Universitat Oberta de Catalunya

uoc.edu

Aplicación segura para la automatización y análisis del consumo eléctrico doméstico a través de contadores digitales

Ángel Cardiel Ferrero

Grado en Ingeniería Informática

Desarrollo web

Nombre Consultor/a: Gregorio Robles Martínez

Nombre Profesor/a responsable de la asignatura: Santi Caballé Llobet

Junio 2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Aplicación segura para la automatización y análisis del consumo eléctrico doméstico a través de contadores digitales*
Nombre del autor:	*Ángel Cardiel Ferrero*
Nombre del consultor/a:	*Gregorio Robles Martínez*
Nombre del PRA:	*Santi Caballé Llobet*
Fecha de entrega (mm/aaaa):	06/2022
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo web*
Idioma del trabajo:	***Castellano***
Palabras clave	*Energía, consumo, web*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

En un contexto de crisis energética y climática, y ante un mercado eléctrico excesivamente complejo, y que se ha demostrado que solo tiene intereses puramente mercantilistas, se plantea la necesidad de acercar al consumidor información sobre su consumo energético para que de alguna forma tome conciencia sobre como mejorar sus medidas de consumo, sus hábitos y rutinas de consumo, sus estadísticas y así poder reducir la huella de carbono y la factura de la luz.

Para la realización de este proyecto se han utilizado tecnologías de desarrollo modernas que actualmente están demandadas en el mercado laboral y pone en práctica mucho de los conceptos aprendidos a lo largo del Grado de Ingeniería Informática.

Con un compromiso claro por la democratización de la energía como bien de primera necesidad, y el conocimiento compartido, sirva este proyecto como base para construir a futuro una solución útil para el consumidor. Por lo que son bienvenidas cuantas aportaciones, y proyectos derivados se quieran.

Abstract (in English, 250 words or less):

In a context of energy and climate crisis, and in the face of an excessively complex electricity market, which has been shown to have exclusively mercantilist interests, the need arises to provide consumers with information about their energy consumption so that they become aware of how to improve their consumption measures, their consumption habits and routines, their statistics and thus be able to reduce their carbon footprint and electricity bills.

To carry out this project we have used modern development technologies that are currently in demand in the labour market and put into practice many of the concepts learned throughout the Degree in Computer Engineering.

With a clear commitment to the democratisation of energy as a basic necessity, and shared knowledge, this project serves as a basis for building a useful solution for the consumer in the future. Therefore, as many contributions and derivative projects are welcome.

Agradecimientos

A Carla, Carmen, Lola, Nieves, Inma, e Ignacio

por su tiempo, su apoyo, sus cuidados y su cariño
durante todos estos años que me han permitido llegar hasta aquí.

A mis hijas Julia y Eva. A mi padre.

Índice

Lista de figuras

1. Introducción

La presente memoria es la concreción por escrito de un proyecto de aplicación web que pretende sintetizar mucho de los conceptos adquiridos durante el Grado de Ingeniería Informática y servir como Trabajo de Fin de Grado. Incluye por tanto, su análisis, definición, modelado, y posterior implementación.

Existe una motivación personal por aportar a la comunidad de forma desinteresada una herramienta que simplifique el acceso y análisis de los datos de consumo energético de instalaciones domésticas, y que a futuro pueda establecer estrategias de recomendaciones que redunden una mejor optimización de los recursos energéticos así como de la reducción de los costes que una familia debe asumir por un bien de primera necesidad como lo es el acceso a la energía.

En todo momento se ha procurado mantener un fuerte enfoque de servicio social, de conciencia energética y de respeto a la privacidad del consumidor (“usuario” en adelante) así como a la seguridad de información.

1.1. Contexto y justificación del Trabajo

Desde hace años la comunidad científica viene advirtiendo sobre los grandes riesgos que como sociedad tenemos que afrontar ante el consumo desproporcionado y desmedido del gasto energético. El científico leonés Antonio Turiel, especialista en recursos energéticos habla de que “Científicos y organizaciones de la ONU llevan tiempo diciendo que estamos haciendo cosas insostenibles pero las continuamos haciendo; y llega un momento en el que todo colapsa” [1]. El sistema capitalista, globalista y de libre mercado se está topando con los límites del crecimiento. Meadows y Randers en ‘Los límites del crecimiento (de un mundo finito) se plantea documentalmente que estamos cerca de alcanzar los límites físicos o de tolerancia den nuestra civilización [2]. Son muchos los gobiernos e instituciones que se plantean de forma inexcusable reducir la huella energética dada la escasez de determinados combustibles fósiles. Alicia Valero del grupo de Ecología Industrial del Instituto CIRCE profundiza en lo que hay detrás de la actual crisis de suministro de materias primas y componentes: la escasez de minerales [3].

Esta situación recrudece las relaciones geopolíticas marcadas principalmente por el acceso y legitimidad de acceso a dichos recursos. Especialmente en los últimos meses hemos podido ver como se han generado conflictos bélicos en el que la compraventa del gas, una de nuestras principales fuentes fósiles de energía, ha sido moneda de cambio para los diferentes gobiernos. Tanto es así que la Unión Europea ha anunciado recientemente un plan para desconectarse

del combustible ruso tras la guerra entre Rusia y Ucrania iniciada formalmente en Febrero de 2022 [4]. Una de las consecuencias ha sido una escalada de precios inaudita que está afectando directamente a los consumidores que atraviesan una crisis inflacionaria con niveles similares a la crisis de 1.985 con un aumento disparado por la energía y los alimentos [5].

Figura 1: Evolución precio diario de la luz en MWh

En nuestro país sufrimos a día de hoy unos precios excepcionalmente elevados por el acceso a la energía. El 8 de marzo de 2.022 se batieron todos los récords histórico sobre el precio de la luz en el mercado mayorista [6]. Esto está provocando que siga aumentando la desigualdad social. Según Luis Ayala y Olga Cantó, las primeras evidencias sobre los efectos económicos de la pandemia apuntan a un aumento de la desigualdad y la pobreza mayor al del resto de los países de la UE-27 [7]. Además, de una forma inequívoca, pone de manifiesto la dura existencia de la 'pobreza energética', que se ceba especialmente en Andalucía. Según la Estrategia a Largo Plazo para la Rehabilitación Energética en el Sector de la Edificación en España, en su actualización de 2020, Andalucía está por encima de la media española. 3 de cada 10 hogares sufren actualmente pobreza energética en Andalucía, tal y como se indica en la Estrategia a Largo Plazo para la Rehabilitación Energética en el Sector de la Edificación en España publicado por el Ministerio de Transportes, Movilidad y Agenda Urbana [8].

El mercado energético es además tremendamente complejo y por tanto muy desconocido por gran parte de la población que a duras penas sabe diferenciar si se rigen por contratos de mercado libre o mercado regulado. El Panel de Hogares de la CNMC, en su publicación del segundo semestre de 2019 [9], pone de relieve el desconocimiento de los usuarios en torno a la factura de la luz: seis de cada diez hogares ignoran en qué mercado tienen contratado el suministro eléctrico y que en pocas ocasiones sabe diferencia entre empresa generadora, comercializadora o distribuidora. Surgen a raíz de esta realidad determinados intereses

privados que ven aquí una oportunidad de generar negocio causando aun más frustración y desapego en el consumidor.

Esta aplicación pretende aportar un pequeño grano de arena para facilitar la lectura de nuestro consumo energético, establecer estrategias de reducción de la cantidad de energía utilizada en nuestros domicilios y hacerlo con un enfoque antimercantilista, anticapitalista y ecologista.

1.2. Descripción y definición

Se trata de un aplicativo que permite consultar datos concretos de nuestro contador, revisar el histórico de consumos por tramos horarios, anticipar las próximas facturas y dar algunas claves que nos permitan controlar el consumo en nuestro día a día. Además, se plantea el almacenaje de esta información en la nube de forma cifrada para que la información almacenada solo pueda ser leída e interpretada exclusivamente por las personas interesadas.

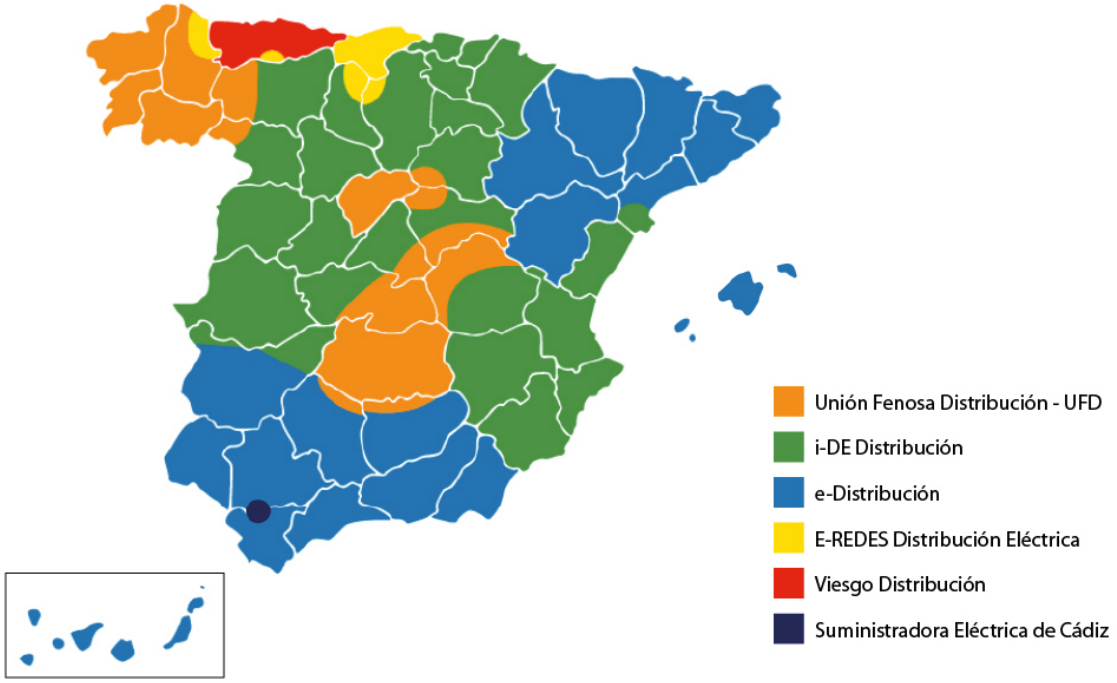


Figura 2: Empresas distribuidoras de energía en el estado español

Quiénes somos clientes de e-distribución (empresa distribuidora de la Endesa), que somos principalmente quienes vivimos en Cataluña, Baleares, Canarias y Andalucía tenemos a nuestra disposición un panel de cliente. Sin embargo no contamos con una API que nos permita sacar el máximo partido a los datos. Por el momento e-distribución solo ofrece API pública para consulta de averías y cortes programados y para alta y seguimiento de solicitudes de conexión a la red, pensadas para empresas del sector y no para el consumo general [10]. Por tanto una de las partes del proyecto será crear una librería que sea capaz de leer

adecuadamente la información de nuestro contador haciendo uso de nuestras credenciales en e-distribución.

El resultado a buscar será un aplicativo que permita consultar datos concretos de nuestro contador, revisar el histórico de consumos por tramos horarios, anticipar las próximas facturas y dar algunas claves que nos permitan controlar el consumo en nuestro día a día. Además, se plantea el almacenaje de esta información en la nube de forma cifrada para que la información almacenada solo pueda ser leída e interpretada exclusivamente por la/s persona/s interesadas.

1.3. Objetivos del Trabajo

- Desarrollar una librería específica que permita la lectura de la información facilitada por la empresa distribuidora que opera en parte del estado (en mi caso se plantea trabajar con e-distribución de Endesa que no cuenta con API pública).
- Crear una aplicación dividida en frontend y backend[1] para la automatización, almacenaje e interpretación de la información.
- Se desarrollará un entorno de desarrollo aislado, replicable y multiplataforma
- Los datos estarán almacenados de forma encriptada de forma que sólo el usuario autorizado sea capaz de decodificarlo.
- Que el aplicativo pueda predecir el precio del siguiente recibo
- Que el usuario pueda consultar el histórico de consumos de forma sencilla e intuitiva
- Que el usuario pueda discriminar el consumo basado en tramos horarios predefinidos.

1.4. Enfoque y método seguido

Al desarrollo de este proyecto no se conoce ninguna aplicación sin interés mercantil que te ayude a mejorar el consumo con datos reales de tu contador. Muchos de ellos solicitan fichero CSV con las lecturas. Además, como cooperativista y cliente de Som Energía, existe una herramienta llamada InfoEnergía que da algunas de las claves de este proyecto. Som Energia es una cooperativa catalana sin ánimo de lucro que fomenta la democratización de la energía como generadores y comercializadores de energía verde. El servicio InfoEnergia que proporciona Som Energia es una herramienta para ayudar en la comprensión del uso de la energía y acompañar a las personas socias en su camino hacia una mayor eficiencia energética y ahorro económico. [12]

1 Frontend se refiere a la capa de presentación en el modelo OSI y suele estar asociado a la vista del lado del cliente. Backend se refiere a la capa de acceso a datos y suele estar asociado a la vista del lado del servidor. Se diseñan divididos siguiente la separación de intereses (separation of concerns, SoC, Dijkstra 1974, [11]) que se sigue en ciencias de la computación.

Por tanto el proyecto se dividirá en dos grandes grupos:

- Creación y publicación de librería para interactuar con e-distribucion
- Aplicación web (llamada Calambre) que haga uso de la librería para lecturas, análisis e interpretación.

1.5. Planificación del Trabajo

TIPO	ASUNTO	FECHA DE INICIO	FECHA DE FINALIZACIÓN
TASK	PEC1 - Plan de trabajo	16/02/2022	01/03/2022
TASK	Definición de TFG	16/02/2022	20/02/2022
TASK	Manifiesto	21/02/2022	23/02/2022
TASK	Inicio de la memoria	24/02/2022	28/02/2022
MILESTONE	Entrega PEC1	01/03/2022	01/03/2022
TASK	PEC2 - Análisis y diseño	02/03/2022	01/04/2022
TASK	Definir tecnologías y alca...	02/03/2022	06/03/2022
TASK	Crear librería PHP eDistri...	07/03/2022	20/03/2022
TASK	Redacción de la memoria	22/03/2022	30/03/2022
MILESTONE	Entrega PEC2	01/04/2022	01/04/2022
TASK	PEC3 - Implementacion	02/04/2022	23/05/2022
TASK	Desarrollo API	02/04/2022	23/04/2022
TASK	Modelado de datos	02/04/2022	06/04/2022
TASK	Implementación	07/04/2022	15/04/2022
TASK	Testing	16/04/2022	23/04/2022
TASK	Desarrollo Frontend	24/04/2022	16/05/2022
TASK	Definir Layout	24/04/2022	02/05/2022
TASK	Implementación	03/05/2022	16/05/2022
TASK	Redacción de la memoria	17/05/2022	23/05/2022
MILESTONE	Entrega PEC3	24/05/2022	24/05/2022

TIPO	ASUNTO	FECHA DE INICIO	FECHA DE FINALIZACIÓN
TASK	PEC4 - Memoria y presentaci...	24/05/2022	09/06/2022
TASK	Pruebas	24/05/2022	28/05/2022
TASK	Finalizar memoria	29/05/2022	08/06/2022
MILESTONE	Entrega PEC4	09/06/2022	09/06/2022
TASK	Defensa ante el tribunal	10/06/2022	19/06/2022

Figura 3: Tablas de planificación temporal

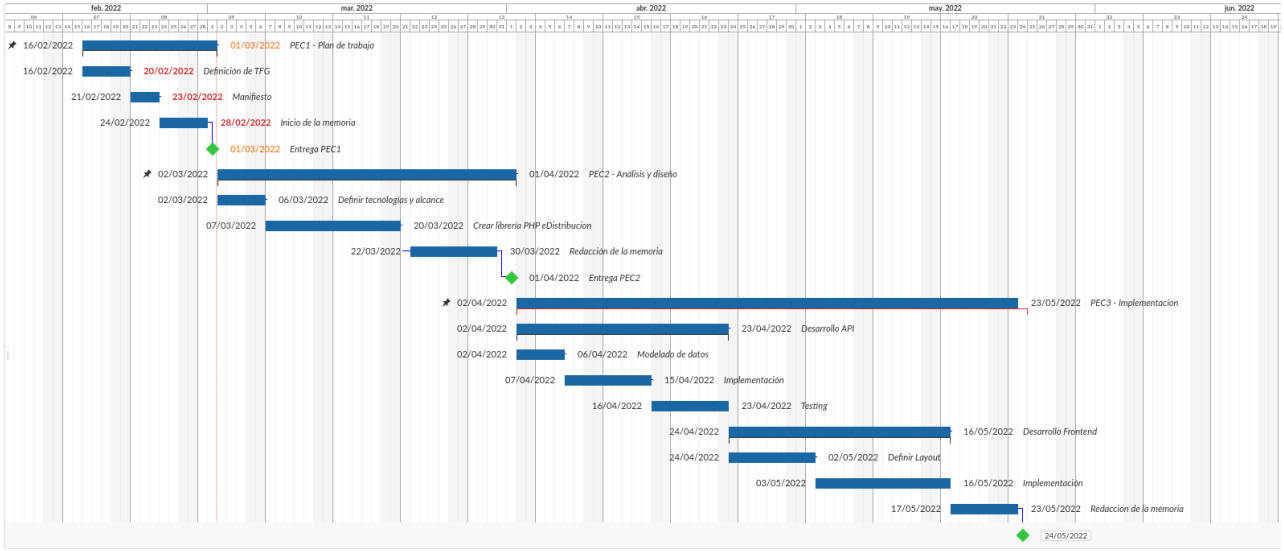


Figura 4: Diagrama de Gantt (Ver Anexo I)

1.6. Breve sumario de productos obtenidos

- Creación y publicación de librería para interactuar con e-distribucion
  - Librería: https://github.com/ancafe/edis
- Aplicación web (llamada Calambre) que haga uso de la librería para lecturas, análisis e interpretación.
  - Backend: https://github.com/ancafe/calambre-back
  - Frontend: https://github.com/ancafe/calambre-front

1.7. Breve descripción de los otros capítulos de la memoria

Haremos un análisis inicial y una definición de los requisitos que serán clasificados como formales y no formales. A continuación detallaremos el desarrollo tanto de la librería como de la aplicación web. Se reseñarán los retos más complejos y la solución planteada. Luego pasaremos a explicar como se instala desde cero. Finalizaremos con unas conclusiones en las que se hablarán de renuncias hechas durante el proyecto y visión de futuro. Termina la presente memoria con un glosario que ayude a entender los principales conceptos y una bibliografía.

