windows, por lo cual se descarga un paquete Redis desde la página oficial *Redis.io*, es necesario instalar una versión Redis 5.0 o mayor, ya que dicha versión presenta el nuevo tipo de datos de flujo con grupos de consumidores. Una vez descargado y descomprimida la carpeta, se debe agregar la ruta de dicha carpeta a la variable *path* del sistema y ejecutar el comando *Redis-server* desde la terminal para comprobar su funcionamiento.

Para acceder a la aplicación de Redis se hace por medio de su puerto 6379 predeterminado.

```
CHANNEL_LAYERS = {
    'default' : {
        'BACKEND':'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            'hosts': [('127.0.0.1',6379)]
        },
        'ROUTING':'SCADA.routing'
    }
}
```

Figura 4.21 Capa de canal en proceso. Fuente: autor

4.3.3 *ASGI - Routing*

El servidor asíncrono en el cual se basa c*hannels* es el *ASGI* el cual, al igual que *WSGI* permite elegir entre diferentes servidores. C*hannels* tiene una clase incorporada llamada *ProtocolTypeRouter* que permitirá usar diferentes protocolos para las distintas solicitudes.

Para llamar el enrutamiento (*routing*) se debe configurar una aplicación raíz única y proporcionar la ruta a ella, está se realiza en el archivo *settings.py* la cual se agrega a la altura de *WSGI_APPLICATION* como se muestra en la siguiente figura.

```
WSGI_APPLICATION = 'SCADA.wsgi.application'
ASGI_APPLICATION = 'SCADA.asgi.application'
```

Figura 4.22 Configuración de enrutamiento asíncrono. Fuente: autor

Posteriormente se configura *ProtocolTypeRouter* como aplicación raíz en el cual se anidan los enrutamientos que permiten la redirección a cada uno de los procesos mediante los *routing*.

Se crea un archivo *routing.py*, el cual es similar a los archivos *urls.py* excepto que en lugar de usar http://. se usan rutas de protocolo WebSocket ws://. Este contiene las rutas para enviar los datos actualizados a la interfaz de los distintos procesos. No hay una regla fija sobre donde se debe colocar el *routing.py*, en el presente trabajo se agrega a la altura del archivo *settings.py*.

Se debe asignar un enrutamiento para cada consumidor, para esto se importa el *consumers* de cada proceso y se le asigna una única dirección dentro del archivo r*outing.py* (figura 4.23)


```
from App1.consumer import Proccess_1_Consumer
from App2.consumer import Proccess_2_Consumer
from App3.consumer import Proccess_3_Consumer


websocket_urlpatterns = [
    path('ws/dataP1/', Proccess_1_Consumer.as_asgi()),
    path('ws/dataP2/', Proccess_2_Consumer.as_asgi()),
    path('ws/dataP3/', Proccess_3_Consumer.as_asgi()),
]
```

Figura 4.23 Asignación del routing para cada uno de los consumidores. Fuente: autor

4.3.4 Consumer

El archivo *consumer* es el equivalente al archivo *views.py* pero para WebSocket, en el cual se pueden recibir los mensajes y realizar cualquier lógica que se requiera. Este se ejecuta en forma asíncrona en segundo plano del sistema. Para efectos del presente trabajo, se crea un consumidor

para cada uno de los procesos agregando en cada aplicación, en el que procesarán los datos pertenecientes a cada proceso permitiendo mejorar su rendimiento.

Los consumidores heredan de *WebsocketConsumer* la utilidad *AsyncWebSocketConsumer,* la cual solo usan bibliotecas de forma asíncrona, por lo tanto, no tienen acceso a los módulos síncronos de Django, todas las funciones deben ser de la forma *async def* y se deben usar corrutinas (*await*) para llamar las funciones asincrónicas que realizan E/S.

Los consumidores son piezas de código individuales que permiten leer y escribir mensajes según la comunicación full-duplex mencionada anteriormente; estos consumidores, tienen 3 métodos importantes:

- El de tipo *connect*, que se ejecuta cuando se crea una conexión con el cliente.
- El de tipo *receive*, que cuya función se ejecuta al momento de recibir un mensaje desde el cliente.
- El de tipo *disconnect*, que permite tener un mecanismo cuando se desconecta del cliente.

Debido a que el ORM de Django es un fragmento de código síncrono y se requiere acceder a este desde un código asíncrono se debe realizar un manejo especial, llamando los métodos de la base de datos en un contexto seguro y sincrónico usando la utilidad *Database_sync_to_async* la cual se importa de *Channels.db* de Django.

Al crear dicha función síncrona, se combina con la función asíncrona para comprobar tanto los datos almacenados en la base de datos con el fin de chequear los límites de las variables para la generación de alarmas como los datos que se reciben los cuales son comparados con los límites, para el caso de las variables análogas, si se superan los límites se envía un evento para indicar al usuario que se ha superado una alarma la cual puede ser del tipo LO, LOLO, HI O HIHI, mientras que para las alarmas digitales se verifica el estado de la variable, si es el valor que se recibe para dicha variable es distinto al valor almacenado, se genera una alarma del tipo DSC, simultáneamente, se almacena en la base de datos la alarma generada la cual es mostrada al usuario mediante tablas.

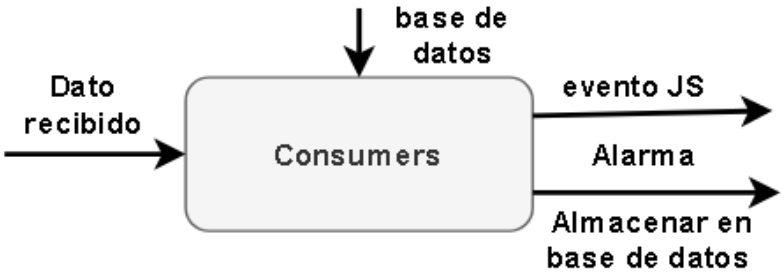

Figura 4.24 Diagrama de generación de alarmas

Para el envío de datos se crea un archivo en *jupyter*, externo al programa para enviar datos aleatorios generados por *random* para cada una de las variables existentes en cada proceso con el fin de realizar pruebas de funcionamiento del sistema, para ello se exporta la librería de WebSocket, esta es una tecnología basada en *ws* que permite enviar los datos generados en el *random* mediante el método *send* a una dirección *routing* específica, la cual varía dependiendo del proceso al cual se envían los datos.

4.3.5 JavaScript

Se crea un archivo de alarmas en JavaScript para cada proceso, el cual se encarga de mostrar los mensajes de alarmas en tiempo real, dichos mensajes se muestran por medio de los marcadores de mensajes. En este archivo se hace el llamado a los datos que envía el cliente mediante la función *newWebSocket(ws://127.0.0.1:8000/)* a la cual se le agrega la dirección definida para cada proceso en el archivo *routing*.