2. Análisis y diseño técnico

A continuación analizamos los diferentes requisitos que definen las decisiones y directrices que ha seguido el proyecto. Los clasificaremos en requisitos funcionales y requisitos no funcionales

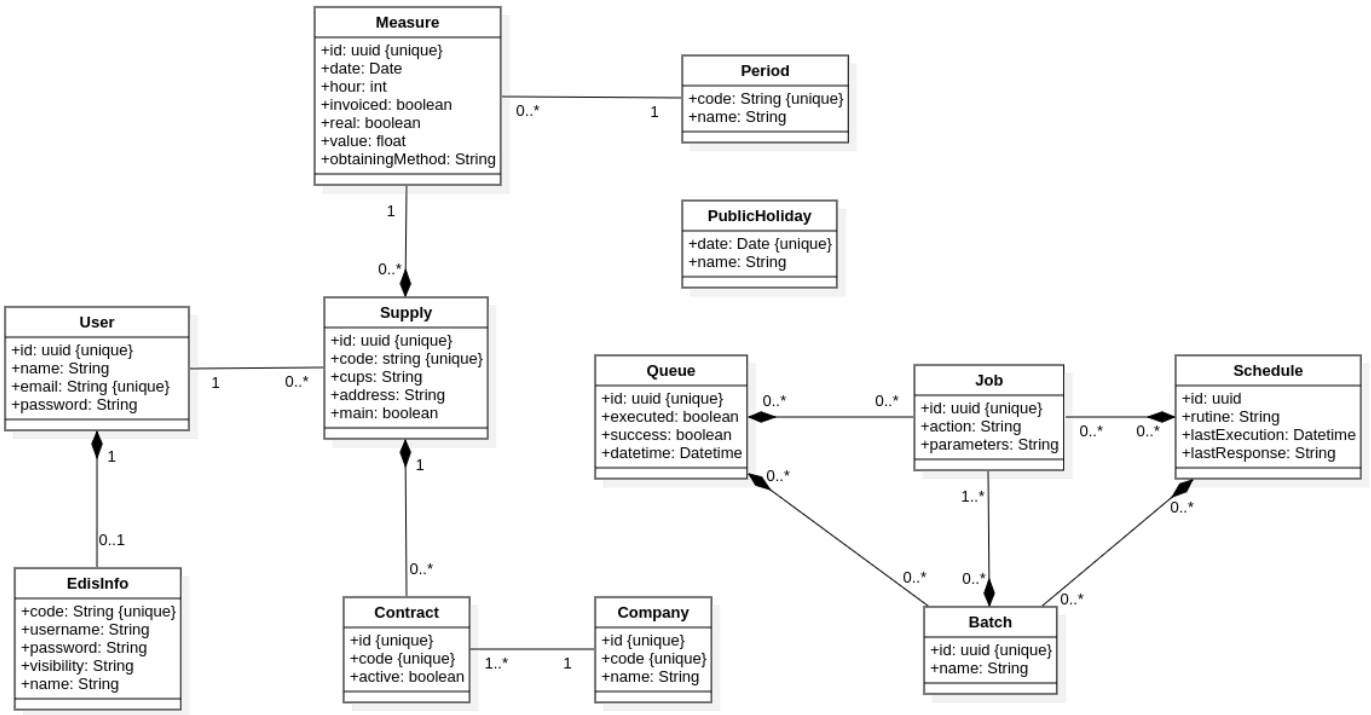
2.1. Requisitos funcionales

Entendemos por requisitos funcionales aquella funcionalidad que requerimos que esté presente durante el desarrollo de un software. Según Wiegers, una función en términos computacionales es aquella que dado un mismo conjunto de entradas y comportamientos, obtenemos una misma salida [13], que son aquellos que establecen cuál debe ser el comportamiento del sistema dado un conjunto de entradas y salidas definidas. Podríamos resumir los requisitos funcionales por aquellos requisitos que definen «qué debe hacer el sistema».

A través de los siguientes sub-apartados iremos aproximándonos al diseño y modelo del conjunto de funciones que a nivel de concepto debe cumplir el sistema propuesto.

2.1.1. Modelo del dominio

Se presenta un diagrama de entidades en el estándar UML así como las diferentes relaciones entre ellas



Se definen las siguientes entidades:

User: Entidad elemental de identificación

EdisInfo: Información de acceso a la plataforma e-distribución

Supply: Contador de luz con dirección asociada

Company: Empresa comercializadora

Contract: Contrato que vincula un contador con una compañía comercializadora

Measure: Entidad de medida. Incluye cantidad de kWh, así como fecha y periodo.

Period: Diferentes periodos regulados por el Gobierno[2] en el BOE 21, de 24/01/2020 [14]

PublicHoliday: Listado de festivos nacionales para cálculo de periodos especiales

Job: Acción para ser procesada por la cola.

Queue: Cola de procesado.

Schedule: Planificador de tiempo para automatizaciones.

Batch: Lote de trabajos para ser procesados en la cola

2.1.2. Modelo de relación

Para el almacenamiento de los datos hemos optado por una base de datos relacional. Si bien es cierto que la entidad `Measures` crecerá con rapidez y tendrá grandes volúmenes de datos, se consideran que ni las búsquedas van a ser complejas ni su estructura variará en el tiempo, que son las principales ventajas de los motores no relacionales. Tal y como comenta Saenz Escobar [15], el principal enfoque de los motores de bases de datos NoSQL (no relacionales), "es maleabilidad, alta Disponibilidad y velocidad para manejar gran cantidad de datos que pueden variar constantemente."

Se plantea además que todos los registros almacenen al menos dos campos adicionales de control: `created_at` y `updated_at` para poder hacer una traza de cada una de las entidades.

Además, como se explica en el apartado de Requisitos no funcionales sobre Requisitos de seguridad (página 28), utilizaremos en varias de las entidades un campo *user* que nos servirá para aplicar políticas RLS (Row Level Security). PostgreSQL, que ofrece esta funcionalidad de forma nativa, tal y como indica en documentación oficial [16], aplicando RLS permitimos que las tablas puedan tener políticas de seguridad de filas que restrinjan, por usuario, qué filas pueden

2 Se definen a nivel doméstico 3 tramos (P1, P2, P3) con horarios fijos y excepcionalidades según el tipo de día de la semana y festivos. Para el ámbito industrial, se establecen 6 tramos (P1 – P6) que varían además en función del mes del año.

ser devueltas por consultas normales o insertadas, actualizadas o eliminadas por comandos de modificación de datos. Esta función también se conoce como seguridad de nivel de fila. Es por ello que optamos por utilizar PostgreSQL como motor de datos relacional, que es un motor de código libre que enfatiza en la extensibilidad y cumplimiento de estándares y se ajusta a nuestros requisitos funcionales.

Adicionalmente se pueden ver tablas auxiliares que controlarán la ejecución de las automatizaciones y trabajos en lote como se explica detalladamente en los Requisitos no funcionales así como los relacionados con el reseteo de claves por parte del usuario y control de migraciones de datos que el framework utilizado brinda por defecto.

El diagrama entidad relación quedará por tanto de la siguiente forma:

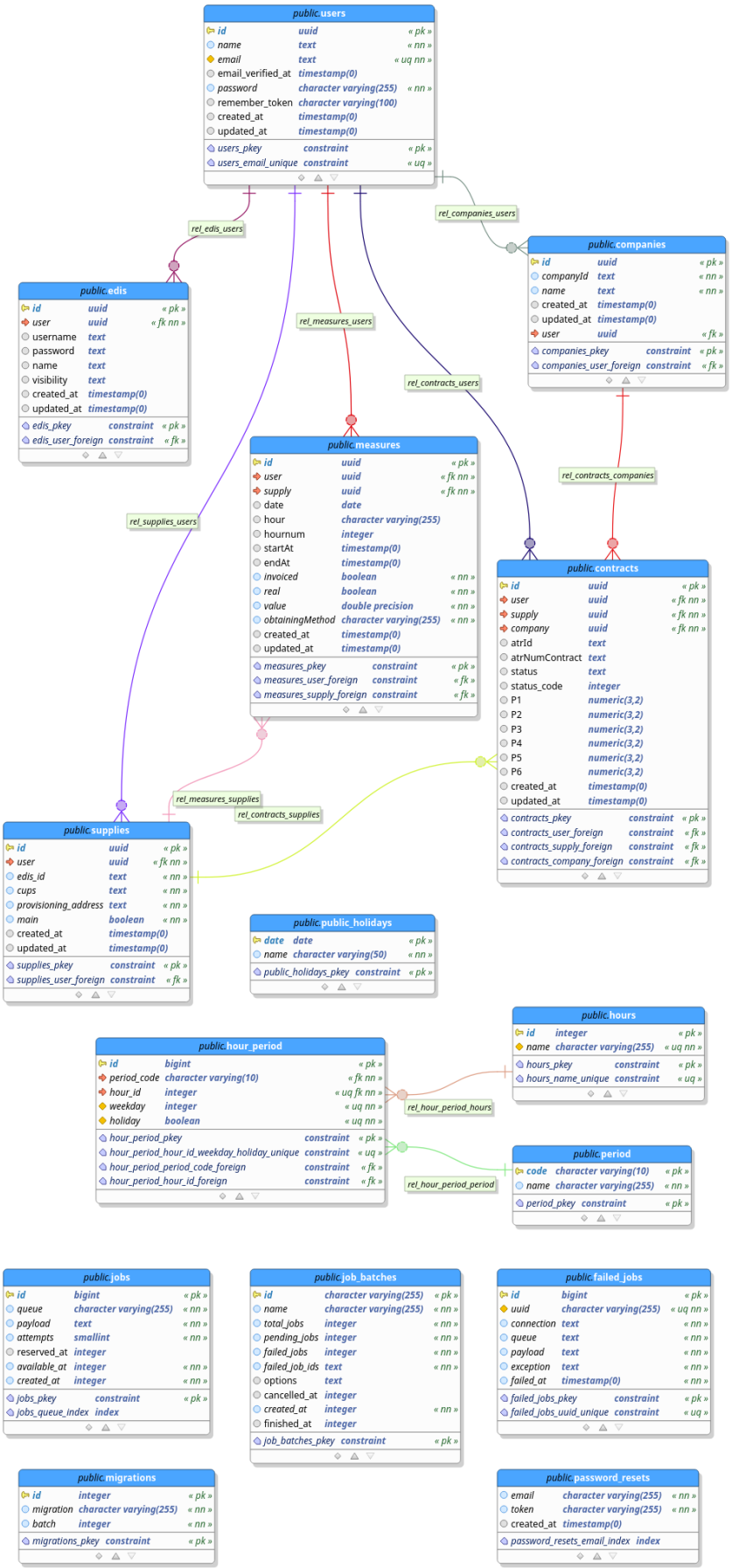


Figura 5: Modelo ER

2.1.3. Casos de uso

Analizamos las funcionalidades requeridas así como las necesidades que van a tener los usuarios de la aplicación. Nos ayudará a crear los flujos de interacción. Se concluye que únicamente existirán dos roles definidos: aquellos usuarios registrados y los que acceden de forma anónima.

ID	Descripción	Rol
RF-ANON-01	Registrarse como usuario	Anónimo
RF-ANON-02	Leer política de datos	Anónimo
RF-USER-01	Iniciar sesión	Usuario registrado
RF-USER-02	Cerrar sesión	Usuario registrado
RF-USER-03	Almacenar datos E-Distribución	Usuario registrado
RF-USER-04	Modificar datos E-Distribución	Usuario registrado
RF-USER-05	Eliminar datos E-Distribución	Usuario registrado
RF-USER-06	Eliminar cuenta y datos	Usuario registrado
RF-USER-07	Obtener información de suministros	Usuario registrado
RF-USER-08	Leer histórico de datos	Usuario registrado
RF-USER-09	Analizar consumos por días	Usuario registrado
RF-USER-10	Analizar consumos por día de la semana	Usuario registrado
RF-USER-11	Analizar consumos por meses del año	Usuario registrado
RF-USER-12	Consulta completo de medidas por día y hora	Usuario registrado

Tabla 1: Requisitos funcionales

2.1.4. Diagrama y descripción de los casos de uso

La función principal del aplicativo es de consulta y siempre siendo el usuario el administrador único y exclusivo de su propia información. No existe la figura del administrador y no existen a priori casos adicionales a la consulta e interpretación de consumos.

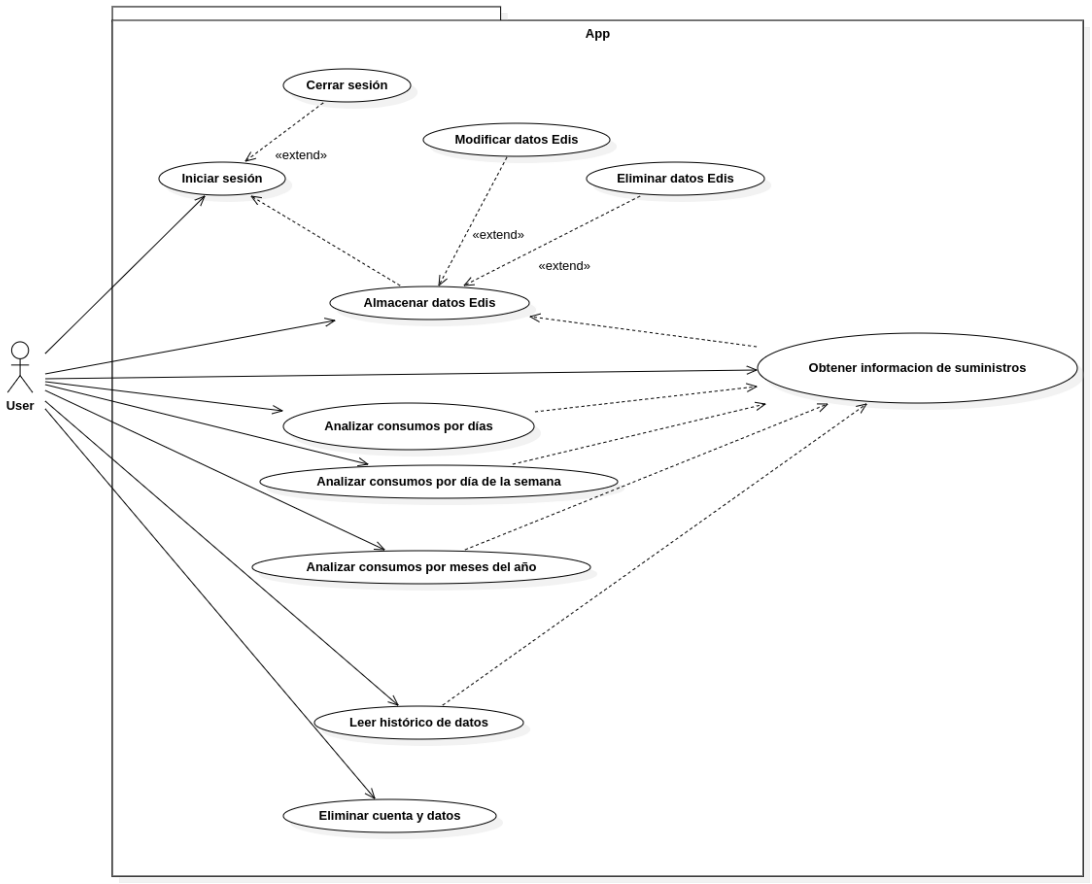


Figura 6: Diagrama de casos de uso

2.1.5. Casos de uso a nivel de usuario

Se muestran a continuación una selección a modo de ejemplo de las fichas de los diferentes casos de uso a nivel de usuario.

Nombre	**INICIAR SESIÓN**
Código	RF-USER-01
Actores	Usuario
Roles	Usuario Registrado
Propósito	Poder identificarse en la aplicación de forma segura y acceder a sus datos de consumo
Precondiciones	El usuario debe estar registrado
Flujo principal	Accede a la URL principal Introduce su usuario y contraseña
Excepciones al caso	El usuario podrá acceder exclusivamente a los enlaces públicos (registrarse, recuperar contraseña, ...) mientras no esté identificado

Tabla 2: Descripción RF-USER-01

Nombre	**MODIFICAR DATOS DE E-DISTRIBUCIÓN**
Código	RF-USER-04
Actores	Usuario
Roles	Usuario Registrado
Propósito	Poder establecer las credenciales de acceso al panel de control de la empresa distribuidora.
Precondiciones	El usuario debe estar registrado
Flujo principal	Hacer login en la aplicación Navegar hasta «Profile» Introducir datos de conexión a E-distribución Salvar los datos
Flujo secundario	Si los datos introducidos no logran conectar con la empresa distribuidora, mostrará un mensaje de error.

Tabla 3: Descripción RF-USER-04

Nombre	Analizar medias de consumo por día de la semana
Código	RF-USER-10
Actores	Usuario
Roles	Usuario Registrado
Propósito	Obtener información sobre el consumo por semana natural (L-D) en un gráfico de barras apiladas discriminando por periodo de consumo.
Precondiciones	El usuario debe estar registrado El usuario debe tener medidas de consumo registradas
Flujo principal	Hacer login en la aplicación Navegar por Supply » CUPS » Monthly
Flujo secundario	Si no existen medidas de consumo registradas, se mostrará la gráfica vacía con un mensaje descriptivo

Tabla 4: Descripción RF-USER-10

2.1.6. Diagrama de secuencia

Tal y como se explica en el correspondiente apartado *"Colas: Organizando las llamadas a terceros" (página 51)*, todas las peticiones a la empresa distribuidora que nos proporciona las lecturas del contador se hacen a través de un gestor de colas.

En nuestro caso usamos Redis como solución abierta y robusta, teniendo en cuenta que no se define simplemente como un gestor de colas. En realidad, tal y como aparece en su documentación oficial, Redis es un motor de base de datos en memoria que se basa en tipo de datos de diccionario o tabla de hashes[17] . Se utiliza como base de datos NoSQL, como gestor de caches, de colas, de pilas, etc.

En este caso Redis en combinación con Laravel utiliza políticas basadas en el tiempo para prevenir llamadas rechazadas por DDoS a la empresa distribuidora haciendo uso de las técnicas descritas por Shahzeb Ahmed en "How to set API Rate Limiting in Laravel" [18]. Es necesario implementar una solución de limite de uso de la API no solo para llamadas entrantes, sino especialmente para llamadas salientes hacia servidores terceros. Debemos controlar el número de peticiones estableciendo un máximo de llamadas por unidad de tiempo. Esto es así porque al no disponer de API pública, debemos forzar un mecanismo de prevención previo para controlar el número máximo de llamadas por segmento de tiempo

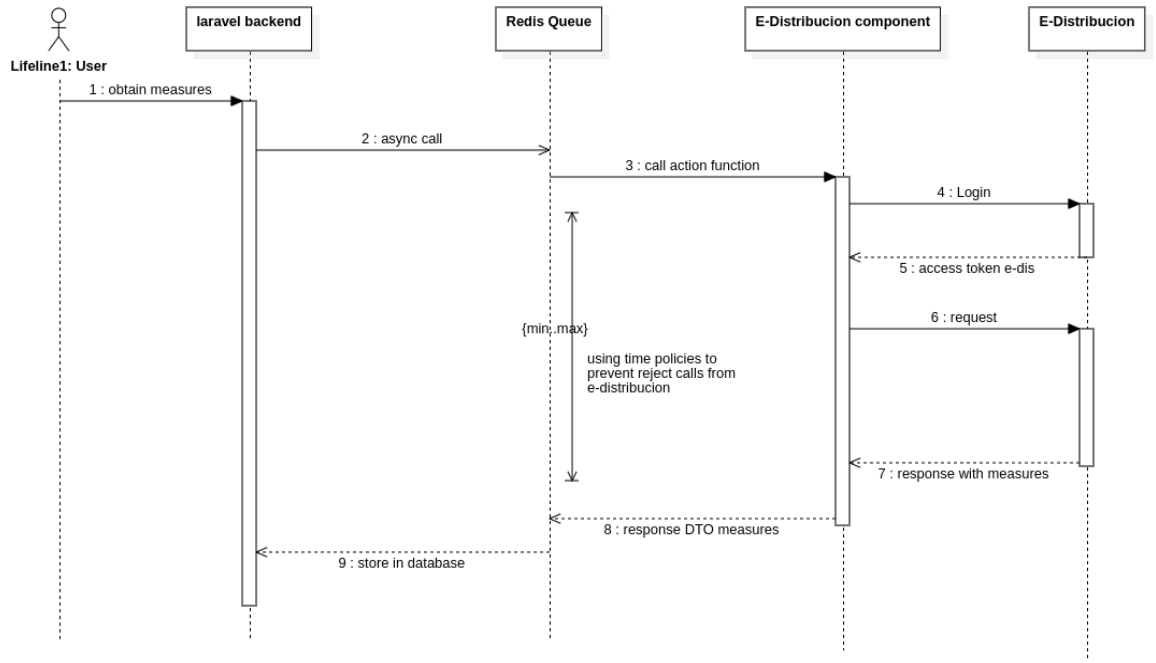


Figura 7: Diagrama de secuencia

2.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que imponen restricciones en el diseño o la implementación (como requisitos de rendimiento, seguridad o confiabilidad). Podríamos resumir los requisitos no funcionales por aquellos requisitos que definen «como debe ser el sistema».

2.2.1. Infraestructura

El presente proyecto se plantea como una solución self-hosted[3] que pueda ser ejecutada de la forma más sencilla posible, es decir, con la menor configuración necesaria posible, en un entorno local y totalmente aislado. Aun así, es potencialmente convertible a una solución SaaS[4] en la nube en la que puedan convivir diferentes datos.

Dada la sensibilidad de los datos tratados, y siempre con un enfoque de respeto absoluto a la información contenida, este proyecto se enfoca diferentes capas de seguridad y con el menor número de servicios dependientes de terceros posibles.