Una vez llamado al *newWebSockets* hace uso de la propiedad *onmessage*, el cual es un controlador de eventos que es llamado cuando se recibe un mensaje del servidor y genera una alarma, la cual se presenta mediante una ventana emergente a través de la librería *SweetAlert2* donde se indica la alarma generada y su nivel de prioridad.

4.4 MÓDULO DE GRÁFICOS REAL E HISTÓRICOS

Al igual que para la generación de alarmas, el archivo *consumers.py* envía los datos recibidos a la base de datos para su almacenamiento y posterior uso, a su vez, también se envían dichos datos a través de WebSocket para ser recuperados en el archivo JavaScript donde se mostrarán los datos mediante gráficas en tiempo real.

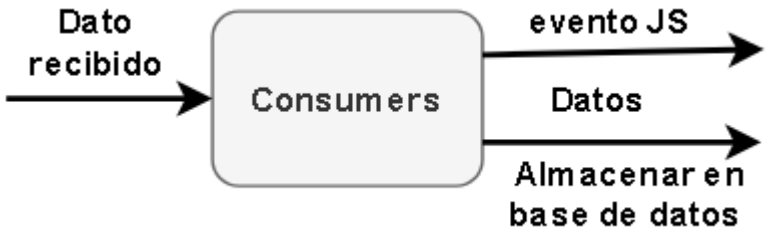

Figura 4.25 Diagrama de procesamiento de datos

Para generar las gráficas en tiempo real se hace uso de la librería de JavaScript *Chart.js*, la cual permite visualizar los datos de una manera fácil e interactiva. Para su uso, se crea un elemento en *canvas* dentro del HTML perteneciente a gráficas, donde se representan los gráficos a mostrar. Dicha librería permite crear diferentes tipos de gráficos y trazar datos en escalas de fecha y tiempo, logarítmica y personalizada.

Las clases *Chart.js* cuentan con muchas propiedades entre las que se tiene:

- *Type*: para seleccionar el tipo de gráficas que se quiere mostrar, en este caso, lineal (*line*)
- *Datasets*: información a mostrar en el gráfico, dentro de este se tienen:
  - *Label*: nombre del gráfico.
  - *Data*: datos a representar
  - *backgroundColor*: color de fondo de cada uno de los datos a representar.
  - *borderColor*: color del borde de cada uno de los datos a representar.

Para generar la gráfica, se crea en la plantilla de HTML una lista de las variables existentes para cada proceso las cuales son consultadas de la base de datos y mostradas en forma de *button*, el cual permite que al seleccionar una de estas variables es enviado al JavaScript un evento mediante *onclick* con el valor de la variable seleccionada.

Luego, en JavaScript se consultan los datos almacenados en la base de datos para la variable seleccionada y se envían dichos datos a la gráfica histórica, tomando los últimos 15 valores almacenados para mostrarlos en la gráfica en tiempo real, estos valores se almacenan en un diccionario, el cual es sustituido por los datos que se reciben mediante WebSocket a través de la propiedad *onmessage* si corresponden con la variable seleccionada previamente.

4.4 MÓDULO DE EVENTOS

Un evento no es una notificación, ya que se registra, pero no se notifica. Los eventos generalmente no se muestran en la pantalla del operador. Si se muestra un evento se debe hacer solo como referencia, este no debe resaltarse.

Los eventos ser registran de forma automática en la base de datos una vez el usuario cree, modifique o elimine una variable o cuando se reconoce una alarma en cada uno de los procesos.

CAPÍTULO V
ANÁLISIS DE RESULTADOS

En el presente capítulo se muestran los resultados obtenidos para cada uno de los módulos desarrollados.

5.1 BASE DE DATOS

Una vez el usuario entre al sistema, se debe dirigir a cada uno de los procesos los cuales cuentan con una barra de menú para navegar por el proceso, al seleccionar la opción de variables aparecerá la ventana de la figura 5.1 la cual inicialmente estará vacía, por lo cual se debe rellenar con los datos referentes al proceso.

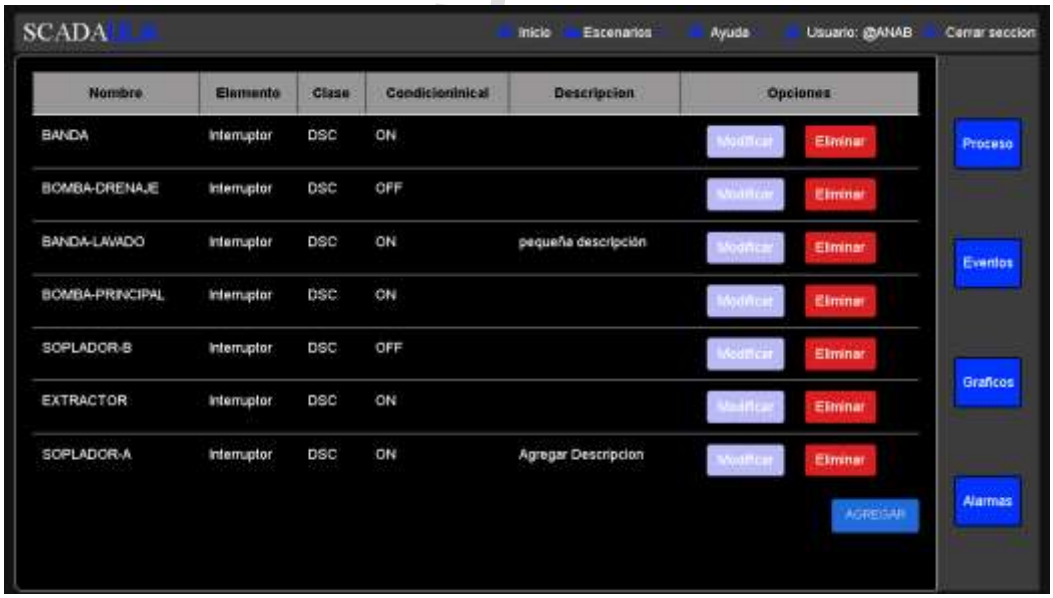

Figura 5.1 Configuración de variables proceso 1, Fuente: autor

En la lista de variables perteneciente al proceso 1 se pueden observar los campos presentes en cada una de las variables creadas, como el nombre, a que elemento y clase pertenece y una

pequeña descripción. En dicho proceso no se configuran las alarmas debido a que todas las variables son del tipo digital.

Para los procesos 2 y 3 se realiza la creación de las alarmas de manera similar al proceso 1, con la diferencia de que se debe establecer el *SetPoint* y los límites de activación de alarmas para las variables análoga como se muestra en las figuras 5.2 y 5.3.

Figura 5.2 Configuración de variables del proceso 2. Fuente: autor

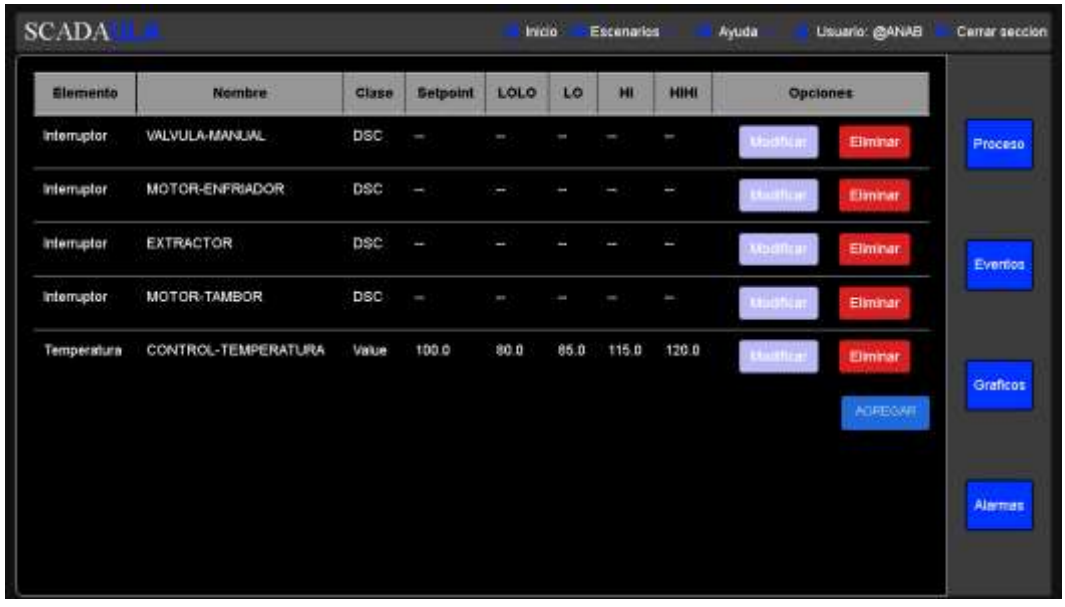

Figura 5.3 Configuración de variables del proceso3. Fuente: autor

Al seleccionar la opción de agregar, se despliega un formulario (figura 5.4) el cual debe ser rellenado, se debe tener en cuenta que el formulario tiene campos obligatorios como lo es el nombre, el cual debe ser único y el elemento al cual pertenece. En el caso del proceso 1 no cuenta con la opción de rellenar los campos de límites y *SetPoint*, pues sus variables son digitales y no es necesario agregar dichos campos.

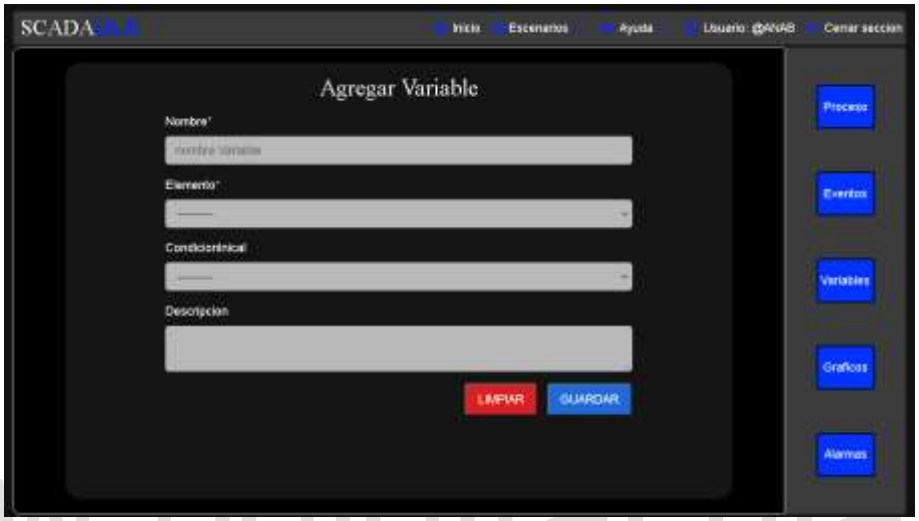

Figura 4.26 Formulario de agregar nueva variable en proceso 1. Fuente: autor

En el caso de los formularios pertenecientes a los procesos 2 y 3, si es obligatorio agregar los límites y el *SetPoint* para las variables análogas (figura 5.5), en las variables digital no es necesario, pues estos campos son reemplazados automáticamente por el campo "--" en el archivo *views.py* antes de ser enviados a la base de datos, para el caso de las variables digitales, solo se debe establecer una condición inicial la cual será la que tome dicha variable al iniciar el proceso. El campo de descripción es opcional, de igual forma se recomienda agregarla para mayor detalle de donde se generan las alarmas

Una vez completados los campos solicitados en el formulario, se tienen dos opciones, limpiar o guardar los datos ingresados. Al presionar el botón *limpiar* se borran los datos antes ingresados, pero al presionar *Guardar*, aparece automáticamente una ventana emergente notificándole que se ha guardado correctamente la variable, como se muestra en la figura 5.6, simultáneamente, se almacenan los datos registrados en la tabla de base de datos llamada *VariableProcesos* perteneciente al proceso en que se encuentre, posteriormente la página se redirecciona a la

página de configuración de variables, donde se pueden verificar los datos almacenados para las distintas variables creadas a modo de prueba para comprobar su funcionamiento.