Se plantea dividir el aplicativo en dos disciplina técnicas diferentes: una parte de backend que se encargará de responder a las llamadas entrantes, tratar los datos, comunicarse con el componente que nos permite acceder directamente a la información de la empresa distribuidora y gestionar usuarios y colas de peticiones, entre otras. Y otra parte aislada e independiente, que será frontend; un proyecto de NuxtJS[5] que nos permitirá trabajar de forma integrada con VueJS y todo su ecosistema (vue-store, vue-routes, etc.), en un entorno totalmente reactivo para aplicaciones modernas con diseños adaptables. VueJS es uno de los frameworks más demandados actualmente según *State of JS* [19] situándose el segundo lugar en el ranking de interés por debajo de Svelte.

El backend se apoyará en servicios auxiliares: estos son; Redis para gestión de colas (explicado en apartados posteriores), Mailhog[6] para captura a nivel de desarrollo de correos entrantes y salientes, Laravel Horizon[7] para procesado y visualización de colas, y PostgreSQL como motor de base de datos.

3 Conocemos self-hosted como el auto-alojamiento de servicios para consumo interno o consumo propio sin contar con entornos de producción públicos. Por el momento no tiene proyección de ser un proyecto de uso público en la nube por su estado de madurez. Pero es usable en entornos controlados.

4 SaaS: Software as a Service.

5 NuxtJS es una librería de código libre que a su vez usa vueJs como framework de interfaces reactivas. Se basa originalmente en NextJS, para React.

6 MailHog es una herramienta de prueba de correo electrónico con un falso servidor SMTP. MailHog ejecuta un servidor SMTP que captura los correos electrónicos salientes que se envían pudiendo monitorizarlos durante la fase de desarrollo

7 Horizon forma parte del ecosistema de Laravel y monitorea métricas claves como la tasa de rendimiento, tiempo de ejecución y fallas de tareas. Útil para el control de procesado de colas de Redis.

El frontend, a su vez, lo hará en librerías que faciliten la creación rápida y dinámica de componentes. Principalmente nos apoyaremos en ApexCharts[8] para el dibujado y renderizado dinámico de gráficas de consumo.

Ambas aplicaciones se comunicarán de forma segura con JWT (JSON Web Token). De cara a facilitar la instalación y la integración de componentes, usaremos Traefik como proxy inverso. Además, nos apoyaremos en GitHub para repositorio como control de versiones.

Cada uno de los servicios irá definido en un contenedor de Docker y orquestado con Docker Compose que nos permitirá aislar el entorno y que cualquier persona que quiera desplegar el proyecto cuente con todas las dependencias. Se especifica el despliegue en el apartado “Repositorios e instrucciones de despliegue” (página 64).

8 ApexCharts es una biblioteca de gráficas JavaScript que permite crear visualizaciones de datos interactivas y reactivas.

Por tanto, a nivel de infraestructura utilizaremos el siguiente esquema:

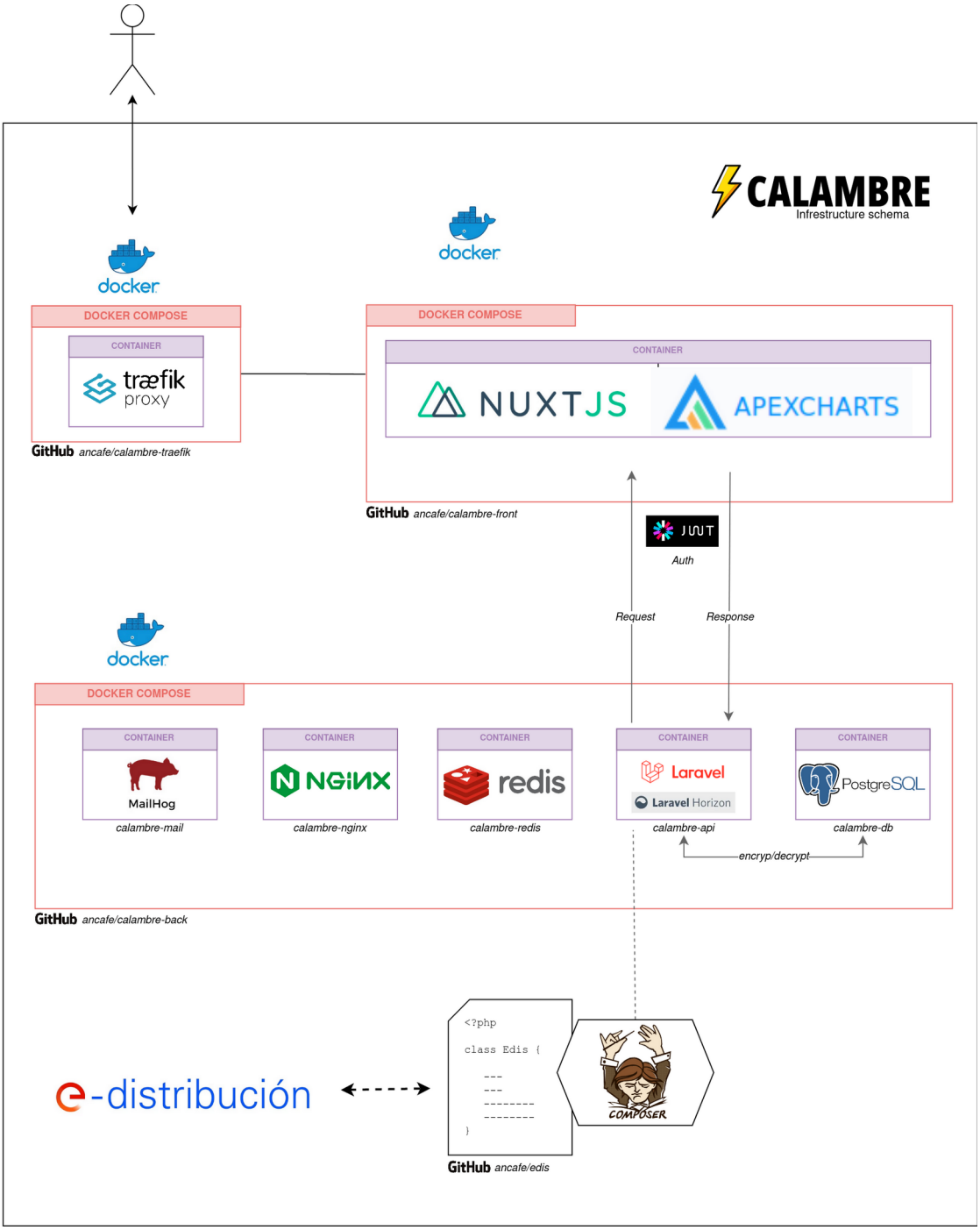


Figura 8: Diagrama de infrestructura

2.2.2. Requisitos de seguridad

Es uno de los puntos centrales de la aplicación y donde más esfuerzos se quiere dedicar. En un mundo donde la información sobre nuestros hábitos sirve de mercadeo habitual y donde la telemetría permite un altísima y detallada monitorización, entendemos que Calambre debe ser una aplicación que garantice seguridad y privacidad negándose a compartir ningún tipo de

información con con librerías de terceros. Es por eso que es requisito aplicar estos 5 niveles de seguridad:

1. Seguridad a nivel de encriptado de la información
2. Seguridad a nivel de comunicación entre backend y frontend
3. Seguridad a nivel de base de datos
4. Seguridad de identificador único universal
5. Seguridad de protocolo de cifrado

2.2.2.1. Seguridad a nivel de encriptado de la información

La aplicación debe cumplir que la información sensible e identificativa de un usuario esté encriptada en la base de datos. De esta forma, si la base de datos se viera comprometida en algún momento, sería ilegible la información vinculante y no serían revelados ni correos electrónicos, ni nombres, apellidos, CUPS de contadores, etc. Solo sería entendible por el ojo humano los datos relacionados con las medidas (por una cuestión de eficiencia dado el alto número de registros) pero en ningún caso una medida del contador podría ser vinculada a ningún contador, ningún CUPS ni ningún usuario porque dicha información debe estar encriptada.

Por otro lado de los diferentes algoritmos de encriptación optaremos por un cifrado simétrico[9] con claves de al menos 128 bits que hagan inviable un ataque por fuerza bruta. Optamos por el que hoy por hoy se considera uno de los algoritmos más seguros, que es AES-256.

Laravel ofrece este cifrado en su librería de Helpers, por lo que habrá que implementar un middleware que se encargue de encriptar y desencriptar de forma transparente tanto para el desarrollador como para el usuario.

Existen dos problemas asociados a esto: los algoritmos de búsquedas (una simple consulta SQL con algún condicional WHERE) sería altamente ineficiente puesto que cada cadena resultante estaría compuesta por una clave aleatoria. Se propone que para cambios de búsqueda específicos se utilice una clave de encriptado común. Esta solución no es la más segura como se demuestra en el artículo publicado por S. Arciszewski, 2017 [20]. Existen implementaciones para búsquedas de textos cifrados de forma segura y medianamente eficientes más modernas pero que agregan una importante carga de complejidad.

El segundo resto es impedir que ni siquiera una persona con acceso a la base de datos pueda desencriptar ninguna información sensible de ningún usuario. Para ello habría que buscar la forma de que cada usuario especificara en el momento del login la clave con la que quiere encriptar y desencriptar la información, nunca almacenandose en el lado del servidor, por lo

9 Entendemos por cifrado simétrico aquel que utiliza una clave compartida. Será el cifrado asimétrico el que requiera de un par de claves público-privada que añade seguridad (por el hecho de no tener que compartir la clave privada) pero también complejidad.

que el éxito del desencriptado siempre recaería sobre el usuario, impidiendo a cualquier persona a desencriptar la información, ni incluso teniendo acceso total al servidor.

2.2.2.2. Seguridad a nivel de comunicación entre backend y frontend

Se plantea el uso de JWT (JSON Web Token), desarrollado y liberado por Auth0 para comunicar de forma segura backend y frontend. El backend se encargará de hacer login del usuario, y si es exitoso, arrojará un Token encriptado en JWT que incluirá un payload con información que usará el frontend en todas sus llamadas. De esa forma, el backend, al recibir una llamada por parte del frontend sabrá si el token es válido o no y determinará el acceso a la información.

2.2.2.3. Seguridad a nivel de registros en base de datos

PostgreSQL incluye de forma nativa las políticas RLS (Row Level Security) por la que se pueden añadir políticas para que determinados registros relativos a un determinado usuario solo sean accesibles (en operaciones CRUD[10]) si el nombre de usuario con el que se mantiene la conexión activa a la base de datos es coincidente. De esta forma se impide que ningún usuario pueda acceder a datos de otros usuarios, sin menoscabo de los filtrados que a nivel de código se pueden aplicar. El reto será crear un usuario en la base de datos sin que el administrador conozca la contraseña por cada uno de los usuarios que utilicen la aplicación.

2.2.2.4. Seguridad de identificador único universal

Se propone usar identificadores únicos universales para identificar los registros de las bases de datos en vez de números enteros secuenciales e incrementales. De esta forma, obtenemos la ventaja de poder conocer el UUID de un registro antes de ser almacenado en la base de datos sin riesgo a tener identificadores duplicados, y de cara a ataques, el hecho de no tener secuencias de número naturales predecibles hace imposible poder apuntar a registros específicos por deducción.

2.2.2.5. Seguridad de protocolo de cifrado

Por obvio que resulte, se deberán utilizar conexiones seguras en el protocolo HTTP para cifrar las comunicaciones.

2.2.3. Requisitos de calidad

Se plantea que la aplicación se fundamente en principios SOLID[11] de programación y en un Linter que monitoree el código en búsqueda de expresiones susceptibles de ser mejoradas y derivadas hacia soluciones de clean-code. No llegaremos en nuestra implementación al nivel de excelencia que Robert C. Martin [22] plantea en el conocido libro Clean Code en el que se

10 CRUD, acrónimo en ingles de “Create, Read, Update and Delete”. Las acciones que generalmente cualquier Entidad suele ofrecer.

11 SOLID: Término acuñado por Robert C. Martin en 2009 [21] y que es acrónimo de *Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion.*

basa mucho de los paradigmas modernos de programación eficiente, pero intentaremos aproximarnos a los conceptos planteados. Nos apoyaremos en SonarLint para tal propósito.

Si bien es cierto que será complicado poder cumplir escrupulosamente con la totalidad de los principios SOLID, asi como con la programación dirigida por tests (TDD, Test-Driven Development) y arquitectura hexagonal para separar la lógica de negocio de la capa de infraestructura y de dominio, será tenida en cuenta a la hora de la implementación del código velando por respetar principios elementales que garanticen legibilidad, mantenibilidad, y calidad del código a futuro.

2.2.4. Requisitos de publicación y difusión

Se usará GitHub como repositorio público, con licencia de código libre para que los repositorios que conforman el proyecto puedan ser usados de forma no comercial y mejorados por parte de la comunidad.

3. Implementación

Entendemos por implementación al proceso de poner en marcha la idea programada y especificada en la lista de requisitos formales y no formales anteriormente descritos. Acompañan a cada uno de los apartados ejemplos de código

3.1. Introducción

El proyecto planteado contiene dos partes bien diferenciadas. La primera de ellas se trata de la implementación de una librería completa en PHP que sea capaz de comunicarse por llamadas HTTP de forma transparente con la empresa distribuidora. En este caso, e-distribución, que es la empresa distribuidora que afecta al territorio desde el que se encuentra quien suscribe el presente TFG, no facilita ningún tipo de API pública por lo que se hace imprescindible crear una librería personalizada con los riesgos que ello implica. Centraremos la primera parte de la implementación en la creación, prueba y publicación de esta librería que posteriormente integraremos en nuestro proyecto.

3.2. Entorno de desarrollo

Se plantea el desarrollo para poder ser ejecutado de forma sencilla en casi cualquier entorno. Para ello haremos uso de gestores de paquetes y de versionado, así como de un entorno que sea aislado e independiente del sistema operativo que se utilice. Por ello usaremos Docker como solución de virtualización que agrega una capa de abstracción y nos permite desplegar el código de forma muy sencilla

Como IDE usaremos PHPStorm[12] aunque cualquier otro IDE sería igualmente válido para tal fin. Se opta por PHPStorm por contar con licencia estudiantil a través de la UOC.

Nos apoyaremos en todo momento en Xdebug[13] para poder seguir la traza del código línea a línea

Por último nos apoyaremos en SonarLint para control de la calidad del código y Postman[14] como aplicación para testear la API de nuestro backend.


12 PHPStorm pertenece a la compañía JetBrains y ofrece una alta integración con los framworks más populares de PHP, entre ellos Laravel, y NuxtJS.

13 Xdebug es una potente extensión de PHP para poder hacer debug y traza con un alto nivel de detalle como se podría hacer en otros lenguajes de programación. Requiere de una configuración detallada y precisa para ser usado en un entorno “dockerizado” y especifica para el IDE por el que se opte.

14 Postman es un potente software para construir APIs

3.3. Librería de comunicación con la empresa distribuidora

Basándonos en la solución para Python existente en la comunidad[15], iniciamos el desarrollo para PHP y su posterior publicación en Packagist.

El principal reto de este código es hacer un Login legítimo en la plataforma de e-distribución que no vulnere de ninguna forma las políticas de acceso de la empresa. Lo haremos enlazando llamadas que van arrastrando diferentes cookies y haciendo scraping[16] del código obtenido hasta poder recibir un JSON con la respuesta por parte del servidor.

Una vez obtenida una respuesta válida, almacenamos en el servidor los datos de la sesión y el token de acceso de e-distribución que se volverá a comprobar en próximos accesos para evitar nuevos logins mientras el token esté vigente.

Teniendo acceso legítimo a nivel de código, es sencillo extraer del panel de control de e-distribución aquellas acciones que nos permite. Nos limitaremos a las relacionadas con la lectura de consumos, lectura de maxímetros, y lectura del ICP de un determinado contador para conocer el consumo instantáneo de un determinado contador.

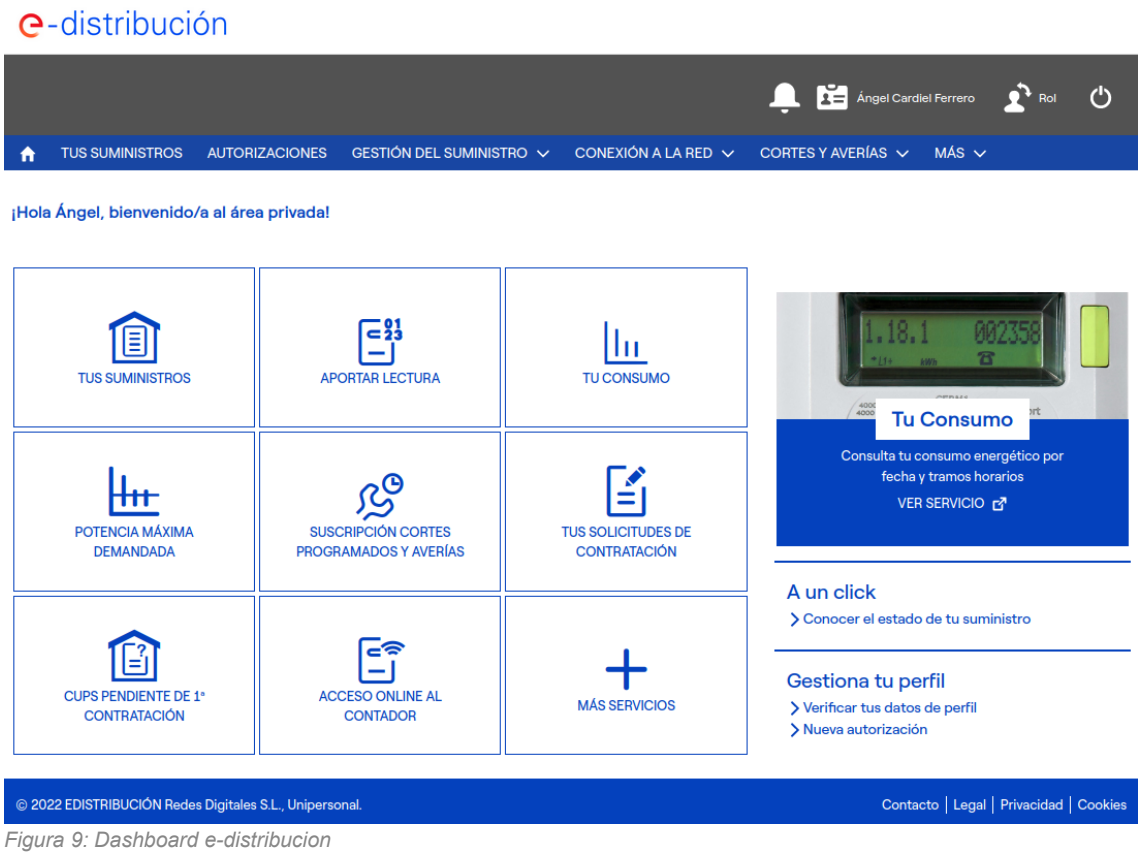


Figura 9: Dashboard e-distribucion

15 Tal y como se detalla y se cita en los agradecimientos, la solución se basa en la publicada por trocotronic para lenguaje Python [23] .

16 Será necesario usar scraping únicamente durante el proceso de login. Entendemos por scrapper el análisis de la estructura DOM de una determinada respuesta HTTP para extracción de información relevante.

Por ejemplo, para conocer cual es la acción que se hace para leer consumos por periodos, podemos usar el inspector de código de Mozilla Firefox para conocer el contenido de la llamada:

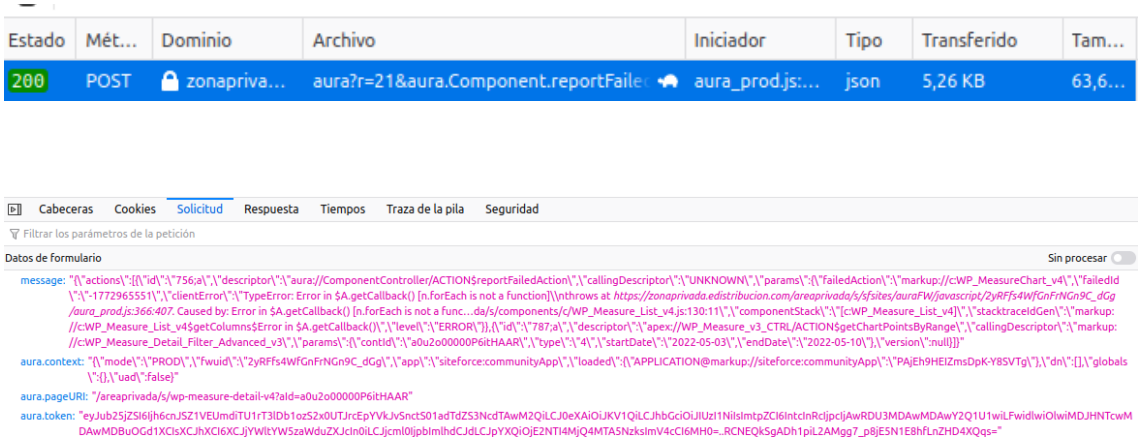


Figura 10: Análisis solicitudes e-distribución

El campo `message` es quien especifica los parámetros de la llamada, y si analizamos el JSON que se envía al servidor, vemos que la última llamada está definida de la siguiente forma:

```
{
  "id": "787;a",
  "descriptor": "apex://WP_Measure_v3_CTRL/ACTION$getChartPointsByRange",
  "callingDescriptor": "markup://c:WP_Measure_Detail_Filter_Advanced_v3",
  "params": {
    "contId": "a0u2o00000P6itHAAR",
    "type": "4",
    "startDate": "2022-05-03",
    "endDate": "2022-05-10"
  },
  "version": null
}
```

Figura 11: JSON con definición de actions para e-distribucion

Es decir, cada acción se define por un Id, un descriptor, un callingDescriptor, parámetros específicos (en este caso para leer entre fechas, el número interno de contador, el tipo de lectura, la fecha de inicio y la fecha de fin). Por tanto, podemos crear una Clase en nuestro código que nos permita hacer estas llamadas:

```php
class GetMeasInterval extends EdisActionGeneric
{
    public function __construct($params)
    {
        $id = 787;
        $descriptor = "WP_Measure_v3_CTRL/ACTION\$getChartPointsByRange";
        $callingDescriptor = "WP_Measure_Detail_Filter_Advanced_v3";
        $extras = [
            "longRunning" => true
        ];
        parent::__construct($id, $descriptor, $callingDescriptor, $params, $extras);
    }
}
```

Figura 12: Ejemplo de action en librería

```php
abstract class EdisActionGeneric
{

    private int $id;
    private string $descriptor;
    private string $callingDescriptor;
    private array $params;
    private array $extras;
    private string $command;

    public function __construct(int $id, $descriptor, $callingDescriptor, $params, $extras = [])
    {
        $this->setId($id);
        $this->setDescriptor($descriptor);
        $this->setCallingDescriptor($callingDescriptor);
        $this->params = $params;
        $this->extras = $extras;
    }

    public function __toString(): string
    {
        $data = [
            "id" => $this->getId(),
            "descriptor" => $this->getDescriptor(),
            "callingDescriptor" => $this->getCallingDescriptor(),
            "params" => $this->params
        ];

        foreach ($this->extras as $extra => $value) {
            $data[$extra] = $value;
        }

        return json_encode($data, flags: JSON_UNESCAPED_SLASHES);
    }
```

Figura 13: Réplica de la estructura esperada

De esta forma hemos construido el mismo JSON que e-distribución espera. Haciendo llamadas una vez logeados, obtenemos el siguiente resultado[17]:

17 Se protegen datos personales sensibles con el uso de asteriscos

```
{
  "id": "787;a",
  "state": "SUCCESS",
  "returnValue": {
    "data": {
      "cupsName": "ES0031********6HY0F",
      "typePM": "5",
      "startDt": "2022-05-02T22:00:00.000Z",
      "endDt": "2022-05-09T22:00:00.000Z",
      "mapParamsWS": "{\"typePM\":\"5\",\"firstCall\":false,\"taxCode\":\"\"
        ,\"endDate\":\"20220510\",\"startDate\":\"20220503\",\"pod\":\"ES0031
        ********6HY0F\"}",
      "totalValue": "63,164",
      "maxPerMonth": "9.674000000000001",
      "mapHourlyPoints": {
        "04-05-2022": [
          {
            "valid": true,
            "hour": "00 - 01 h",
            "invoiced": false,
            "obtainingMethod": "R",
            "real": true,
            "value": 0.229
          },
          {
            "valid": true,
            "hour": "01 - 02 h",
            "invoiced": false,
```


Figura 14: Recepción válida de datos de consumo por intervalo

Esta estructura de respuesta la usaremos para configurar la clase Measure.

Repitiendo esta idea, implementamos las siguientes acciones:

Tabla 5: Acciones disponibles edistribucion/edistribucion

Acción	Descripción
GetAllCups	Obtiene listado de CUPS (suministros / supplies) disponibles
GetAtrDetail	Obtiene información del contrato de un CUPS
GetCupsDetail	Obtiene detalles de un CUPS
GetCupsStatus	Obtiene estado de un CUPS
GetListCups	Obtiene listado elemental de CUPS para lecturas
GetListCycles	Obtiene listado de ciclos de facturación
GetLoginInfo	Obtiene datos de usuario
GetMaximeter	Obtiene información sobre maxímetros
GetMeas	Obtiene la medida del último día disponible por día
GetMeasInterval	Obtiene la medida de un intervalo de días por horas
GetMeasure	Obtiene la medida del último día disponible por horas
GetSolicitudAtrDetail	Obtiene el estado de solicitud
ReconnectICPDetail	Solicita reconectar el ICP
ReconnectICPModal	Permite la solicitud para reconectar un ICP

Una vez implementada y testeada la aplicación se procede a subir a Github en primer lugar y luego a Packagist para poder ser introducida como dependencia en nuestro futuro proyecto:

https://packagist.org/packages/edistribucion/edistribucion

3.4. Solución API / backend

Una vez que tenemos disponible una librería que abstrae la comunicación con la empresa, pasamos a desarrollar el backend para nuestra aplicación que llamamos Calambre.

3.4.1. Docker compose y servicios

Lo primero será crear el entorno de desarrollo específico para esta parte del proyecto. La estructura del directorio será simple:

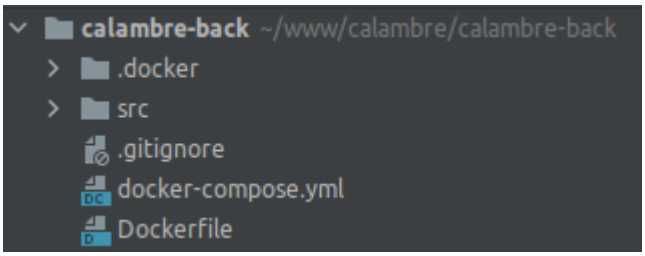


Figura 15: Estructura de directorio backend

Una carpeta `src` donde se almacena el código fuente del proyecto. Un fichero `docker-compose.yml` que define los contenedores, y una carpeta `.docker` donde se almacenan datos de configuración específicos.

En el caso del backend necesitaremos 5 servicios.

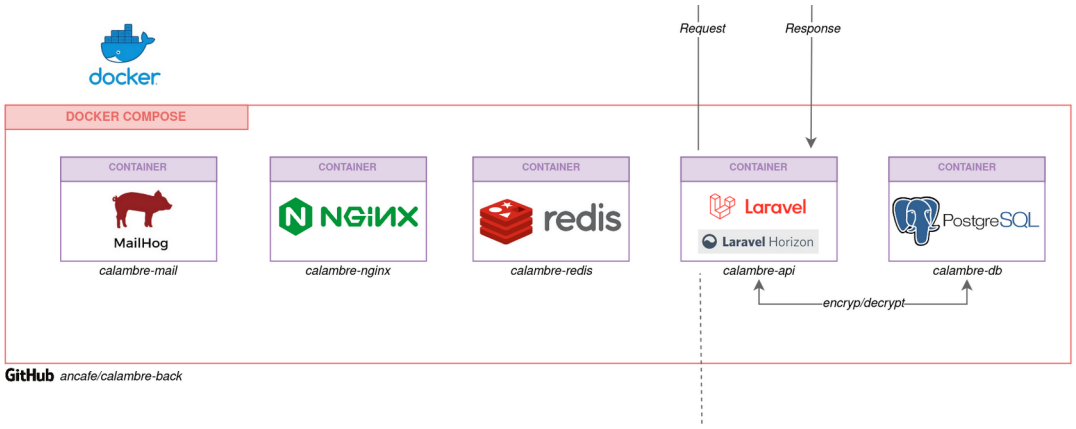


Figura 16: Servicios para backend

Sobre el servicio calambre-api, que será el principal, configuraremos los volúmenes de la forma que nos permita live-reload para que automáticamente los cambios en el código se reflejen dentro del docker, lo cual requiere una configuración específica en la definición de `docker-compose.yml` y `Dockerfile`

Además configuraremos Xdebug para poder hacer debug desde PHPStorm.

```yaml
app:
  build:
    args:
      user: 1000
      uid: 1000
    context: ./
    dockerfile: Dockerfile
  image: calambre
  container_name: calambre-api
  restart: unless-stopped
  working_dir: /var/www/
  volumes:
    - ./src:/var/www
    - ./.docker/xdebug.ini:/usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
    - ./.docker/xdebug_error_reporting.ini:/usr/local/etc/php/conf.d/error_reporting.ini
  networks:
    - calambre
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

Figura 17: Configuracion servicio API para backend

Sobre el servicio calambre-db configuraremos las credenciales por defecto y estableceremos el almacén de datos dentro de la carpeta .docker para aislarlo del sistema anfitrión, pero añadiéndo dicha carpeta al fichero .gitignore para que no sea considerada dentro del repositorio Git.

```yaml
database:
  image: postgres
  container_name: calambre-db
  restart: unless-stopped
  user: 1000:1000
  environment:
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: calambre_db
  volumes:
    - ./.docker/postgres-data:/var/lib/postgresql/data
  ports:
    - '5432:5432'
  networks:
    - calambre
```

Figura 18: Configuracion servicio PostgreSQL para backend

Los servicios faltantes se configuran con sus valores por defecto: redis para gestión de colas, nginx como servidor web, mailhog como mail-catcher.

Una vez configurado, podremos lanzar los servicios:

```
→ calambre-back git:(main) ✗ docker compose up
[+] Running 5/0
 ⠿ Container calambre-api    Created
 ⠿ Container calambre-queue  Created
 ⠿ Container calambre-mail   Created
 ⠿ Container calambre-nginx  Created
 ⠿ Container calambre-db     Created
Attaching to calambre-api, calambre-db, calambre-mail, calambre-nginx, calambre-queue
```

Figura 19: Detalle de ejecución Docker-Compose

3.4.2. Principales hitos en el desarrollo del backend

Se describen a continuación los principales hitos en el desarrollo del backend, teniendo en cuenta los requisitos formales y no formales anteriormente definidos y basándonos en el framework Laravel

3.4.2.1. Seguridad: Implementando Row Level Security

PostgreSQL ofrece de forma nativa la posibilidad de implementar políticas de seguridad al nivel de registros específicos. De esta forma, podemos definir un usuario en base de datos que exclusivamente pueda acceder (leer, escribir, filtrar...) entre el conjunto de registros que cumplan un criterio específico.

Por ejemplo, podríamos definir que un usuario A (`user_a`) solo pueda tener acceso a los registros de una tabla (`vehicles`) que cumplan el criterio en el que la columna `type = 'car'`. De esta forma, establecida la política, una consulta directa a la base de datos del tipo `SELECT * FROM public.vehicles` siempre filtraría los registros al criterio impuesto de la misma forma que lo haría una clausula `WHERE type = 'car'`.

La ventaja inequívoca de esta potente característica que nos ofrece PostgreSQL es que si la base de datos se viera comprometida con los datos de conexión del usuario, este solo tendría acceso a los registros a los que tuviera acceso.

Por tanto, es fácil pensar que el criterio de filtrado en el RLS lo vincularemos a los datos de conexión del usuario actual.

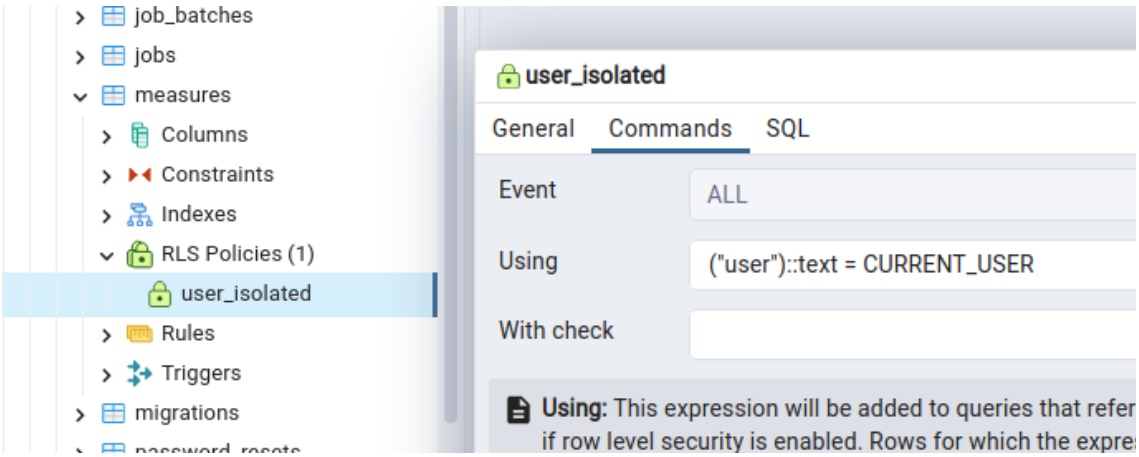


Figura 20: RLS para aislado de datos por usuario

Una vez establecida la política ahora necesitamos implementar dos soluciones adicionales:

La primera se trata de crear un usuario de conexión a la base de datos específico por cada usuario. Es decir, cuando un usuario se registra en nuestro sistema, se debe crear un usuario en la base de datos. Se usará de contraseña la establecida por el usuario en el registro, por lo que es desconocida por el administrador, profundizando en nuestra idea de que el administrador de la base de datos nunca conozca información sensible del usuario. Para

implementar esta solución, usamos el siguiente servicio de responsabilidad única (siguiendo principios SOLID):

```php
class CreateDatabaseUserWhenUserRegisteredService
{

    public function create(string $user, string $password)
    {
        $sql = '
        CREATE ROLE "' . $user. '" WITH
        LOGIN
        NOSUPERUSER
        INHERIT
        NOCREATEDB
        NOCREATEROLE
        NOREPLICATION
        PASSWORD \'' . $password. '\'
        ';

        DB::statement($sql);
        DB::statement( query: '
        GRANT "RLS_Users" TO "' . $user. '";
        ');

        Log::info( message: "Created user $user in database with password $password");
    }

}
```

Figura 21: Creación de usuario en base de datos

El siguiente paso es indicar a Laravel que queremos que una vez identificado el usuario en particular, la conexión a la base de datos la haga con datos de conexión dinámicos y cargados «al vuelo» en cada ejecución.

Para ello hacemos uso de los middleware que Laravel nos ofrece para que en el ciclo de vida de la llamada podamos capturar el momento de conexión y reemplazarlo por los datos específicos del usuario:

```php
class RLSUserMiddleware
{

    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)  $next
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     */
    public function handle(Request $request, Closure $next)
    {
        if (auth()->user()) {
            //logged --> RLS
            $userUUID = auth()->payload()->get('id');
            Config::set("database.connections.pgsql.username", $userUUID);
            Config::set("database.connections.pgsql.password", auth()->user()->getAuthPassword());
            Schema::connection( name: 'pgsql')->getConnection()->reconnect();
            return $next($request);
        }
        return $next($request);
    }
}
```

Figura 22: Middleware para modificar datos de conexión Laravel

Registramos el middleware

```php
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'jwt.verify' => \App\Http\Middleware\JwtMiddleware::class,
    'RLS' => \App\Http\Middleware\RLSUserMiddleware::class
];
```

Figura 23: Listado de middleware disponibles y registrados

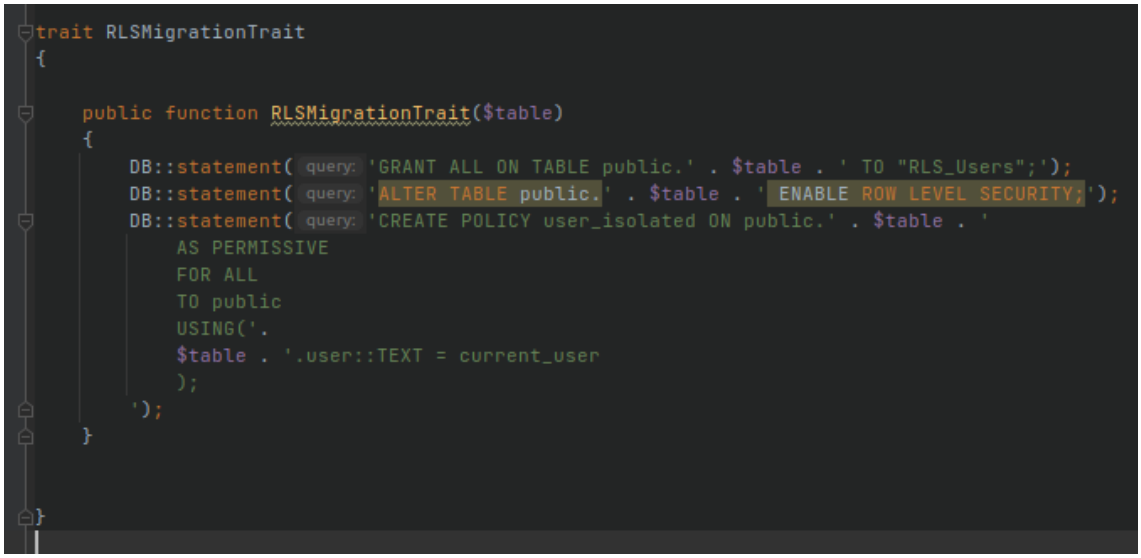
y sólo nos quedaría que para la generalidad de las rutas, se ejecute

```php
Route::group(['middleware' => ['jwt.verify', 'RLS']], function () {
    Route::get( uri: 'me',  action: \App\Http\Controllers\User\GetUserInformationController::class);
    Route::post( uri: 'logout',  action: \App\Http\Controllers\User\LogoutController::class);
    Route::prefix( prefix: '/edis')->group( callback: __DIR__ . '/API/edis.php');
    Route::prefix( prefix: '/supply')->group( callback: __DIR__ . '/API/supply.php');
    Route::prefix( prefix: '/contract')->group( callback: __DIR__ . '/API/contract.php');
});
```

Figura 24: Incoporación del middleware RLS en rutas

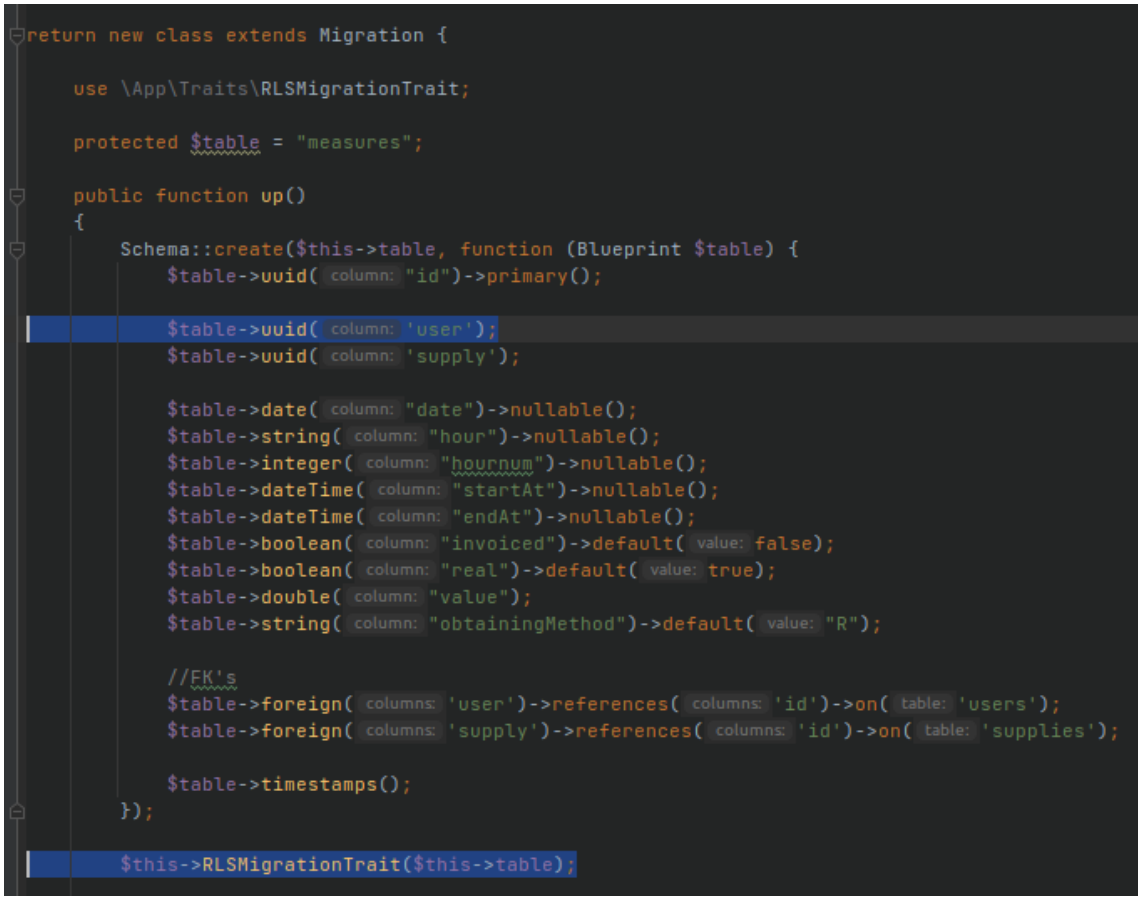
A partir de ahora, cualquier consulta que ejecute Laravel la estará haciendo en PostgreSQL usando el usuario con los filtros activados por defecto.

Debemos tener en cuenta por tanto que en todas las tablas que queramos que los registros sean específicos para los usuarios, incluya un campo ‘user’ en el que activemos la política RLS user_isolated. Para facilitar y prevenir repeticiones, creamos un Trait específico para ello:



```
trait RLSMigrationTrait
{

    public function RLSMigrationTrait($table)
    {
        DB::statement( query: 'GRANT ALL ON TABLE public.' . $table . ' TO "RLS_Users";');
        DB::statement( query: 'ALTER TABLE public.' . $table . ' ENABLE ROW LEVEL SECURITY;');
        DB::statement( query: 'CREATE POLICY user_isolated ON public.' . $table . '
            AS PERMISSIVE
            FOR ALL
            TO public
            USING('.
            $table . '.user::TEXT = current_user
            );
        ');
    }


}
```

Figura 25: Trait para migrations



```
return new class extends Migration {

    use \App\Traits\RLSMigrationTrait;

    protected $table = "measures";

    public function up()
    {
        Schema::create($this->table, function (Blueprint $table) {
            $table->uuid( column: "id")->primary();

            $table->uuid( column: 'user');
            $table->uuid( column: 'supply');

            $table->date( column: "date")->nullable();
            $table->string( column: "hour")->nullable();
            $table->integer( column: "hournum")->nullable();
            $table->dateTime( column: "startAt")->nullable();
            $table->dateTime( column: "endAt")->nullable();
            $table->boolean( column: "invoiced")->default( value: false);
            $table->boolean( column: "real")->default( value: true);
            $table->double( column: "value");
            $table->string( column: "obtainingMethod")->default( value: "R");

            //FK's
            $table->foreign( columns: 'user')->references( columns: 'id')->on( table: 'users');
            $table->foreign( columns: 'supply')->references( columns: 'id')->on( table: 'supplies');

            $table->timestamps();
        });

        $this->RLSMigrationTrait($this->table);
```

Figura 26: Estandarización de migrations para cumplir con políticas RLS

3.4.2.2. Seguridad: Encriptación de datos

El siguiente reto de seguridad es implementar encriptación de forma transparente para que los datos en la base de datos se almacenen con un algoritmo robusto de encriptación y queden desencriptado una vez Laravel arroja la respuesta.

Para ello, Laravel implementa de forma nativa un Cast[18] de tipo de datos llamado `encrypted`.

Utilizaremos el algoritmo AES-256 [25] por ser uno de los más robustos y extendidos a día de hoy. En primer lugar debemos conocer cómo funciona el algoritmo, puesto que va a ayudarnos a entender la aleatoriedad de la cadena encriptada resultante. Josh Lake ofrece un detalle amplio sobre el funcionamiento de AES-256 en su artículo “What is AES encryption and how does it work?” [26]

Debemos entender que AES-256 , que es el algoritmo que utilizaremos, es de tipo simétrico, es decir, que con una clave compartida, podremos encriptar y desencriptar cualquier datos.

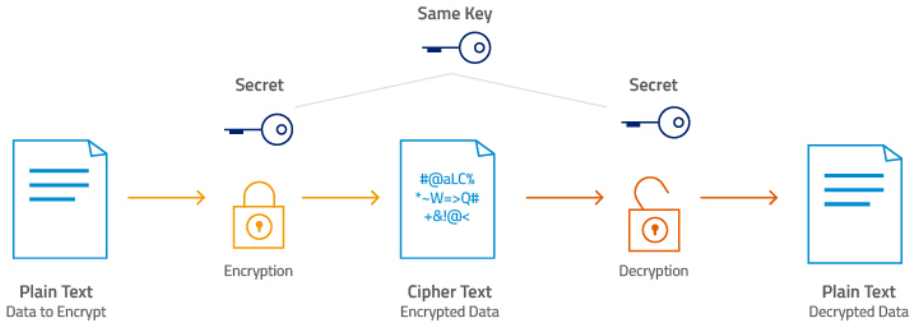


Figura 27: Esquema de encriptado síncrono

Esa clave queda definida en la configuración de entorno del framework tras ejecutar

`php artisan key:generate`

Que nos devolverá una clave única y secreta para encriptar y desencriptar los datos.

De esta forma, si la base de datos se viera comprometida, será imposible desencriptar información sin acceder también a este fichero de configuración que estará potencialmente en otro servidor y necesitaría también ser comprometido. Es decir, la base de datos por sí sola, no tiene valor para un atacante y protege toda la información sensible y relevante de los usuarios; nombre, apellidos, DNI, correo electrónico, CUPS, direcciones...

En cada modelo, a cada dato que queramos proteger, le establecemos el Cast `encrypted` y Laravel se encargará de hacerlo de forma transparente.

18 Llamamos Casting Cast o Casteo a la conversión forzada de un determinado tipo de dato. Laravel nos ofrece extensas herramientas para poder tratarlos que aparecen descritas en su documentación oficial [24]

```php
protected $casts = [
    'name' => 'encrypted',
];
```

Si miramos en la base de datos veremos la información guardada de la siguiente forma:

```
Name              |Value
------------------+------------------------------------------------------------
id                |c7eaee2e-7537-4a93-83f4-6001f78fd6b5
name              |eyJpdiI6IklkaFdOOVlaYlZHM0FUbDdsOEJyT1E9PSIsInZhbHVlIjoiNC
email             |eyJpdiI6InFWVkpnV1F4aGpHaWhwZ09qSldFUkE9PSIsInZhbHVlIjoicn
email_verified_at|
password          |$2y$10$X4g8ob20Pi8BG3WSQFA7JOAcZTbPFmpQx0rVkmDcc47qbigetQH
encrypt_validator|eyJpdiI6InFWVkpnV1F4aGpHaWhwZ09qSldFUkE9PSIsInZhbHVlIjoiKz
remember_token    |
created_at        |2022-05-17 23:45:21.000
updated_at        |2022-05-17 23:45:21.000
```

Figura 28: Ejemplo de datos almacenados en base de datos

que es ininteligible para el ojo humano y necesita de la clave de encriptación secreta que encontramos en el fichero de entorno de Laravel:

```
APP_NAME=Calambre
APP_ENV=local
APP_KEY=base64:FSqpXMJI94Hg**********Jf2P/kEw6muPa8T0=
APP_FIX_IV_FOR_EMAIL=qVVJgWQxhjGihpgOjJWERA==
```

Figura 29: Extracto de .env

3.4.2.3. Resolviendo problemas en comparativas de valores encriptados

Observamos que ocurre que si encriptamos dos veces el mismo dato con la misma clave, obtenemos resultados diferentes:

```
1000@df1ab2b822fe:/var/www$ php artisan tinker
Xdebug: [Step Debug] Could not connect to debugging client. Tried: host.docker.internal:9
003 (through xdebug.client_host/xdebug.client_port) :-(
Psy Shell v0.11.2 (PHP 8.1.5 — cli) by Justin Hileman
>>> encrypt("Hola");
=> "eyJpdiI6Ik1XN2pFbE95UGxkZEFiUzcyRnFBZ1E9PSIsInZhbHVlIjoiTXpIQ29Tc2xML0dpMlVFWGNsRWNyc
z09IiwibWFjIjoiMzdmNTBmMGJhODE4ZjY3ZTAzZGQyNTcwNzM3MDZiZjRmYWJiNjBhNWJlMWZkOTc4Y2Y4N2JkY2
IwZmI2MTI3YiIsInRhZyI6IiJ9"
>>> encrypt("Hola");
=> "eyJpdiI6IndOcVdIV05iYlY0NW54QVdZN0oxUkE9PSIsInZhbHVlIjoiNXRMTHIrcVl3WlhaRmJDZW50VjRkQ
T09IiwibWFjIjoiMzI0OTYxNGEwODYyZTU3OTM5MjViYjA3MTcyODIxYmMwYTNmYzQ2MjQyMWE2NzU2OGM1OTllMD
M3MTVjYjAyZCIsInRhZyI6IiJ9"
>>>
```

Figura 30: Demostración de aleatoriedad para misma cadena

Es cierto que si desencriptamos ambas cadenas, obtendremos el mismo resultado de origen

```
>>> decrypt("eyJpdiI6Ik1XN2pFbE95UGxkZEFiUzcyRnFBZ1E9PSIsInZhbHVlIjoiTXpIQ29Tc2xML0dpMlVF
WGNsRWNydz09IiwibWFjIjoiMzdmNTBmMGJhODE4ZjY3ZTAzZGQyNTcwNzM3MDZiZjRmYWJiNjBhNWJlMWZkOTc4Y
2Y4N2JkY2IwZmI2MTI3YiIsInRhZyI6IiJ9")
=> "Hola"
>>> decrypt("eyJpdiI6IndOcVdIV05iYlY0NW54QVdZN0oxUkE9PSIsInZhbHVlIjoiNXRMTHIrcVl3WlhaRmJD
ZW50VjRkQT09IiwibWFjIjoiMzI0OTYxNGEwODYyZTU3OTM5MjViYjA3MTcyODIxYmMwYTNmYzQ2MjQyMWE2NzU2O
GM1OTllMDM3MTVjYjAyZCIsInRhZyI6IiJ9")
=> "Hola"
```

Figura 31: Demostración de obtención de dato original con dos cadenas diferentes

Pero claro, esto nos genera un enorme problema para comparar datos (una simple clausula WHERE sobre datos encriptados no sería aplicable) puesto que para un mismo dato existen infinitas cadenas de encriptado.

¿Por qué ocurre esto? Por el uso del vector de inicialización. Definido por Bruce Schneider en 1994 [27], el vector de inicialización también es conocido por el valor IV, que es un valor aleatorio (pseudoaleatorio en términos de computación) que se inicializa en cada operación de cifrado.

Pensando por ejemplo en el proceso de Login, dado los datos:

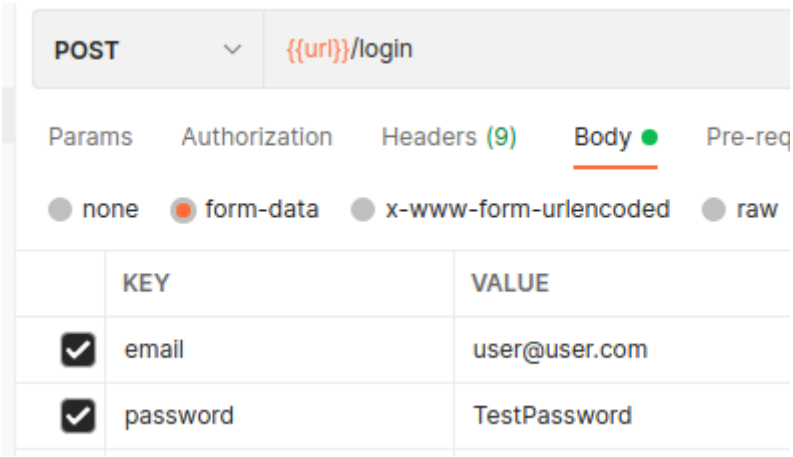


Figura 32: Ejemplo de llamada login usando Postman

Si queremos comprobar la validez de las credenciales, podríamos pensar en la siguiente consulta:

`SELECT * FROM public.users WHERE email = 'user@user.com' AND password = encrypt('TestPassword').`

Pero claro, si para un mismo dato obtenemos cadenas aleatorias de encriptado, esa consulta siempre nos devolverá un conjunto vacío de resultados y por tanto el login no será fructuoso.

Para solucionar este problema tendremos que eliminar la aleatoriedad del vector de inicialización exclusivamente para aquellos registros que necesitamos que sean comparables, o lo que es lo mismo, que sean susceptibles de ser incluidos en una cláusula WHERE.

En este caso, para la tabla `users`, el campo `email` necesitamos que su cadena de encriptación sea siempre la misma. Creamos un Cast personalizado en Laravel para tal propósito

```
protected $casts = [
    'email' => EncrypterCast::class,
    'email_verified_at' => 'datetime',
    'name' => 'encrypted',
];
```

Figura 33: Casteo de datos

Inyectamos una clase `FixEncrypter` que nos ayudará a usar un valor fijo del IV (vector de inicialización).

```
class EncrypterCast implements CastsAttributes
{
    private FixEncrypter $fixEncrypter;

    public function __construct()
    {
        $this->fixEncrypter = new FixEncrypter();
    }

    public function get($model, string $key, $value, array $attributes)
    {
        return $this->fixEncrypter->decryptString($value);
    }

    public function set($model, string $key, $value, array $attributes)
    {
        return $this->fixEncrypter->encryptString($value);
    }
}
```

Figura 34: Definición de FixEncrpter

Almacenaremos ese valor de IV fijo en nuestro fichero de configuración y lo leeremos en cada proceso de encriptación

```php
class FixEncrypter extends Encrypter
{
    private string $iv;

    public function __construct()
    {
        $this->iv = base64_decode(config( key: 'app.fix_iv_for_email'));
        $key = base64_decode(Str::after(config( key: 'app.key'), search: 'base64:'));

        parent::__construct($key, config( key: 'app.cipher'));
    }
```

Figura 35: Eliminar aleatoriedad del vector de inicialización

De esta forma, para un mismo dato, siempre obtendremos la misma cadena encriptada, que seguirá siendo segura porque seguirá necesitando de la clave de encriptado:

```
>>> $fixEncrypter = new \App\Services\FixEncrypter();
=> App\Services\FixEncrypter {#3706}
>>> $fixEncrypter->encrypt("user@user.com");
=> "eyJpdiI6InFWVkpnV1F4aGpHaWhwZ09qSldFUkE9PSIsInZhbHVlIjoiM2hqNFBkcHhoS1k2enJVRTcwTkQ0d
HRyZ2tMcWNTUXBwRmRHdkZiS2Zraz0iLCJtYWMiOiI1NWQxYTY1MzRjNzVmYTMwNmU5NDNlYWI1NDc2ZmRjODRkY2
Y5ODM3ZjgzYTk0Mzk2ZDEzZWYwYzc2YTNkMzY3In0="
>>> $fixEncrypter->encrypt("user@user.com");
=> "eyJpdiI6InFWVkpnV1F4aGpHaWhwZ09qSldFUkE9PSIsInZhbHVlIjoiM2hqNFBkcHhoS1k2enJVRTcwTkQ0d
HRyZ2tMcWNTUXBwRmRHdkZiS2Zraz0iLCJtYWMiOiI1NWQxYTY1MzRjNzVmYTMwNmU5NDNlYWI1NDc2ZmRjODRkY2
Y5ODM3ZjgzYTk0Mzk2ZDEzZWYwYzc2YTNkMzY3In0="
>>>
```

Figura 36: Demostración de cadenas idénticas usando el mismo IV

De esta forma, habremos evitado que el Login falle. Usamos la misma solución para claves de búsqueda de otros Modelos

```
'companyId' => EncrypterCast::class,                    Company.php 22
'atrId' => EncrypterCast::class,                        Contract.php 31
'visibility' => EncrypterCast::class,                   EdisInfo.php 31
'edis_id' => EncrypterCast::class,                      Supply.php 23
```

3.4.2.4. Seguridad: Reemplazo de enteros secuenciales por identificadores universales únicos

En los requisitos no formales hemos definido que la aplicación no debe usar número enteros secuenciales ascendentes como se suele utilizar de forma clásica en las bases de datos relacionales. Por ello, vamos a sustituir las claves primarias, por identificadores universales únicos, conocidos como UUID, definidos por la IETF en 2005 [28].

Laravel específicamente utiliza la versión 4 que está basada en la aleatorización. Otras versiones utilizan el reloj de la máquina, la MAC de la tarjeta de red, o los hostname.

En PostgreSQL tendremos que activar la extensión que nos permite trabajar con el tipo de dato UUID.

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

En nuestros modelos usamos el Trait UUID

```
class Supply extends Model
{
    use UUID;
```

Que nos permite usarlo para claves primarias.

```
trait UUID
{
    /**
     * Boot function from Laravel.
     */
    protected static function boot()
    {
        parent::boot();

        $creationCallback = function ($model) {
            if (empty($model->{$model->getKeyName()})) {
                $model->{$model->getKeyName()} = Str::uuid()->toString();
            }
        };

        static::creating($creationCallback);
    }

    /**
     * Get the value indicating whether the IDs are incrementing.
     *
     * @return bool
     */
    public function getIncrementing()
    {
        return false;
    }

    /**
     * Get the auto-incrementing key type.
     *
     * @return string
     */
    public function getKeyType()
    {
        return 'string';
    }
}
```

Figura 37: Trait para uso de UUID en vez de ID

3.4.2.5. Seguridad: Implementando JWT como mecanismo de autenticación

Laravel ofrece muchas capas de autenticación pero optamos por implementar JWT (JSON Web Token) como solución para comunicar backend y frontend. JWT se ha propuesto para ser convertido en un estándar y está reconocido como por la IETF[19] desde mayo de 2015 como el

19 Internet Engineering Task Force es una organización internacional abierta de normalización, con objetivo del contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, enrutamiento y seguridad.

estandar RFC7519. Es un mecanismo compacto de intercambiar información de forma segura y con verificación de integridad entre dos aplicacativos independientes. Cuenta con una estructura elemental de 3 partes: Header, donde se detalla el algoritmo de cifrado que se utiliza, el Payload donde se contienen los datos a enviar/recibir, y por último una Signature para verificar la integridad de la información y asegurarnos de que la información no ha mutado ni se ha adulterado durante el origen y el destino. Por eso entendemos que es la mejor solución para la arquitectura detallada (ver Figura 8: Diagrama de infrestructura).

Para implementar JWT usaremos una librería dedicada a ello.

Para obtener un token válido, durante el login encriptamos el email que nos llega como parámetro (ahora ya sabemos que podemos comparar cadenas encriptadas cuando estas están tipadas como FixEncrypted con un IV fijo), e intentamos la identificación.

```php
public function login(Request $request)
{
    $credentials = $request->only( keys: 'email', 'password');

    // Encrypt email with fix IV to get always the same chain
    if (array_key_exists( key: 'email', $credentials)) {
        $credentials['email'] = $this->fixEncrypter->encryptString($credentials['email']);
    }

    try {
        if (! $token = auth()->attempt($credentials)) {
            throw new ApiUnauthorizedError([ErrorDtoFactory::invalidCredentials()]);
        }
    } catch (JWTException $e) {
        throw new ApiInternalError([ErrorDtoFactory::couldNotCreateToken()]);
    }

    return $this->respondWithToken($token);
}
```

Figura 38: Login para devolver JWT válido

En caso positivo devolvemos el token.