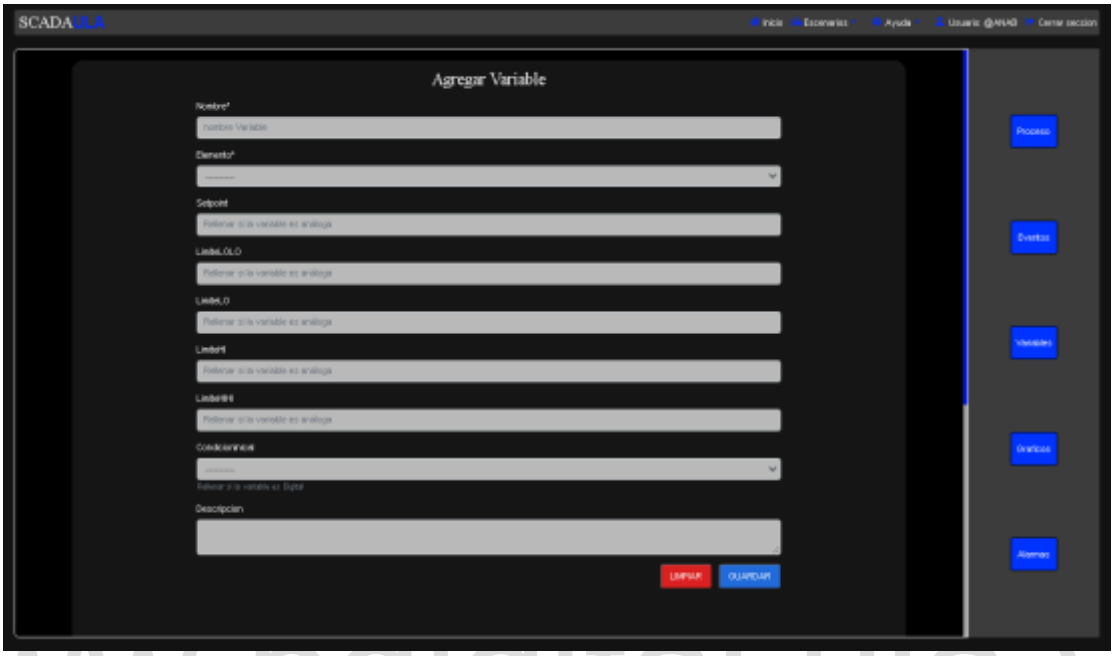

Figura 5.4 Formulario para agregar nueva variable, proceso 2 y 3. Fuente: autor

Figura 5.5 Confirmación de variable creada. Fuente: autor

Cada una de las variables almacenadas cuanta con un botón para modificar y otro para eliminar dicha variable, como se muestra en las figuras de configuración de variables pertenecientes a las figuras 5.1, 5.2 y 5.3 mostradas anteriormente.

Al seleccionar la opción modificar en una de las variables almacenadas, se despliega un formulario similar al de agregar variables (figuras 5.4 y 5.5), pero esta vez los campos se auto rellenan con la información perteneciente a la variable seleccionada almacenada en la base de datos como se muestra en la figura 5.7, al igual que el formulario de agregar variable, una vez modificada dicha variable se despliega una ventana emergente (figura 5.8) y se redirecciona nuevamente a la página de configuración de variables de las figuras 5.1, 5.2 o 5.3, esta depende del proceso en que se encuentre. El nombre de las variables no es posible modificarlo debido a que este es único en la base de datos.

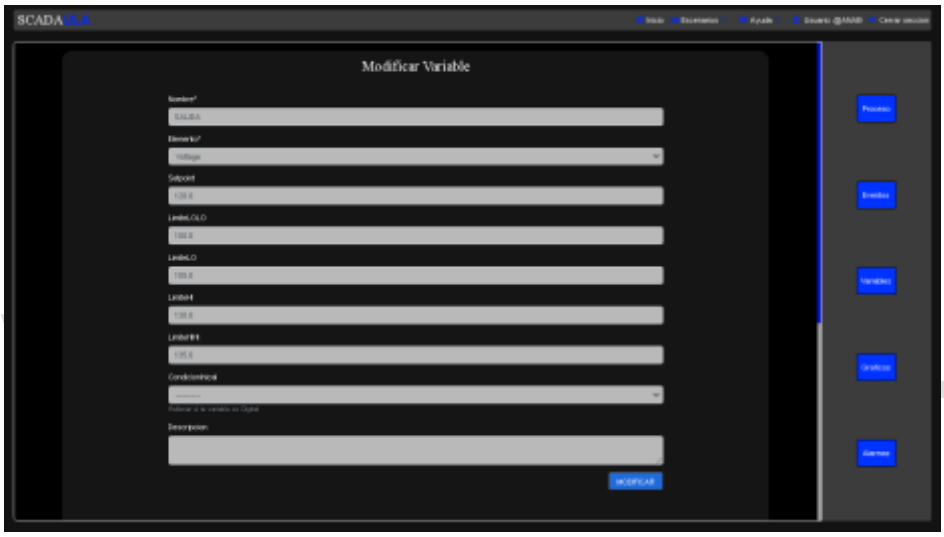

Figura 5.6 Modificar variable. Fuente: autor

Figura 5.7 Notificación de variable modificada. Fuente: autor

A la hora de seleccionar la opción de eliminar una variable en la configuración de variable (figura 5.1, 5.2 o 5.3), se despliega una ventana emergente para confirmar dicha acción (figura 5.9 a). Una vez confirmada la eliminación, se despliega la ventana de la figura 5.9. b), de lo contrario se mostrará la lista de configuración de variable nuevamente.

Figura 5.8 (a) Confirmación para eliminar. (b) Notificación de variable eliminada. Fuente: autor

Los datos de entradas se almacenan en la tabla de la base de datos llamada *DatosEntrada* a la cual se envían los datos desde el *consumer* de la aplicación para cada proceso, dichos datos aparte de ser enviados a la base de datos, se envían a un archivo JavaScript que permite trabajar en tiempo real, este grafica los datos de entrada en tiempo real y comprueba las alarmas del sistema. Si se genera una alarma, se envían automáticamente todos los datos relacionados a dicha alarma a la tabla de *AlarmasGeneradas* para luego ser mostrado al usuario mediante tabla en el HMI.

Las generaciones de eventos se almacenan automáticamente desde el archivo *views.py* cada vez que el usuario cree, modifique o elimine una variable o cuando se reconozca una alarma generada.

5.2 ALARMA

Las alarmas se generan automáticamente una vez se superen los límites de las variables establecido por el usuario, dichas alarmas generadas son almacenadas en la tabla *AlarmasGeneradas* de cada proceso para luego ser mostrada mediante tabla, como se muestra en la figura 5.10 donde se identifican claramente el nivel de prioridad de cada alarma por su color de identificación.

- Las alarmas de tipo LOLO o HIHI tienen prioridad 1 y se muestran en color rojo
- Las alarmas de tipo LO o HI tienen prioridad 2 y se muestran de color rojo
- Las alarmas de tipo DSC tiene prioridad 3 y se muestran de color amarilla

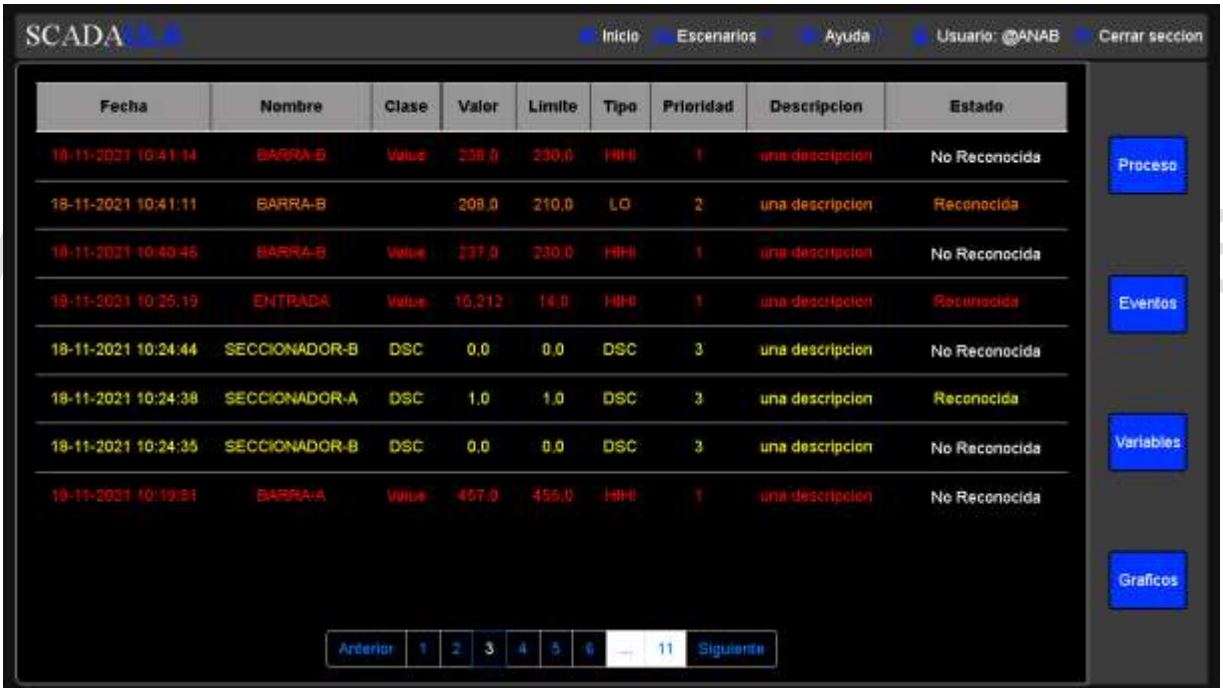

Figura 5.9 Alarmas generadas. Fuente: autor

Al generarse una nueva alarma en tiempo real, se muestra al operador una ventana emergente con una alerta de la alarma que se ha generado y que limite se ha superado en la cual, según el tipo de alarma se muestra el icono de color rojo, naranja o amarillo.

Figura 5.10 Alerta de alarma. Fuente: autor

Cada una de las alarmas generadas (figura 5.10) cuenta con un *Estado* donde las alarmas tienen la opción de ser reconocidas por el operador, una vez el operador seleccione la opción de reconocer la alarma se despliega una ventana emergente indicando la confirmación de dicho reconocimiento (figura 5.12), al confirmar este, se actualiza la página para realizar el cambio y se muestra el estado de dicha alarma como reconocida en la tabla de alarmas generadas mostrada en la figura 5.10.

Figura 5.11 Confirmación de reconocimiento de alarma. Fuente: autor

5.3 GRÁFICAS

Al abrir la ventana de gráficas de un proceso en específico, se muestra una lista de botones en la parte derecha de la pantalla, con todas las variables almacenadas en la base de datos, al seleccionar una de estas variables se muestra en la parte superior una gráfica con los últimos 15 datos almacenados en la base de datos los cuales se van a ir sustituyendo cada vez ingrese un nuevo valor en tiempo real para dicha variable (figura 5.13).

En la parte inferior se observa la gráfica histórica de la variable seleccionada, la cual en la parte superior muestra la etiqueta de las gráficas, y el parte inferior se puede observar la fecha y la hora en la que se tomó la muestra con respecto al valor adquirido, dicha gráfica cuenta con *scroll* tanto en horizontal como en vertical el cual permite navegar en esta para observar su comportamiento en el tiempo y una pequeña ventana interactiva que se despliega al posicionar el cursor sobre un punto en concreto, esta ventana muestra fecha y hora, nombre de la variable y el valor exacto del resultado. Simultáneamente, en las variables análogas se muestra el valor del *SetPoint* establecido para la variable graficada (figura 5.14).

Dichas gráficas fueron comprobadas para cada uno de las variables de cada proceso.

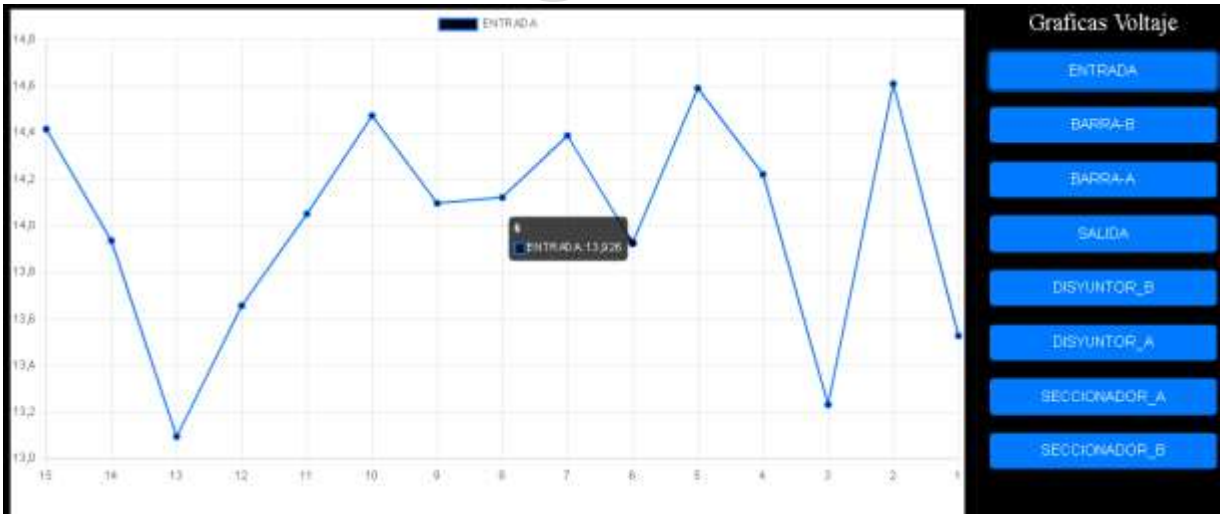

Figura 5.12 Gráfica en tiempo real, proceso 2. Fuente: autor

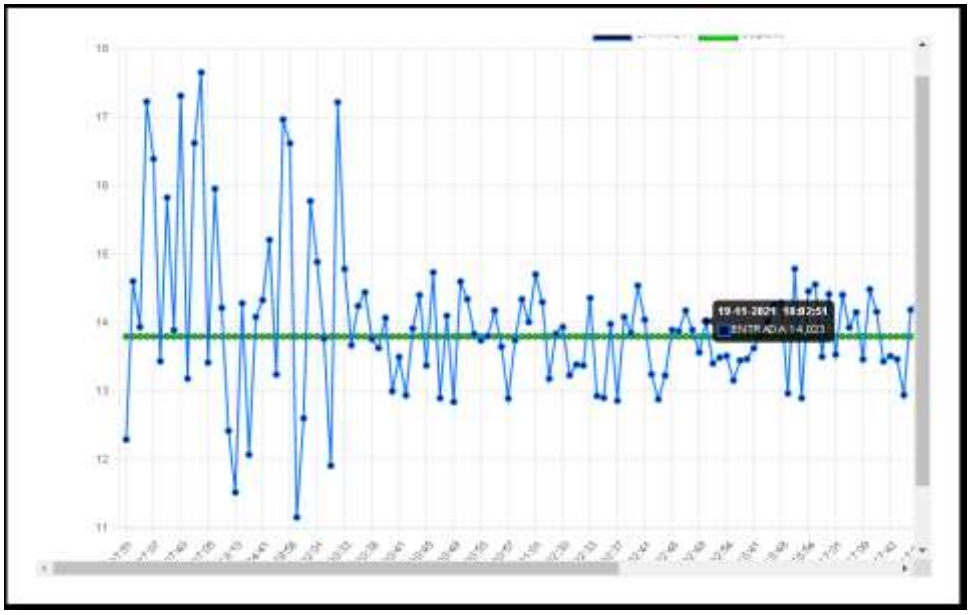

Figura 5.13 Gráfica histórica, proceso 2. Fuente: autor

5.4 EVENTOS

Una vez almacenados los eventos en la tabla *Eventos* de la base de datos, se muestran mediante tabla (figura 5.15) donde se expone la fecha y hora en la que se generó, porque acción se generó, y una breve descripción donde se resalta que usuario generó dicho evento y en que variable ocurrió este.