```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXBpL2xvZ2luIiwiaWF0IjoxNjUyODY2NzE2LCJleHAiOjE2N
        I4NzAzMTYsIm5iZiI6MTY1Mjg2NjcxNiwianRpIjoiaDFOMnFTUGxZOWM4Rms3VyIsInN1YiI6ImI3NWYyNDVhLWF
        MzctNGIwNy1iMzYwLTllYWMwMDcwM2E2YSIsInBydiI6IjIzYmQ1Yzg5NDlmNjAwYWRiMzllNzAxYzQwMDg3MmRiN
        E1OTc2ZjciLCJpZCI6ImI3NWYyNDVhLWFkMzctNGIwNy1iMzYwLTllYWMwMDcwM2E2YSIsImVtYWlsIjoiYW5nZWw
        Y2FyZGllbEBwcm90b25tYWlsLmNvbSIsImVuY3J5cHRfa2V5IjpudWxsfQ.
        17_71Ycke9Eg9Z8dcLXLVdsihiWD8DsJQLNYUijJwbI",
    "token_type": "bearer",
    "expires_in": 3600
}
```


Figura 39: Ejemplo de respuesta segura tras identificación

Si analizamos el token obtenido, vemos como en el Payload podemos ver los datos que viajarán entre las URLs y que nos permitirán identificar al usuario en cuestión.

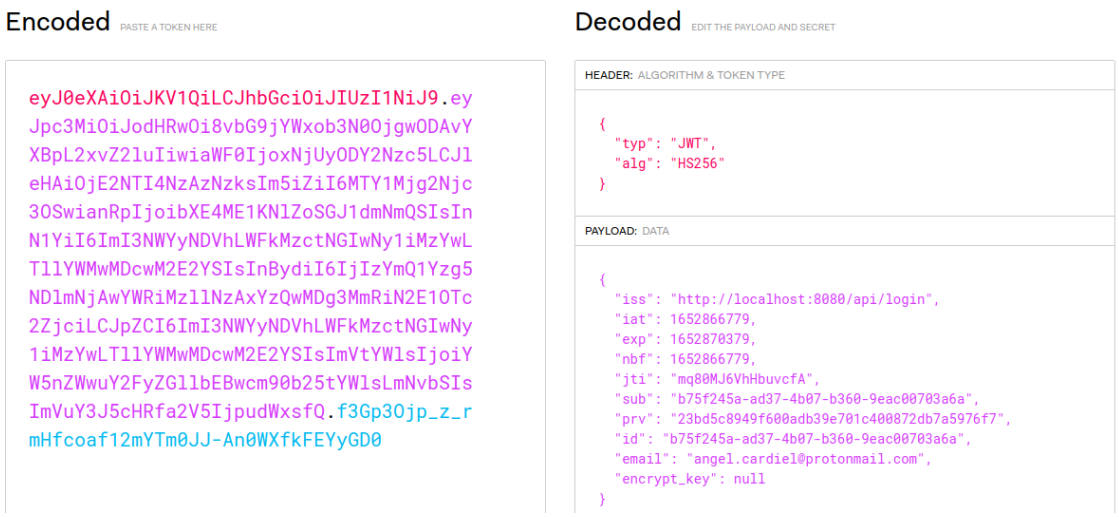


Figura 40: Análisis de estructura de JWT

Para comprobar, en una llamada entrante, que el token pertenece legítimamente a un usuario del sistema, creamos un middleware (al igual que hicimos con la implementación de RLS), para verificar que el Token recibido existe, es válido y es vigente y en ese caso cargamos los datos del usuario en cuestión. En caso contrario mostraremos una excepción.

```php
public function handle(Request $request, Closure $next)
{
    try {
        $user = JWTAuth::parseToken()->authenticate();
    } catch (Exception $e) {
        if ($e instanceof \PHPOpenSourceSaver\JWTAuth\Exceptions\TokenInvalidException){
            throw new ApiError([ErrorDtoFactory::tokenInvalid()]);
        }else if ($e instanceof \PHPOpenSourceSaver\JWTAuth\Exceptions\TokenExpiredException){
            throw new ApiError([ErrorDtoFactory::tokenExpired()]);
        }else{
            throw new ApiError([ErrorDtoFactory::tokenAbsent()]);
        }
    }
    return $next($request);
}
```

Figura 41: Extraer información del usuario identificado en las llamadas API

3.4.2.6. Seguridad: Implementando conexiones seguras a través de SSL

Por falta de tiempo y al estar en un entorno controlado y de ejecución local, se omite esta implementación. Afecta principalmente a la configuración de Docker, Traefik, y Nginx. La inversión de tiempo para trabajar con certificados self-signed es alta y este proyecto no se contempla por el momento ser puesto en ningún entorno de producción. Por tanto se plantea como futura implementación.

3.4.2.7. Colas: Organizando las llamadas a terceros

Uno de los principales problemas a los que se enfrenta esta aplicación es al hecho de que consume datos de un portal de cliente de una tercera parte sin hacer uso de una API propia

que controle el número de llamadas por segmento de tiempo, y facilite el acceso a los datos de forma controlada.

Esto tiene un debate propio en lo referido a derechos del consumidor, transparencia de las empresas distribuidoras y potenciar el software que ayude a entender y mejorar el consumo eléctrico de nuestros domicilios (o industriales).

Por tanto, corremos el riesgo de que una falta de control en estas llamadas “en bruto” que hacemos a la plataforma de e-distribución acaben por identificar nuestras consultas como un ataque si se hace de forma recursiva, o con iteraciones muy largas o si por ejemplo, en un hipotético escenario en producción, el número de usuarios creciera exponencialmente y hubiera mucha concurrencia de llamadas.

Como queremos evitar a toda costa este escenario, necesitamos controlar el número de llamadas por segmento de tiempo. La mejor forma de hacer esto es haciendo uso de colas.

Las colas son unas estructura de datos ampliamente utilizadas en ingeniería de la computación para establecer un orden FIFO[20] y evitar colapsos.

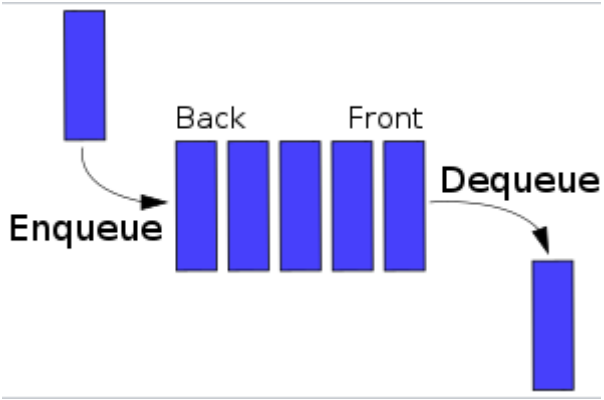


Figura 42: Cola elemental First-Input, First-Output

Laravel establece la capacidad de utilizar muchos tipos de gestión de colas (queues). Pero además de utilizar colas, necesitamos que esta utilice un throttle, que puede ser entendida como una “válvula reguladora”. Y esta opción sólo esta permitida para servidores Redis, que es la tecnología por la que optaremos.

Baśicamete el throttle nos permite definir unas políticas de velocidad y ratios por segmento de tiempo para que el procesado de la cola se ejecute exactamente a la velocidad y ritmo que marquemos. Esto es clave para evitar que nuestras peticiones puedan ser entendidas como un ataque, y nos previene igualmente de problemas en el desarrollo puesto que un bucle mal ejecutado nos puede llevar directamente a la lista negra del servidor de destino y por tanto, la aplicación no tendría de donder alimentarse de datos.

20 FIFO: First Input – First Output

Por eso esta es pieza fundamental y clave del aplicativo. Es lo que nos va a permitir, por ejemplo, poder obtener todo el histórico de datos de un determinado contador en bloques de 60 días (que es lo que e-distribución permite como máximo), sin correr ningún riesgo.

Por tanto, una lectura de contador de 1 años, 360 días, se convertirá en 7 trabajos (Jobs) encolados en nuestro gestor de colas. Estos 7 trabajos serán agrupados en un Lote (Batch)

Para entender el concepto de como implementamos esta solución, usaremos un diagrama.

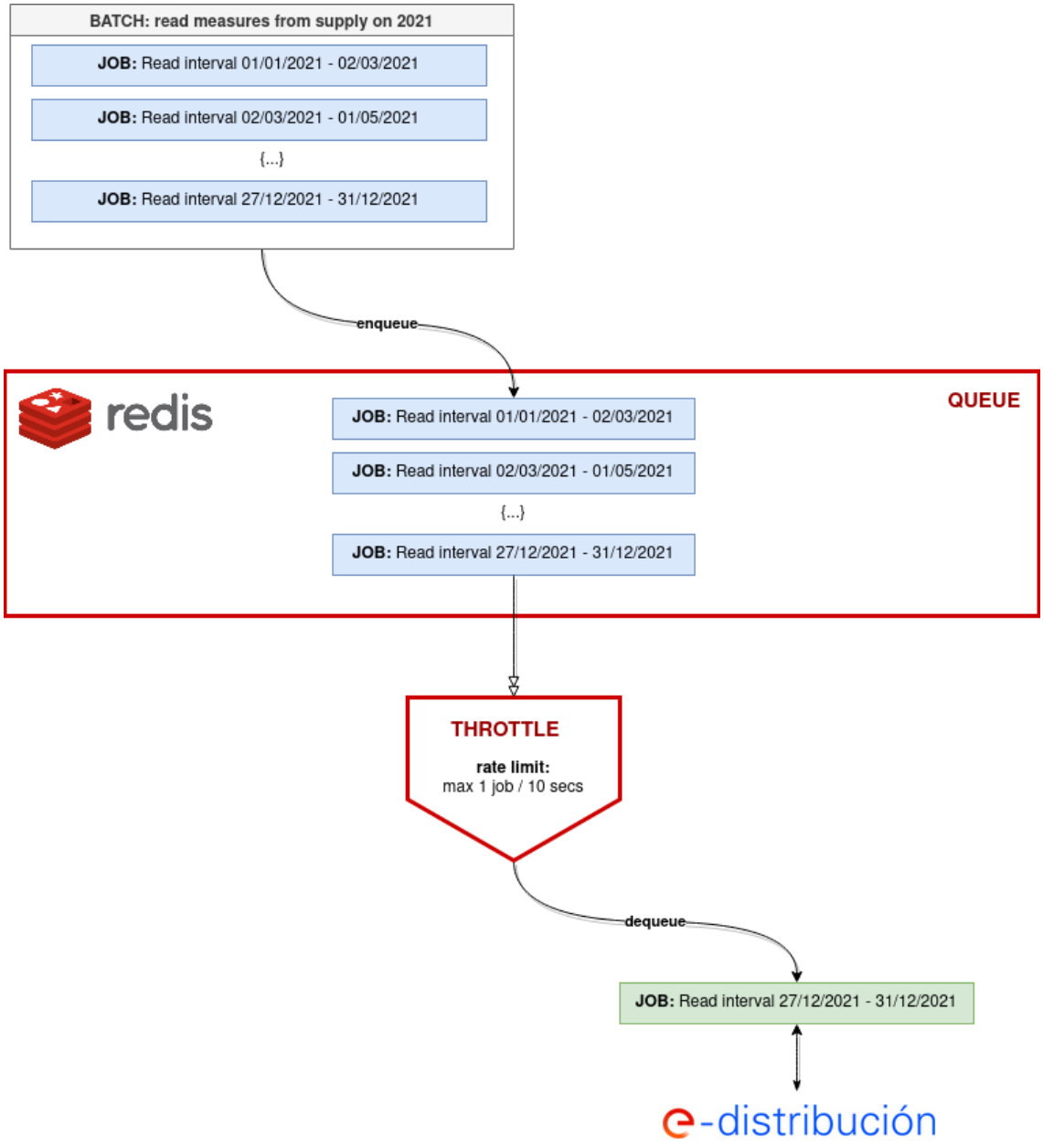


Figura 43: Diagrama de encolado y regulador de velocidad

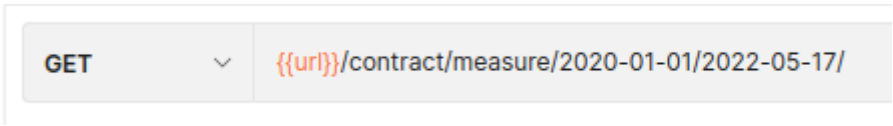
En nuestro caso usamos la clase RateLimiterForRedis para marcar esos límites de una forma excepcionalmente lenta (no conocemos los límites reales que nos marca el servidor de e-distribución y nos permite hacer un seguimiento teórico del comportamiento de la cola).

Así que configuramos nuestro throttle en para que solo pueda ejecutar 1 job por cada 10 segundos. Para obtener 1 año de consumo de 1 contador, necesitamos 7 trabajos y tardaremos alrededor de los 70-80 segundos (teniendo en cuenta las latencias de la propia llamada).

```php
class RateLimiterForEdis
{
    /**
     * Process the queued job.
     *
     * @param  mixed  $job
     * @param  callable  $next
     * @return mixed
     */
    public function handle($job, $next)
    {
        Redis::throttle( name: 'edis')
            ->block( timeout: 0)->allow( maxLocks: 1)->every( decay: 10)
            ->then(function () use ($job, $next) {
                Log::info( message: "executed job at ". now());
                $next($job);
            }, function () use ($job) {
                // Could not obtain lock...
                $job->release(30);
            });

    }
}
```

Figura 44: Limitador de llamadas a e-distribución usando throttle

Ahora ya estamos en disposición de poder encolar de forma segura Batch con el número de Jobs que necesitemos teniendo garantías de que se ejecutarán progresivamente y respentando las políticas de velocidad especificadas:




```
{
    "code": 200,
    "success": true,
    "msg": "Batch with id 9653e0a6-66bd-4afc-be8b-213f22ad9139 with 15 jobs sent to queue. It will execute in few seconds...",
    "variables": []
}
```


Figura 45: Respuesta de confirmación la inserción en la cola para lectura por intervalos

Laravel nos ofrece además un componente para visualizar el estado de una cola y hacer seguimiento detallado de las ejecuciones de cada uno de los trabajos. Se llama Laravel Horizon que lo arrancamos con el comando `php artisan horizon`

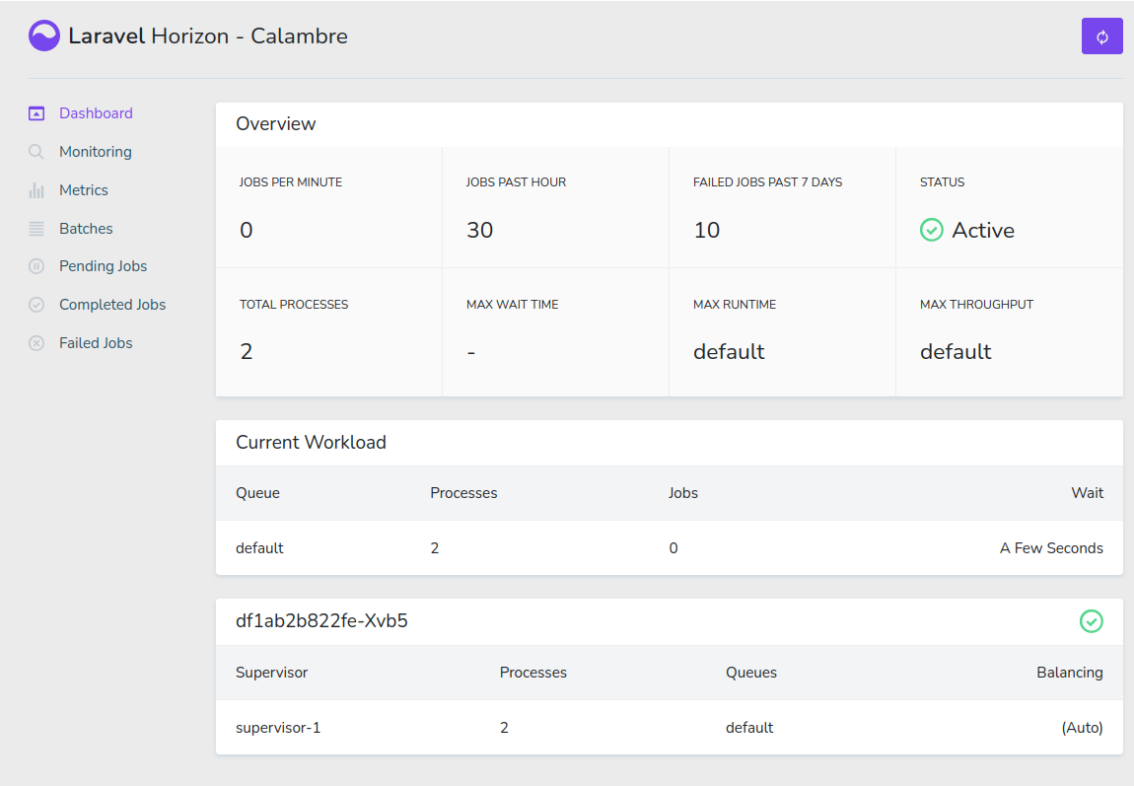


Figura 46: Dashboard de Laravel Horizon para control de colas

Y podemos ver el estado de los trabajos

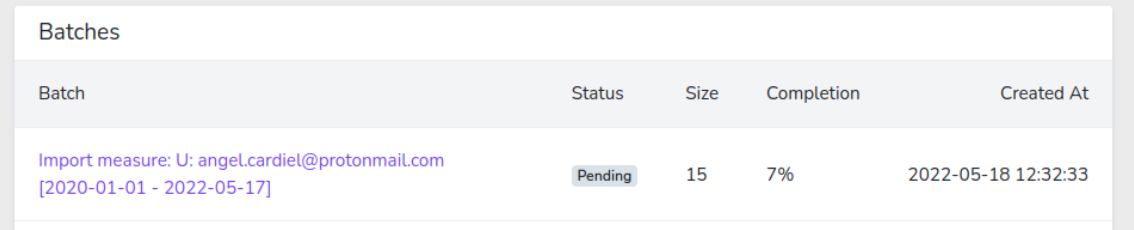


Figura 47: Visor de Lotes (Batches) de Laravel Horizon

Por tanto, y enconclusión de esta implementación, cualquier llamada a E-distribucion debemos hacerla a través de Redis encolando un trabajo (definidos en la carpeta Jobs).

3.4.2.8. Documentación: Documentación automatizada con Swagger

OpenAPI es una especificación camino de convertirse en estandar para documentar API. Nos permite utilizando la sintaxis adecuada generar documentación automática sobre las funcionalidades de una API, los parámetros de entrada y salida, los códigos HTTP de respuesta, licencias, autenticaciones, servidores y un largo etcétera.

Implementamos gracias a la librería L5-Swagger la posiblidad de usar en Laravel dicha especificación y documentar nuestros endpoints. De esta forma, además de ello, tendremos a

nuestra disposición la posibilidad de acceder a la documentación de forma interactiva con posibilidad de hacer peticiones en la API.

```
/**
 *
 * @OA\Get(
 *     path="/api/contract/{contract}/measure/{from}/{to}/",
 *     summary="Get measure interval from contract",
 *     @OA\Parameter(
 *         description="UUID of a contract",
 *         in="path",
 *         name="contract",
 *         required=true,
 *         @OA\Schema(type="string")
 *     ),
 *     @OA\Parameter(
 *         description="Date from",
 *         in="path",
 *         name="from",
 *         required=true,
 *         example="2021-01-01",
 *         @OA\Schema(type="string")
 *     ),
 *     @OA\Parameter(
 *         description="Date to",
 *         in="path",
 *         name="to",
 *         required=true,
 *         example="2021-01-15",
 *         @OA\Schema(type="string")
 *     ),
 *     @OA\Response(
 *         response=200,
 *         description="OK"
 *     ),
 *     security={{ "apiAuth": {} }}
 * )
 *
 * @throws ApiError
 * @throws \Exception
 */
```

Figura 48: Ejemplo usando annotations para definir la documentación de un endpoint

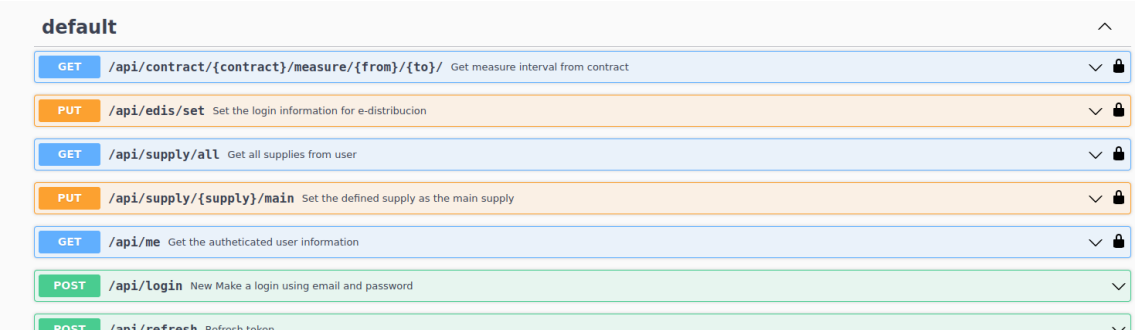


Figura 49: Ejemplo de endpoints disponibles y documentados en OpenAPI

3.4.2.9. Endpoints: Validación de rutas

Laravel cuenta con una funcionalidad para poder forzar el cumplimiento de un determinado parámetro de URL y evitar errores de tipado o incluso posibles ataques. Por ello, definimos los siguientes “data bindings” para validar que los parámetros de nuestras URLs que esperen un UUID efectivamente lo sean, así como las fechas.