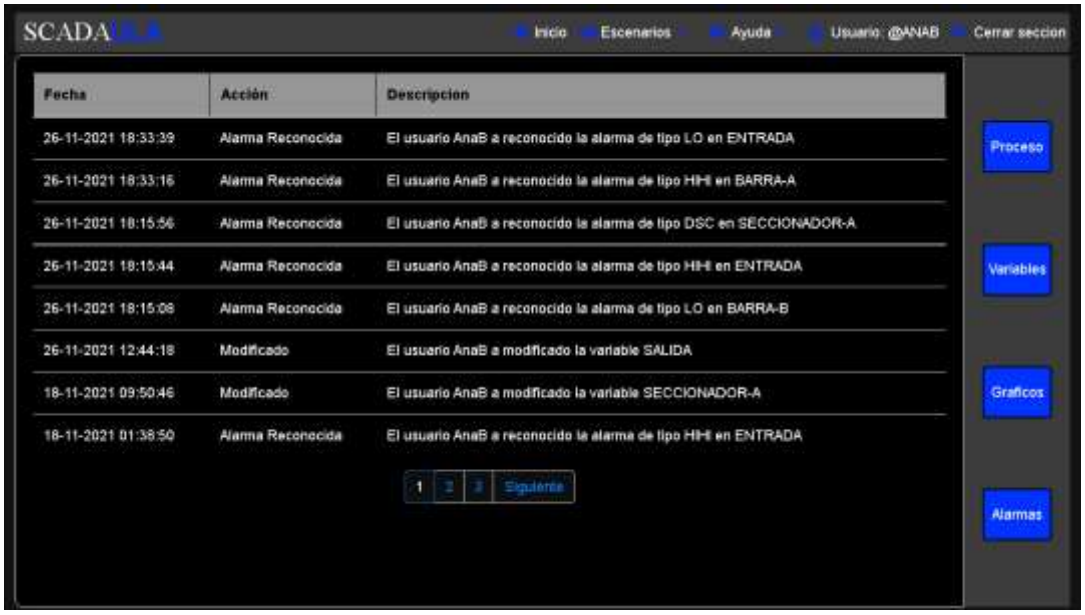

Figura 5.14 Lista de eventos. Fuente: autor

5.5 INTEGRACIÓN DE MÓDULOS CON EL HMI

Se realizó la unión de los módulos pertenecientes al presente proyecto con el módulo HMI [40] para el sistema SCADA en desarrollo, combinando los archivos *views.py*, *models.py* y *urls.py* e integrando los ficheros *consumer.py* y *forms.py* junto con los archivos HTML ubicados en la carpeta *template* de cada aplicación, también los archivos CSS y JavaScript que se encuentran en la carpeta *static* de cada una de las aplicaciones.

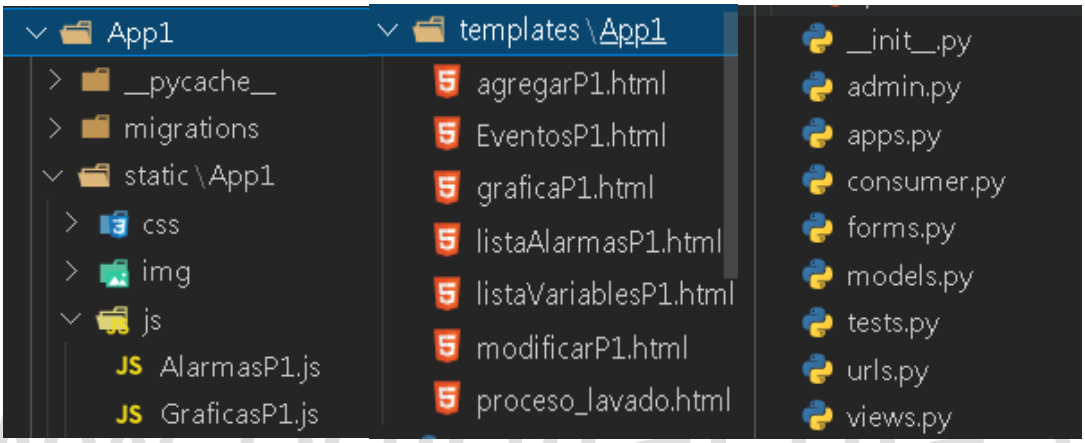

Figura 5.15 Archivos pertenecientes al proceso 1. Fuente: autor

A continuación, se detalla el contenido de cada uno de los archivos que se encuentran en App1 los cuales pertenecen al proceso 1, cabe destacar que los procesos 2 y 3 son similares al proceso 1, por tal motivo no es necesario especificar el contenido de sus archivos.

La carpeta de *template* contiene archivos como:

Agregar: permite estructurar el formulario para agregar nuevas variables en los procesos.

Eventos: contiene la estructura para mostrar la tabla con cada evento generado junto con la paginación que permite una visualización más cómoda de todos los eventos.

Gráficas: contiene el esqueleto de las gráficas las cuales se realizan dentro de un archivo canvas, desde dicho archivo se envía al JavaScript de gráficas toda la información referente a las distintas variables del proceso para la representación de estas tanto en tiempo real como en histórica.

ListaAlarmas: contiene la estructura a representar mediante tablas, donde se exponen todas y cada una de las alarmas generadas, las cuales cuentan con una paginación que permite mayor comodidad a la hora de visualizar dichas alarmas. Cada alarma cuanta con un enlace en su última columna que permite reconocerla. En dicho archivo también se especifica el color de cada uno de los datos para lo cual se toma en cuenta la prioridad de cada alarma.

ListaVariables: la estructura de dicho archivo permite generar las tablas que muestran las variables existentes junto con la paginación, este archivo cuenta con botones en cada variable los cuales redirigen a los archivos modificar y eliminar, así como también un botón para agregar a la base de datos las variables requeridas por el proceso.

Modificar: su estructura es igual que la de agregar, solo cambia su funcionalidad.

En la carpeta *static* se encuentra el archivo CSS perteneciente a cada una de las aplicaciones donde se define el estilo de las páginas, también se encuentran los archivos JavaScript, se cuenta con un archivo de alarmas el cual, con ayuda de los marcadores de mensajes, permite mostrar al usuario las alarmas de cada proceso en tiempo real, también se cuenta con un archivo para las gráficas, donde se implementa la librería *Chart.js* para el desarrollo de las gráficas tanto en tiempo real como en histórica.

El fichero de *forms.py* contiene toda la estructuración referente a los formularios mostrados en los archivos agregar y modificar

El fichero *consumer.py* procesa los datos de entrada al sistema en tiempo real.

CONCLUSIONES

Para llevar a cabo el presente trabajo de grado referente al diseño de un sistema SCADA basado en software libre se analizaron distintos softwares de desarrollo con el fin de seleccionar los el más adecuados para el diseño de los distintos módulos.

En el caso del módulo de base de datos se seleccionó el sistema de gestión de base de datos PostgreSQL, gracias a las características que ofrece entre las que destacan: cuenta con una arquitectura cliente-servidor, es compatible con el modelo relacional, fácil conexión con Python y Django a través de un adaptador de base de datos en Python y está entre las aplicaciones de base de datos más avanzadas, de código abierto y con una licencia única, pues, MySQL, a pesar de que es más popular que PostgreSQL, cuenta con una licencia dual.

Como software de desarrollo se seleccionó el *framework* de desarrollo Django el cual está escrito en Python, es de código abierto y cuenta con características fundamentales para el almacenamiento y análisis de los datos adquiridos, se trabajó con Django c*hannels* el cual permitió crear aplicaciones en tiempo real y conexiones persistentes entre el navegador del usuario y el servidor web mediante WebSocket. Los canales permitieron la comunicación entre varios procesos por medio de la transmisión de mensajes.

Se comprobó la creación, modificación y eliminación de las variables para cada uno de los procesos verificando su correcto funcionamiento, dichas variables se almacenaron en la base de datos y se visualizaron mediante tablas mostradas en la interfaz del HMI.

También se comprobó el almacenamiento de los datos de entradas los cuales son generados mediante valores *random* para cada variable, esto permitió comprobar el funcionamiento de la gráfica tanto en tiempo real, utilizando las funciones síncronas y asíncronas de los canales a

través de WebSocket, como las gráficas históricas en la cual se visualizan los datos de las distintas variables almacenados en la base de datos.

La supervisión y generación de alarmas en tiempo real se realizó mediante los consumidores los cuales enviaban la información a una ventana emergente para las variables cuyo *random* superaba los límites establecidos, esta se almacenó en la base de datos para su posterior análisis y visualización a través de tablas mostradas en el HMI.

El módulo de eventos permitió crear y almacenan automáticamente los eventos que se generaron cada vez que el usuario efectuaba una acción en las variables o alarmas. Estos se visualizan en la interfaz del HMI en forma de tabla.

La utilización de códigos de colores en los elementos del SCADA le permite al usuario una mayor visualización de alarmas en la supervisión del proceso. Dicho código de colores le permite al usuario darle prioridad a los elementos que se encuentran fuera del estándar definido.

Se procedió a la integración de los distintos módulos desarrollados con el módulo HMI, para el cual se realizaron distintas pruebas, se comprobaron cada uno de los módulos integrados y se verificó que las pruebas realizadas al programa funcionaron correctamente.

RECOMENDACIONES

Es importante realizar capacitaciones a todo el estudiantado en cuanto a los sistemas SCADA con el fin de que se familiaricen con el área de automatización y control la cual está muy demandada en la mayoría de las industrias tanto nacionales como internacionales.

Se recomienda seguir desarrollando el software de manera continua, para darle al sistema una mayor funcionalidad, y globalizarlo con el fin de que este pueda adaptarse a cualquier entorno en el que se necesite automatizar los procesos de supervisión y control.

Es recomendable reforzar la seguridad del sistema y así aprovechar las funcionalidades que trae Django en dicho ámbito, el cual es muy completo

En el caso de la lista de alarmas generadas, se recomienda agregar un filtro que permita consultar dichas alarmas por jerarquía de prioridad, o filtrar las alarmas que ya han sido reconocidas o no por el usuario.

También se recomienda agregar una función que permita consultar las gráficas históricas mediante rango de fecha para una mayor comodidad del usuario.

REFERENCIAS

[1] M. T. Hernández, «Manual de trabajo de Grado de Especialización y Maestría y Tesis Doctorales,» FEDUPEL. La editorial pedagógica de Venezuela, Caracas, 2006.

[2] R. D. Moya, «El Proyecto Factible: una modalidad de investigación,» FEDUPEL. La editorial pedagógica de Venezuela, Caracas, 2002

[3] Sanchéz , G; Custodio, A, «Servidor para un sistema de supervisión y control de procesos industriales bajo software libre,» UNEXPO, vol. Vol.14, nº Nº.54, pp. p.31-44, Marzo 2010.

[4] C. Abaffy y J. Lárez, «Desarrollo de Scada en una Plataforma de Software Libre,» CITEG Revista Arbitrada, nº N°. 2, pp. pp.88-97, Noviembre 2007.

[5] E. Pérez, «Los Sistemas Scada en la automatización industrial",» Tecnología en marcha, vol. Vol. 28, nº N°.4, pp. pág. 3-14, Febrero 2015.

[6] Blanco. O, Controladores Lógicos Programables, Sistemas SCADA y Protocolos de Comunicación Industriales, Mérida, 2019.

[7] Villajulca. J., «Curso de sistemas scada,» 12 2009. [En línea]. Disponible: https://instrumentacionycontrol.net/category/control/control-y-automatizacion/curso-de-sistemas-scada/. [Último acceso: 21 08 2021].

[8] A. R. Penin., Sistemas SCADAS, Mexico: Alfaomega Grupo Editorial S.A, 2013.

[9] G. M. Vela., «Gestión De Alarmas En Sistemas De Control Distribuido Filosofía De Alarmas Para Una Central Térmica De Ciclo Combinado Y Eficiencia Del Sistema De Alarmas,» 09 2014. [En línea]. Disponible: https://cupdf.com/document/gestion-de-alarmas-en-sistemas-de-control-distribuido-gerardo-marina.html. [Último acceso: 07 09 2021].

[10] C. V. D. Henst, «¿Qué son Bases de Datos?,» 26 09 2001. [En línea]. Disponible: http://www.maestrosdelweb.com/que-son-las-bases-de-datos/. [Último acceso: 07 09 2021].

[11] N. Chapaval, «Bases de datos: qué tipo existen y cómo funcionan,» Platzi, 2018. [En línea]. Disponible: https://platzi.com/blog/bases-de-datos-que-son-que-tipos-existen/. [Último acceso: 24 11 2020].