```php
$patternDate = "^[0-9]{4}-(0[1-9]|1[0-2])-(0[1-9]|[1-2][0-9]|3[0-1])$";  //YYYY-mm-dd
$patternUUID = "^[0-9a-f]{8}-[0-9a-f]{4}-[0-5][0-9a-f]{3}-[089ab][0-9a-f]{3}-[0-9a-f]{12}$"; //uuid

Route::pattern( key: 'contract', $patternUUID);
Route::pattern( key: 'supply', $patternUUID);
Route::pattern( key: 'date', $patternDate);
Route::pattern( key: 'startDate', $patternDate);
Route::pattern( key: 'endDate', $patternDate);
```

Figura 50: Data bindings para validar parámetros de las URL

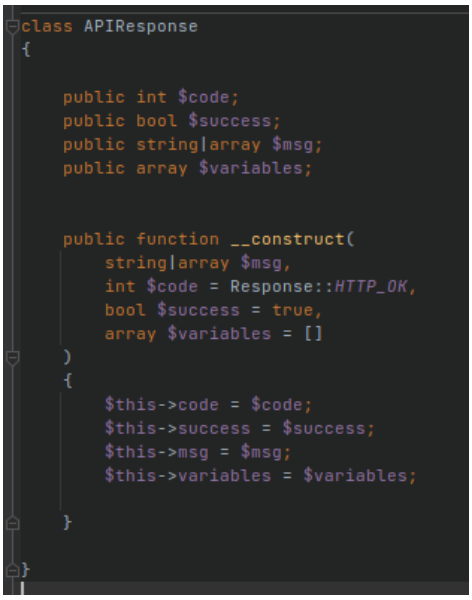
3.4.2.10. API: Sistematizando errores y respuestas

Es importante que todas las llamadas obtengan un resultado en un JSON entendible por el frontend, independientemente del código HTTP de respuesta que devuelvan. Por ello hemos creado un handler para ocuparse de que todas las llamadas tienen formato JSON. El siguiente código es para tratar las excepciones durante el tiempo de desarrollo:

```php
public function register()
{
    $this->reportable(function (Throwable $e) {
        //
    });

    $this->renderable(function (Throwable $e) {

        $msgError = $this->isJson($e->getMessage()) ? json_decode($e->getMessage()) : $e->getMessage();

        if ($e instanceof NotFoundHttpException) {
            throw new ApiNotFoundError([ErrorDtoFactory::routeNotFound()]);
        }

        if ($e->getCode() >= 100 && $e->getCode()<600) {
            return response([
                'code' => $e->getCode(),
                'success' => false,
                'msg' => $msgError
            ], status: $e->getCode() ?: 400);
        }


    });
```

Figura 51: Ante cualquier error, devolvemos JSON

Para respuestas válidas, se envuelve el contenido de la respuesta en un array que devolvemos en forma de JSON



```php
class APIResponse
{

    public int $code;
    public bool $success;
    public string|array $msg;
    public array $variables;


    public function __construct(
        string|array $msg,
        int $code = Response::HTTP_OK,
        bool $success = true,
        array $variables = []
    )
    {
        $this->code = $code;
        $this->success = $success;
        $this->msg = $msg;
        $this->variables = $variables;

    }

}
```

Figura 52: Formato estándar para respuesta con código 200

Se define una tabla de errores que sean lo más descriptivos posible para ser devueltos en excepciones durante el código.

CÓDIGO	ERROR
0-0	Fatal Error. Check ErrorDtoFactory.php
0-1	Route not found
0-1	Validation error
1-1	Bad credentials to login into e-distribucion
1-2	Something wrong during storing Edis Login information
2-1	Login error - Token Invalid
2-2	Login error - Token Expired
2-3	Login error - Authorization Token not found
2-4	The email it is already registered
2-5	Invalid credentials
2-6	Could not create valid token
2-7	User not found
3-1	No default supply founded
3-2	More than one default contract founded
3-3	No active contract founded for supply
3-4	No measure founded
3-5	Interval malformed. Negative days.
3-6	Interval out of range. Max days between dates.
4-1	Cannot connect to Redis server.

Tabla 6: Tabla de errores

3.5. Solución reactiva / frontend

Se opta por usar Nuxt.JS como librería que utiliza Webpack y VueJS entre sus dependencias porque permite una creación rápida de sitios web con tecnologías reactivas modernas.

3.5.1. Ciclo de vida de VueJS

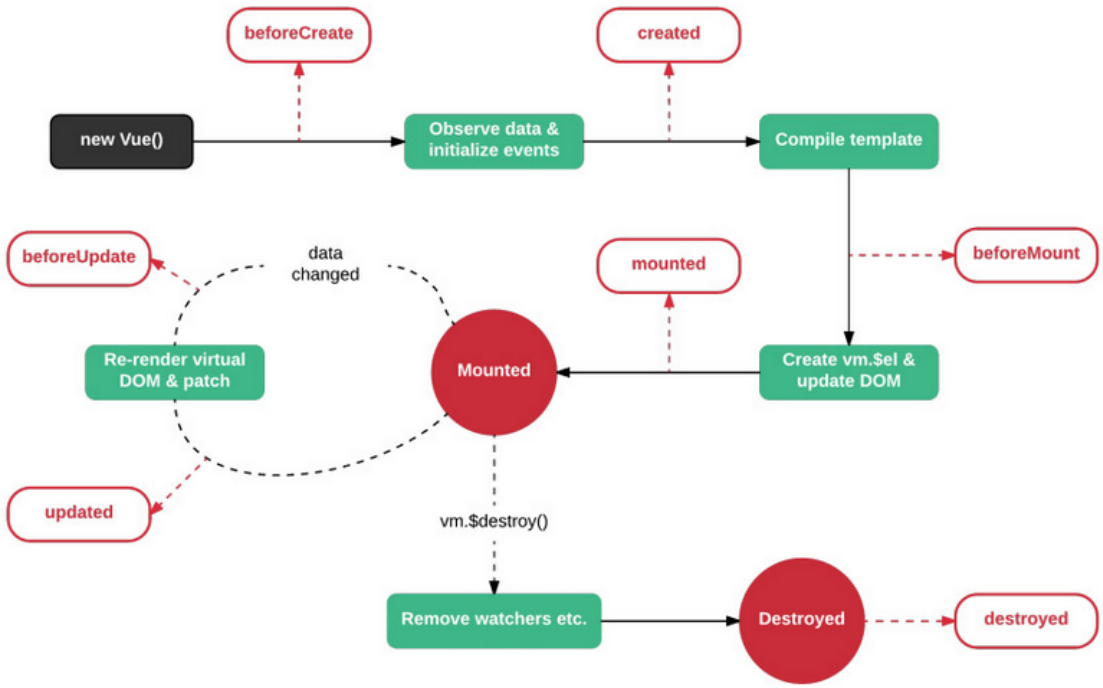


Figura 53: Ciclo de vida de un componente Vue

3.5.2. Apariencia y CSS

Para la construcción del layout y estética básica usaremos el framework Bulma CSS que permite en pocas lineas de código y con una denominación de las clases muy similar a la expresión human construir interfaces potentes y de aspecto agradable

3.5.3. Renderizado dinámico de gráficas

Para renderizar de forma dinámica las gráficas usaremos la librería ApexCharts que permite componer gráficas de alta complejidad y con un alto nivel de personalización.

Hemos configurado NuxtJS para que en todas las llamadas incluya JWT que permite autorizar e identificar al usuario en cuestión.

3.5.4. Pantalla de acceso

Una simple pantalla para validar el acceso

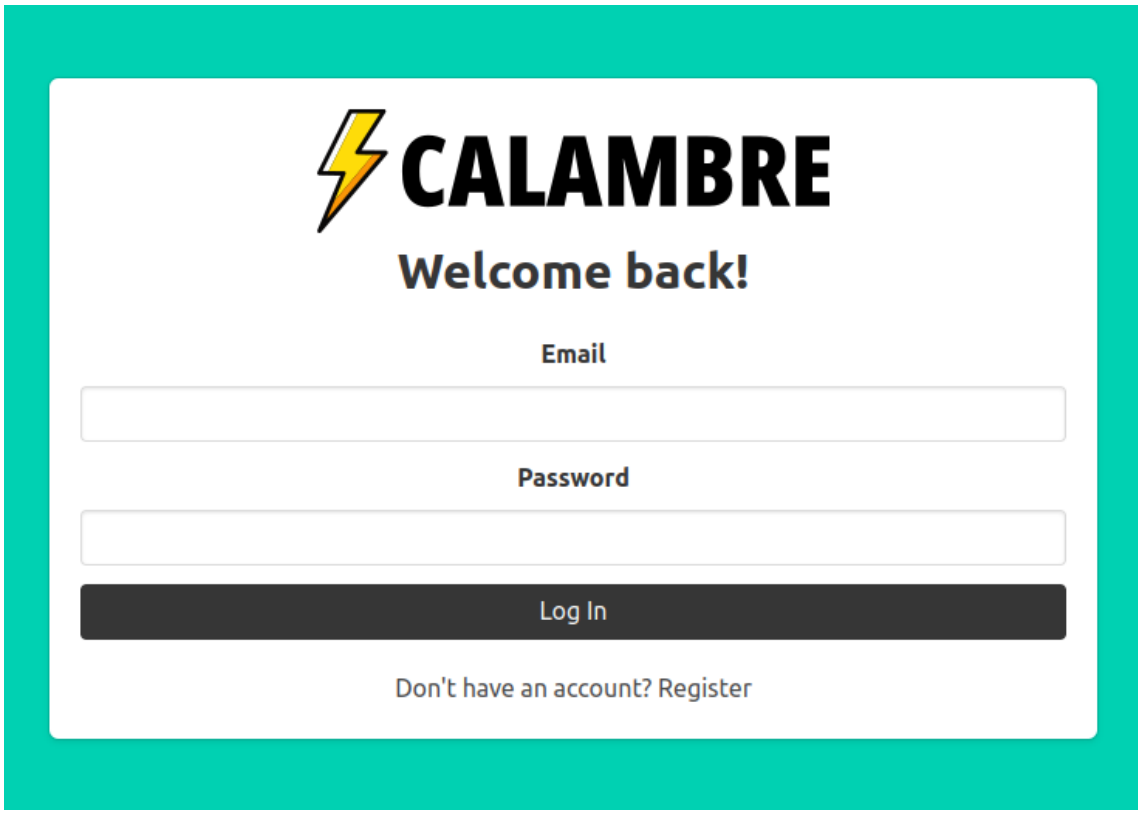


Figura 54: Pantalla de login

3.5.5. Pantalla de registro

Pantalla para registrarse como usuario

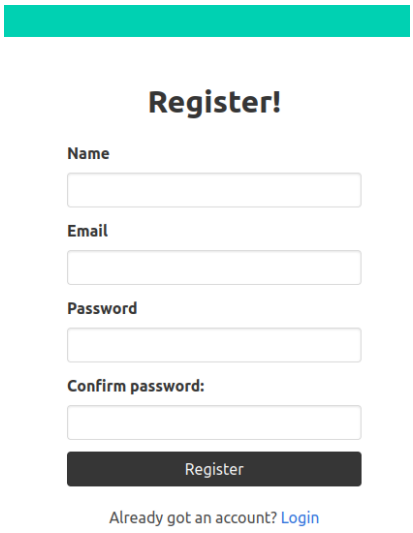


Figura 55: Pantalla de registro de nuevo usuario

Arail

3.5.6. Pantalla de suministros

Una vez el usuario es identificado en la aplicación, se muestra un listado de los suministros disponibles. Como vemos, la información está desencriptada (se encarga el backend).

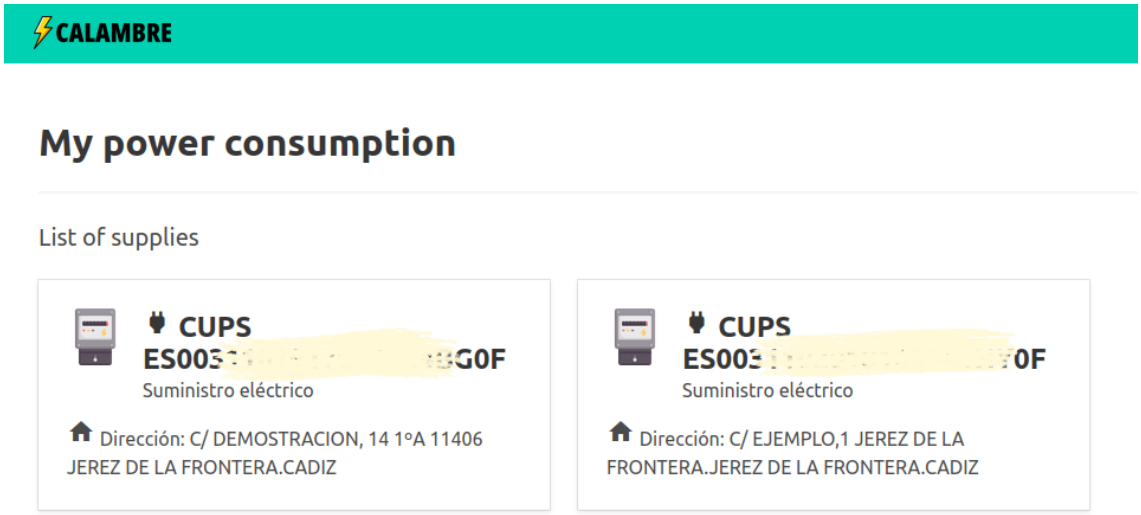


Figura 56: Pantalla principal de suministros disponibles

3.5.7. Pantalla de gráfica por horas del día

Se muestra un gráfico dinámico con un seleccionable que recarga automáticamente los datos con las horas de consumo de ese día en particular. Se clasifican las horas por los periodos que marca el gobierno y que condiciona la facturación final por parte de la comercializadora.

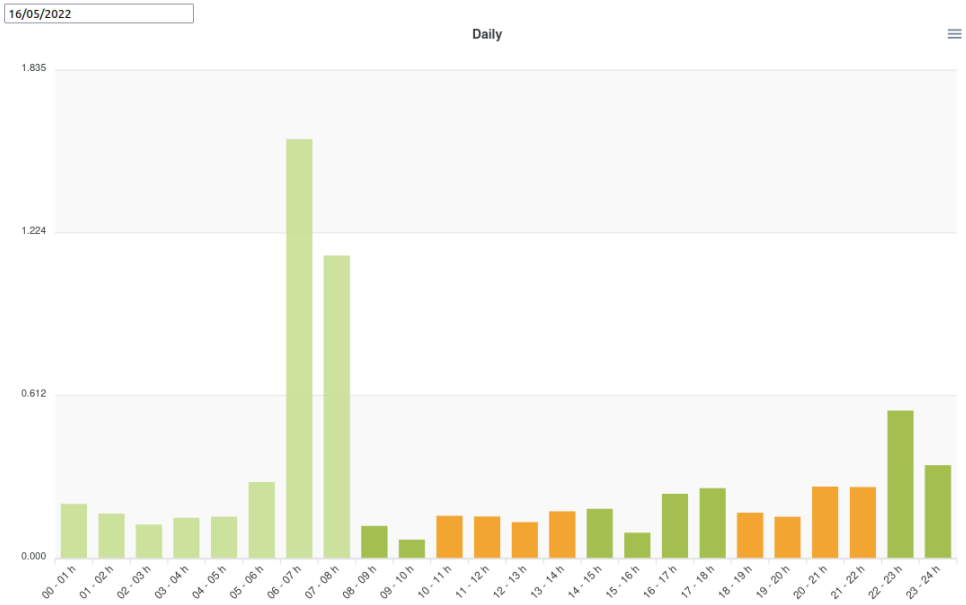


Figura 57: Vista de consumo por día

3.5.8. Pantalla de gráfica por día de la semana

Pantalla que muestra los consumos apilados por tramos y clasificados por los días de la semana de Lunes a Domingo.

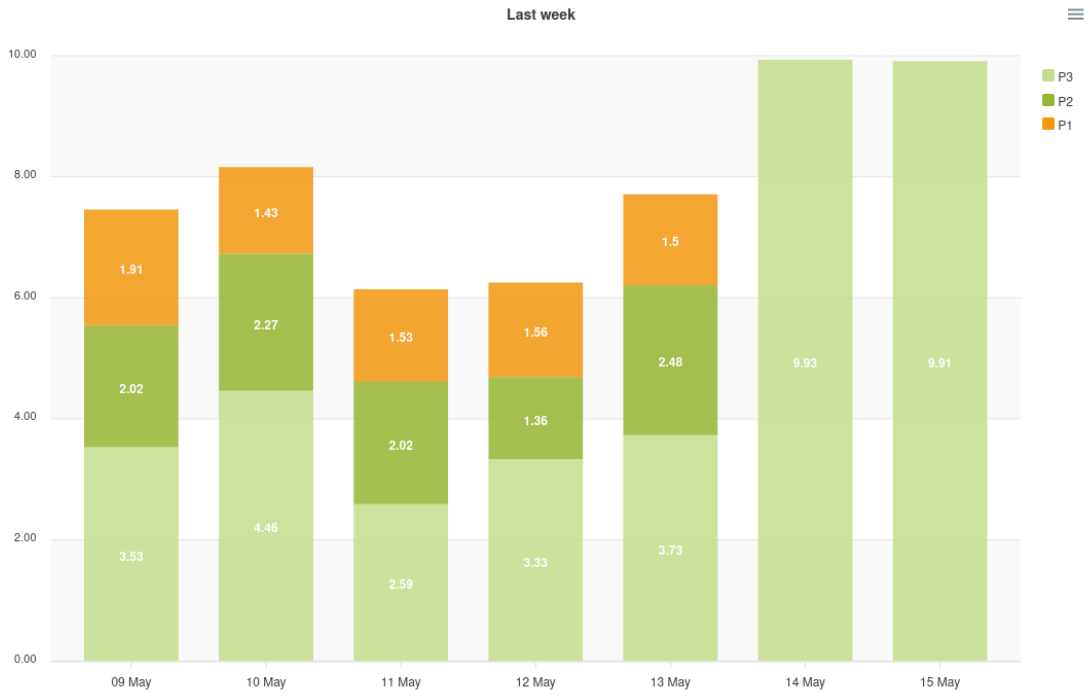


Figura 58: Agrupado de periodos por día de la semana

3.5.9. Pantalla de gráfica por meses

Podemos analizar también el consumo de nuestro contador mes a mes.

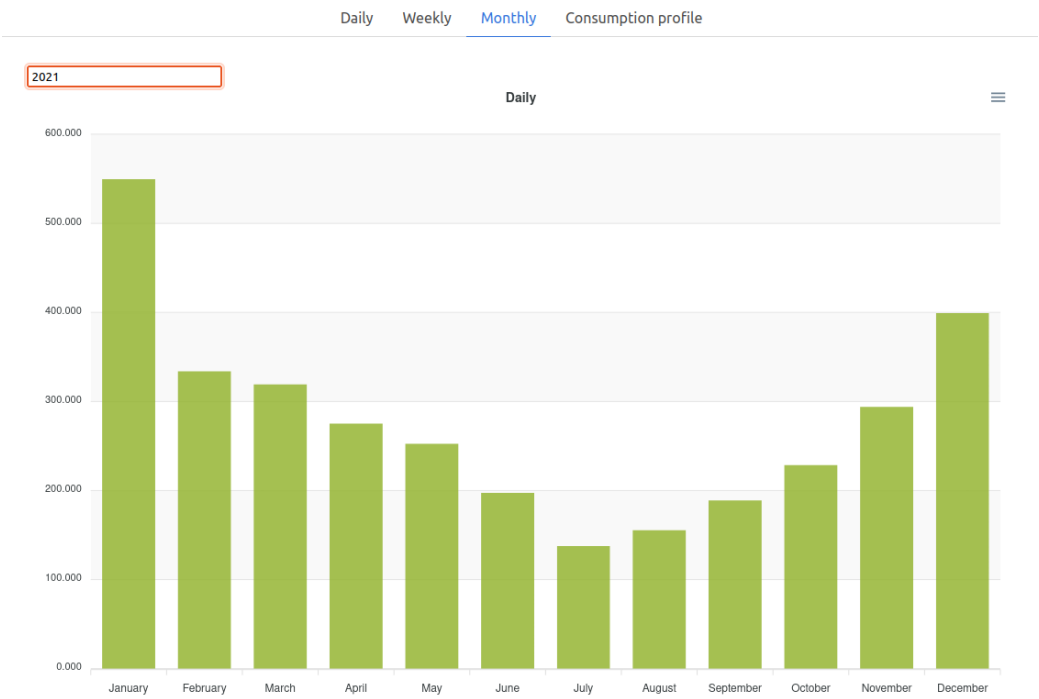


Figura 59: Vista por meses del año

3.5.10. Pantalla de perfil de consumo

Por falta de tiempo queda pendiente la implementación del siguiente gráfico circular, aunque se plantea para futuras versiones.

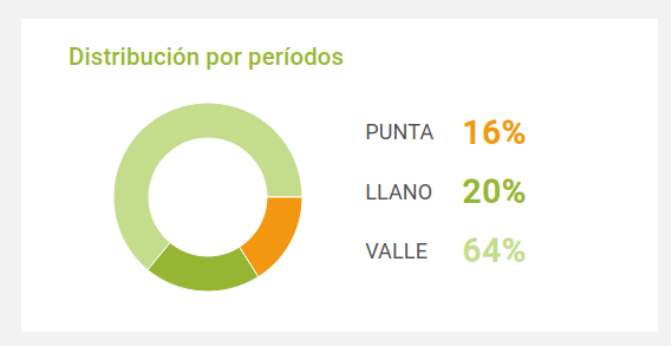


Figura 60: Perfil de consumo por periodos

4. Repositorios e instrucciones de despliegue

Cada uno de los repositorios incluye un `README.md` en formato Markdown para facilitar el procedimiento de instalación. No obstante, se hace una síntesis para integración de los respositorios.

4.1. Requisitos previos

Se parte de la base de que se cuenta con un equipo con Docker instalado. La documentación oficial es bastante completa para realizar la instalación en equipos Windows, MacOS o Linux.

4.2. Repositorios

Si bien, como se ha detallado en Breve sumario de productos obtenidos en la página 16, contamos con tres repositorios. El relativo a `Librería` no es necesario descargarlo pues forma parte del listado de dependencias automáticas que Composer instala junto al Backend.

- Librería: https://github.com/ancafe/edis
- Backend: https://github.com/ancafe/calambre-back
- Frontend: https://github.com/ancafe/calambre-front

4.3. Instalación paso a paso

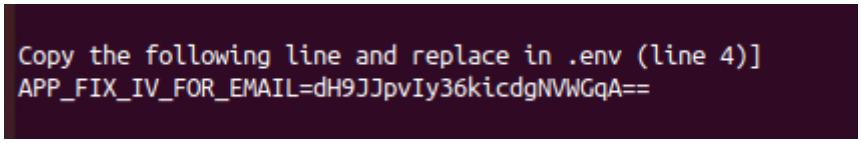
1. Ejecutar los siguientes comandos:

```
mkdir calambre && cd calambre
git clone https://github.com/ancafe/calambre-back
git clone https://github.com/ancafe/calambre-front
cd calambre-back
cp src/.env.example src/.env
mkdir .docker/postgres-data
```

2. Edita el fichero src/.env:

```
docker compose up -d –build
docker exec -it calambre-api bash
composer install
php artisan key:generate
php artisan jwt:secret
php artisan migrate –seed
clear
php artisan IV:generate
```

3. Reemplaza en el fichero `src/.env` la linea 3 por la que acaba de mostrarse en pantalla



4. Crea un usuario de prueba (opcional)

```
curl --location --request POST 'http://calambre-nginx/api/register' --header
'Authorization: Bearer null' --form 'email="admin@calambre.localhost"' --form
'name="Demo User"' --form 'password="Demo1234"' --form
'password_confirmation="Demo1234"'; echo
```

5. Sal del contenedor de backend y entra en la carpeta de frontend:

```
exit
cd ../ && cd calambre-front
```

6. Ejecuta los siguientes comandos

```
mkdir calambre && cd calambre
git clone https://github.com/ancafe/calambre-front
cd calambre-front
docker compose up
```

7. Ya podrá navegar en http://localhost:3000

5. Conclusiones

La idea principal de este proyecto fue concebida mucho antes del inicio puesto que tengo especial interés en la difusión de información clara, entendible y medible sobre el consumo energético para crear conciencia crítica y útil para entornos domésticos. En general, lo que se le quiere aportar al usuario no es simplemente la idea de ahorrar en la factura de la luz, sino por reducir el consumo (y, por tanto, la consecuencia directa se traduzca en ahorrar costes). Por eso he querido contribuir a la comunidad con la construcción de una librería dada la ausencia de una API de e-distribución para poder implementar soluciones como estas, que esperemos que llegue a futuro.

Desde luego ha supuesto un reto personal porque he procurado además utilizar paradigmas modernos de programación (aunque no siempre se ha conseguido) y tecnologías en demanda dentro del mercado laboral. Se ha pensado siempre como una solución escalable para que a futuro pudiera llegar a ser una aplicación disponible y gratuita abierta al público. Considero que actualmente no cuenta con la madurez suficiente (especialmente la relativa a frontend) como para plantearlo como SaaS, pero creo que se han asentado los cimientos para poder ir añadiendo funcionalidades que aporten valor sin interés mercantil al usuario.

Es cierto que ha habido desviaciones temporales importantes que han supuesto el no poder cubrir las expectativas iniciales. Ha sido necesario hacer renuncias importantes que han restado valor al resultado final. No obstante, considero que en términos generales se ha desarrollado un proyecto que cumple con lo planificado en la presente memoria y que se ha construido con soluciones de cierta complejidad. Creo que además se han puesto en valor conocimientos aprendidos a lo largo del Grado de Ingeniería Informática, especialmente lo relativos a:

- Estructura de datos y gestión de colas
- Algoritmos de encriptado y seguridad.
- Infraestructuras de red
- Programación orientada a objetos
- Modelado de bases de datos relacionales.
- Gestión de proyectos.

5.1. Renuncias y vistas de futuro

Como se ha comentado, el proyecto se ha planteado con una planificación temporal que ha sufrido desajustes, por lo que ha habido que realizar ciertas renuncias, inicialmente consideradas. Todas ellas son de gran valor y por tanto se plantean que puedan existir versiones posteriores a la firma de la presente memoria que mejoren e implementen las siguientes cuestiones:

5.1.1. Renuncia - Testing automático

Una de las renuncias más dolorosas de asumir ha sido la de tener que sacrificar el testing automático que aporta calidad y seguridad a todo el ciclo de vida del producto. Validaciones y asserts que no solo verificarían el correcto funcionamiento de la aplicación sino que además nos ayudan a mejorar la cobertura de código[21].

5.1.2. Renuncia – Clave de encriptado por usuario

En lo relativo al encriptado de datos, inicialmente se pensaba que cada usuario tuviera una clave exclusiva (además de su password de acceso) para cifrar y descifrar la información y que en ningún caso se almacenara en el lado del servidor. Esta renuncia viene después de varios intentos fallidos de implementación por las exigencias en número de bits de las claves (que hacía inviable ser recordado fácilmente por una persona) y porque impide la posibilidad de automatizar lecturas (con un cronjob) a menos que la clave se almacene en los payloads de los Jobs de Redis, con lo que perdería todo el sentido, porque tendríamos el dato más sensible en el lado del servidor. Sin duda agregaría una capa de seguridad mucho mayor porque nunca nadie, ni administradores, ni atacantes, ni nadie podría descifrar la información de un determinado usuario sin conocer la clave de cifrado. Existen potencialmente varias soluciones; el uso de cifrados asíncronos con pares de clave púbico-privada,, el uso de cookies del lado del cliente, etc.

Finalmente se ha optado por prescindir de esa nueva capa de seguridad y limitarnos a las existentes que consideramos que aportan valor por sí mismas.

5.1.3. Potencialidad – Notificaciones push

Utilizando la funcionalidad de Broadcasting que de forma nativa tiene Laravel (uniendo Laravel Echo y el servidor Redis como websocket), sería viable hacer un sistema de notificaciones push para que el usuario pueda recibir alertas cuando esté consumiendo más de lo “normal” (en base a sus mediciones) en un rango especialmente caro, o cuando se integren nuevos datos, etc.

5.1.4. Potencialidad – Planificador de ejecución

Laravel incluye un Scheduler de forma nativa que permitiría hacer lecturas automatizadas (por ejemplo todas las noches) de un determinado suministro. Combinado con el throttler nos aseguraríamos que no incurriríamos en ningún ataque al servidor de e-distribución. También nos permitiría conocer el estado de consumo inmediato en nuestro contador.

21 La cobertura de código es una medida (porcentual) en las pruebas de software que mide el grado en que el código fuente de un programa ha sido comprobado con pruebas de software. Sirve para determinar la calidad del test que se lleve a cabo y para determinar las partes críticas del código que no han sido comprobadas y las partes que ya lo fueron, además se puede utilizar como técnica de optimización dentro de un compilador optimizador para llevar a cabo una eliminación de código muerto, más específicamente sirve para detectar código inalcanzable

5.1.5. Potencialidad – Medir consumo de un aparato doméstico

La librería de consulta publicada dentro del presente trabajo (edistribucion/edistribucion) admite lecturas del consumo inmediato del contador. Con dos lecturas y un pequeño asistente paso-a-paso que guíe al usuario, se podría estimar el consumo de un determinado aparato eléctrico y ver qué porcentaje sobre la potencia contratada supone.

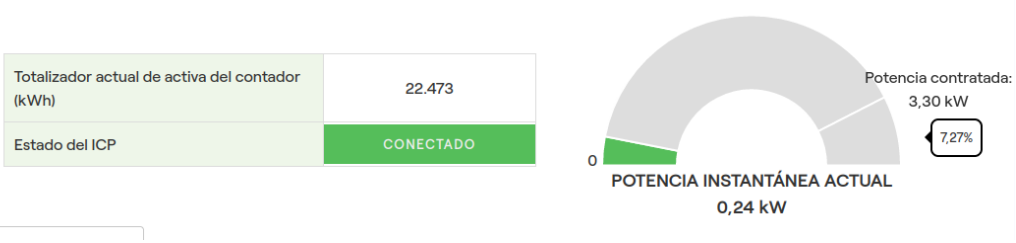


Figura 61: Ejemplo de consumo instantáneo de e-distribución

5.1.6. Potencialidad – Recomendaciones automatizadas (IA)

La idea origina del Calambre es que ayude a detectar malos patrones de consumo para mejorar y reducir el consumo, por lo que sería positivo poder analizar con inteligencia artificial determinados patrones y hacer recomendaciones personalizadas. Algunas de ejemplo:

- Has consumido más/menos de ayer
- Este mes has aumentado/disminuido tu consumo con respecto al año pasado
- Es buena hora para…. (en base a la mejor y más barata hora de consumo)
- Hoy es un buen día para …. (en base al mejor día de consumo)
- Cuidado con … (en base a las horas de mayor consumo)

5.1.7. Potencialidad – Cuantificación y tarificación

Se ha planteado solo trabajar con la magnitud del consumo, pero sería trasladable también al precio final de la factura. Eso necesita de mayor tiempo de desarrollo porque hay que introducir las diferentes tarifas, impuestos (que son variables y revisables), oferta comercial (horas 'gratis', descuento sobre consumo, etc.).

5.1.8. Potencialidad – Solución multilenguaje

Se ha utilizado el inglés como idioma vehicular, a sabiendas de que la solución es especialmente interesante para hispanoparlantes y catalonoparlantes por ser los idiomas en los que la distribuidora e-distribución opera en la actualidad.

5.1.9. Potencialidad – Abstracción sobre la comercializadora

Actualmente Calambre solo sabe comunicarse con e-distribución, pero a futuro sería potencialmente abstraíble sobre el uso de otras librerías que comunicaran con otras empresas distribuidoras del estado. (UFD, i-DE, E-Redes, Viesgo, Eléctricas de Cádiz)

6. Glosario

API: es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.

Backend: Es la parte del desarrollo web que se encarga de aplicar toda la lógica de negocio.

Clase: En programación orientada a objetos, Una clase es la descripción de un conjunto de objetos similares; consta de métodos y de datos que resumen las características comunes de dicho conjunto

CRUD: Acrónimo de "Crear, Leer, Actualizar y Borrar", que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software

CSV: Tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas

CUPS: Iniciales de Código Universal de Punto de Suministro, una clave de 20 o 22 dígitos alfanuméricos que es imprescindible para identificar un determinado punto de abastecimiento de energía

DDoS: Es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos

Encriptar/Desencriptar: Es un procedimiento que utiliza un algoritmo de cifrado con cierta clave (clave de cifrado) para transformar un mensaje, sin atender a su estructura lingüística o significado, de tal forma que sea incomprensible o, al menos, difícil de comprender a toda persona que no tenga la clave secreta

DOM: Acrónimo de Document Object Model. Es una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML

Frontend: Es la parte del desarrollo web que se encarga de generar y diseñar interfaces usables por los usuarios.

HTML: siglas en inglés de HyperText Markup Language. Define una estructura básica y un código para la definición de contenido de una página web

HTTP: Es el protocolo de comunicación que permite las transferencias de información en la World Wide Web

JSON: Es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

JWT: Es un sistema estandarizado que permite el intercambio seguro de datos entre dos dispositivos con garantía de integridad y posibilidad de ser firmado y/o cifrado.

kWh: Es una unidad de energía equivalente a 1 kilovatio (1 kW) de potencia sostenida durante 1 hora. Es la unidad de medida general para facturación del suministro eléctrico.

Librería: es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. Su fin es ser utilizada por otros programas, independientes y de forma simultánea

MAC: Es un identificador único asignado a un controlador de interfaz de red (NIC)

Middleware: Proporcionan un mecanismo conveniente para filtrar solicitudes HTTP entrantes a tu aplicación.

PHP: Es un lenguaje de programación de uso general que se adapta especialmente al desarrollo web.

PosgreSQL: Es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.

Proxy inverso: Es un tipo de servidor proxy se caracteriza por situarse en medio del enlace entre los servidores y los usuarios, lo que permite que estos últimos realicen las solicitudes web yendo al proxy de forma directa, en lugar de pasar primero por el servidor. Y este se encarga de reenviar hacia el servidor determinado

Redis: Es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes

RLS: Acrónimo de Row Level Security. En una base de datos, restringe y filtra los datos de nivel de fila de una tabla de acuerdo con las políticas de seguridad definidas por el usuario

SaaS: Es un modelo de distribución de software donde el soporte lógico y los respectivos datos que maneja se alojan en los servidores de un proveedor, cuyo acceso es a través de Internet

SQL: Es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.

TFG: En términos académicos, se considera el acrónimo Trabajo de Fin de Grado.

Token: Se refiere al proceso de sustitución de un elemento de datos sensible por un equivalente no sensible, que no tiene un significado o valor extrínseco o explotable.

Trait: Metodología de reutilización de código utilizada en lenguaje PHP

UML: Lenguaje Unificado de Modelado es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad

7. Bibliografía

- [1]: Bordera, Juan y Turiel, Antonio, El otoño de la civilización, 2022, 978-84-09-38126-5
- [2]: Dennis Meadows, Donella Meadows et Jorgen Randers, Les limites à la croissance (dans un monde fini), 2012, 978-2-37425-074-8
- [3]: Alicia Valero, “Estamos cerca de alcanzar los límites geológicos del planeta”, 2021, https://www.elsaltodiario.com/escasez-de-recursos/entrevista-minerales-materias-primas-escasez-alicia-valero-estamos-cerca-alcanzar-limites-geologicos-planeta
- [4]: Euronews, La UE desvela un plan de 210.000 millones de euros para dejar el combustible ruso, 2022, https://es.euronews.com/next/2022/05/18/ucrania-crisis-eu-energia
- [5]: La Razón, La inflación se desboca y lleva a España a la crisis de los 80, Marzo 2022, https://www.larazon.es/economia/20220331/qvscv2bmyzfa5nyaavgdxnn43q.html
- [6]: Cadena Ser, El precio de la luz fulmina su récord histórico en España: 545 euros por megavatio hora, 2022, https://cadenaser.com/2022/03/07/el-precio-de-la-luz-fulmina-su-record-historico-en-espana-545-euros-por-megavatio-hora/
- [7]: Luis Ayala, Olga Cantó, Radiografía de medio siglo de desigualdad en España, 2021, 978-84-9900-312-2
- [8]: Ministerio de Transportes, Movilidad y Agenda Urbana, Estrategia a Largo Plazo para la Rehabilitación Energética en el Sector de la Edificación en España, Junio 2020
- [9]: Comisión Nacional de Mercado y Comercio, Panel de Hogares CNMC, 2019-II
- [10]: e-distribución, APIs e-distribución, , https://www.edistribucion.com/es/servicios/APIsedistiribucion.html
- [11]: Edsger W. Dijkstra, On the role of scientific thought, 1974
- [12]: Som Energia, El servicio Infoenergia, 2022, https://es.support.somenergia.coop/article/673-el-servicio-infoenergia
- [13]: Wiegers, Karl E., Microsoft Software Requirements, 2003
- [14]: Comisión Nacional de los Mercados y la Competencia, Circular 3/2020, de 15 de enero, de la Comisión Nacional de los Mercados y la Competencia, por la que se establece la metodología para el cálculo de los peajes de transporte y distribución de electricidad. , 2020, https://www.boe.es/eli/es/cir/2020/01/15/3/con

- [15]: Rodolfo Antonio Saenz Escobar, Comparativa SQL vs NoSQL, s.f., https://openwebinars.net/blog/sql-vs-nosql-comparativa-para-elegir-correctamente/
- [16]: , Row Security Policies, s.f., https://www.postgresql.org/docs/current/ddl-rowsecurity.html
- [17]: Redis Ltd, Introduction to Redis, s.f., https://redis.io/docs/about/
- [18]: Shahzeb Ahmed, How to Set API Rate Limiting in Laravel , 2021, https://www.cloudways.com/blog/laravel-and-api-rate-limiting/
- [19]: State of JS, Front-end frameworks, 2020, https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/
- [20]: Scott Arciszewski, How to Search on Securely Encrypted Database Fields, 2017, https://paragonie.com/blog/2017/05/building-searchable-encrypted-databases-with-php-and-sql
- [21]: Robert C. Martin (Uncle Bob), Getting a SOLID start. , 2009, https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start
- [22]: Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 2008
- [23]: @trocotronic, API para e-distribución (Endesa distribución) , s.f., https://github.com/trocotronic/edistribucion
- [24]: Laravel, Eloquent: Mutators & Casting, s.f., https://laravel.com/docs/9.x/eloquent-mutators
- [25]: Federal Information Processing Standards Publication , Advanced Encryption Standard (AES), 2001, https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf
- [26]: Josh Lake, What is AES encryption and how does it work?, 2020
- [27]: Bruce Schneier, Applied Cryptography, 1994, ISBN 0-471-59756-2
- [28]: P. Leach, M. Mealling, R. Salz, A Universally Unique IDentifier (UUID) URN Namespace, 2005, https://datatracker.ietf.org/doc/html/rfc4122