[12] EcuRed contributors, «"Sistema Gestor de Base de Datos".,» EcuRed, 13 07 2019. [En línea]. Disponible: https://www.ecured.cu/index.php?title=Sistema_Gestor_de_Base_de_Datos&oldid=3455261. [Último acceso: 09 01 2021].

[13] PostgreSQL, «What is PostgreSQL?,» PostgreSQL, 2020. [En línea]. Disponible: https://www.postgresql.org/about/. [Último acceso: 11 08 21].

[14] «PostgreSQL: el gestor de bases de datos a fondo,» ionos, 2021. [En línea]. Disponible: https://www.ionos.es/digitalguide/servidores/know-how/postgresql/. [Último acceso: 16 09 2021].

[15] S. Francisco Alonso, «Programacion en SQL con PostgreSQL,» [En línea]. Disponible: https://www.academia.edu/10174027/Programaci%C3%B3n_en_SQL_con_PostgreSQL. [Último acceso: 28 09 2021].

[16] M. Presta, «Los 10 principales lenguajes de programación de backend,» Back4app, 2020. [En línea]. Disponible: https://blog.back4app.com/es/lenguajes-de-programacion-de-backend/. [Último acceso: 09 10 2021].

[17] N. Chapaval, «Qué es Frontend y Backend: diferencias y características,» Platzi, 2017. [En línea]. Disponible: https://platzi.com/blog/que-es-frontend-y-backend/. [Último acceso: 23 10 2021].

[18] «Tutorial HTML,» w3schools, 2021. [En línea]. Disponible: https://www.w3schools.com/html/. [Último acceso: 12 08 2021].

[19] Mozilla, «Conceptos básicos de HTML,» Mozzila, 12 03 2020. [En línea]. Disponible: https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics#entonces_%C2%BFqu%C3%A9_es_html_en_realidad. [Último acceso: 19 08 2021].

[20] C. Heilmann, «Conceptos básicos de CSS,» 12 03 2017. [En línea]. Disponible: http://mosaic.uoc.edu/ac/le/es/m6/ud1/. [Último acceso: 19 09 2021].

[21] Concepto Definición, «JavaScript,» 30 01 2021. [En línea]. Disponible: https://conceptodefinicion.de/javascript/. [Último acceso: 04 09 2021].

[22] J. López, «Estructura Básica De Una Función JavaScript,» [En línea]. Disponible: https://josetxu.com/blog/estructura-basica-de-una-funcion-javascript/. [Último acceso: 06 09 2021].

[23] Programacion.net, «9 librerías de javascript para crear gráficos interactivos,» Programacion en Castellano, S.L., 10 15 2016. [En línea]. Disponible: https://programacion.net/articulo/9_librerias_de_javascript_para_crear_graficos_interactivos_1365. [Último acceso: 29 09 2021].

[24] R. Pernaz , «Python: El lenguaje de programación más popular para aprender en 2021,» 01 2021. [En línea]. Disponible: https://www.crehana.com/blog/web/que-es-python/. [Último acceso: 03 11 2021].

[25] Python Software Foundation, «Python,» [En línea]. Disponible: https://www.python.org/about/. [Último acceso: 24 09 2021].

[26] C. Kopecky, «¿Qué es Node.js? Introducción al tiempo de ejecución de JavaScript para principiantes,» educative, 30 09 2020. [En línea]. Disponible: https://www.educative.io/blog/what-is-nodejs. [Último acceso: 02 10 2021].

[27] Á. Robledano, «Qué es C++: Características y aplicaciones,» Openwebinars, 22 Julio 2019. [En línea]. Disponible: https://openwebinars.net/blog/que-es-cpp/. [Último acceso: 19 10 2021].

[28] A. Gardey y J. Pérez Porto, «DEFINICION DE PHP,» definicion.de, 2012. [En línea]. Disponible: https://definicion.de/php/. [Último acceso: 11 09 2021].

[29] D. Ortego Delgado, «¿Qué es C#? Introducción,» Openwebinars, 29 03 2017. [En línea]. Disponible: https://openwebinars.net/blog/que-es-c-introduccion/. [Último acceso: 2 10 2021].

[30] Mozilla, «Introducción a Django,» mozzilla, 18 05 2021. [En línea]. Disponible: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction. [Último acceso: 12 09 2021].

[31] «Django Concepts,» RIP Tutorial, [En línea]. Disponible: https://riptutorial.com/django/example/2355/django-concepts. [Último acceso: 08 10 2021].

[32] A. Cruz, «*WebSockets* en Django empleando Django *Channels*, Servidores wsgi y asgi,» Desarrollo Libre, 18 12 2020. [En línea]. Disponible: https://www.desarrollolibre.net/blog/django/*WebSockets*-en-django-empleando-django-*Channels*. [Último acceso: 17 10 2021].

[33] Real the Docs,, «ASGI,» Read the Docs, 2018. [En línea]. Disponible: https://asgi.readthedocs.io/en/latest/index.html. [Último acceso: 28 09 2021].

[34] Desarrollo web, «¿Qué es WebSocket?,» ionos, 17 08 2020. [En línea]. Disponible: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-web*Socket*/. [Último acceso: 16 08 2021].

[35] Django Software, Foundation, «*Channels*,» Read the Docs, 2018. [En línea]. Disponible: https://*Channels*.readthedocs.io/en/stable/. [Último acceso: 27 09 2021].

[36] «¿Qué es Flask Python?,» python basics, 2021. [En línea]. Disponible: https://pythonbasics.org/what-is-flask-python/. [Último acceso: 12 10 2021].

[37] Amazon, «what is redis,» aws, 2021. [En línea]. Disponible: https://aws.amazon.com/es/elasticache/what-is-redis/. [Último acceso: 17 08 2021].

[38] Wikipedia, «Visual Studio Code,» 2021. [En línea]. Disponible: https://es.wikipedia.org/wiki/Visual_Studio_Code. [Último acceso: 12 08 2021].

[39] campusmvp, «Cómo son los desarrolladores en 2021 según Stack Overflow,» Stack Overflow, 12 08 2021. [En línea]. Disponible: https://www.campusmvp.es/recursos/post/como-son-los-desarrolladores-en-2021-segun-stack-overflow.aspx. [Último acceso: 03 11 2021].

[40] J. J. Rojas R., Desarrollo de modulo sinóptico para un sistema SCADA basado en software libre para ser implementado en el Laboratorio de Control de la Facultad de Ingeniera, Mérida - Venezuela: Universidad de los Andes, s, No Publicado, 2022.

[41] L. L. Rondón S., Desarrollo conceptual de un sistema SCADA basado en software libre y diseño de los módulos servidor y comunicación, para el Laboratorio de Control de la Escuela de Ingeniería Eléctrica de la Universidad de Los Andes, Mérida - Venezuela: Universidad de los Andes, s, No Publicado, 2022.