Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

(Mención Computación)

Desarrollo y despliegue de un sistema de intercambio de energía entre iguales basado en Blockchain

Autor:

Álvaro Benito Navarro

Tutores:

Manuel Barrio Solórzano

Susana María Gutiérrez Caballero

Agradecimientos

Me gustaría agradecer a la Fundación CARTIF, ya que, sin ella, este proyecto no habría nacido. En concreto a mi tutora del TFG allí, Susana, por su total disposición a ayudarme, y al resto de compañeros del departamento por hacer de mi primera experiencia laboral un periodo agradable y entretenido.

A mis padres por levantarme cada vez que me caigo.

Y a mis amigos por sacarme de casa a rastras a que tomara un poco de aire del exterior.

Resumen

Este proyecto se ha realizado como Trabajo de Fin de Grado de Ingeniería Informática con mención en Computación de la Universidad de Valladolid.

El trabajo descrito en esta memoria consiste en la realización de un sistema basado en Blockchain que permita la compra y venta de energía entre participantes de una misma Microgrid. Cada participante o nodo del sistema consistirá de un cliente web, un servidor proxy, una API REST, y los componentes propios del framework utilizado, en este caso Hyperledger Sawtooth, que son el Validator y el Transaction Processor.

El cliente web se ha desarrollado utilizando HTML, CSS y JavaScript (NodeJS), mientras que el Transaction Processor está escrito en Python 3.

El proyecto puede desplegarse en cualquier sistema operativo compatible con Docker, como Windows 10 o Ubuntu 18.

Abstract

This work corresponds to the Bachelor Final Project for the Computer Science Degree at the University of Valladolid (Spain).

This document describes a Blockchain based system that permits energy interchange between participants in the same Microgrid. Each participant or node in the system will consist of a web client, a proxy server, a REST API, and the particular framework elements, in this case Hyperledger Sawtooth, which are the Validator and the Transaction Processor.

The web client has been developed using HTML, CSS and JavaScript (NodeJS), while the transaction processor is written entirely in Python 3.

The project can be deployed in any operative system compatible with Docker, like Windows 10 or Ubuntu 18.

Índice

Agradecimientos 2
Resumen 3
Abstract 4
I. Introducción 10
1. Motivación 10
2. Objetivos 11
3. Definiciones y acrónimos 12
4. Estructura del documento. 14
II. Estado del arte 15
5. Tecnologías relevantes 15
6. Trabajos relacionados 17
6.2 Trabajos sobre Microgrids 17
6.3 Intercambios de energía con Blockchain 18
III. Fundamento teórico 20
7. Blockchain 20
7.1 Hashing 21
7.2 Creación de un bloque 21
7.3 La cadena de bloques 23
7.4 Smart Contracts 23
8. Mercados de energía en Microgrids 25
8.1 Disposición de la Microgrid (C1) 25
8.2 Conexión a la red general (C2) 26
8.3 Sistema de información (C3) 26
8.4 Mecanismo de mercado (C4) 27
8.5 Mecanismo de precios (C5) 27
8.6 Sistema de administración e intercambio de energía (C6) 27
8.7 Regulación (C7) 28
IV. Proyecto software 29
9. Plan de desarrollo 29
9.1 Actividades y planificación 29
10. Modelo de negocio 34
10.1 Modelo de negocio planteado 34

10.2 Mecanismos de mercado 36
10.3 Revisión del caso de uso planteado frente a la definición de mercado Microgrid (Apartado 8) 38
11. Análisis 39
11.1 Requisitos funcionales 39
11.2 Requisitos no funcionales 39
11.3 Requisitos de información 40
11.4 Casos de uso 41
11.5 Descripción de los casos de uso 42
11.6 Modelo de dominio 42
11.6.1 Modelo de las transacciones 42
11.6.2 Modelo del estado global 43
12. Diseño 48
12.1 Arquitectura 48
12.2 Diagramas de secuencia 51
13. Despliegue 51
14. Guía de instalación 52
14.1 Instalación en un sistema Linux 53
14.2 Instalación en un sistema Windows 55
15. Manual de usuario 58
15.1 Página Sales 59
15.2 Página Create Sell 62
15.3 Página Buy Petitions 63
15.4 Página Create buy petition 65
15.5 Páginas de History of Energy bought from sales y satisfied energy petitions 66
V. Conclusiones y trabajo futuro 67
16. Conclusiones 67
17. Trabajo futuro 68
Referencias 69
VI. Anexo 70
18. Descripción de los casos de uso 70
18.1 CU01: Poner a la venta energía 70
18.2 CU02: Gestionar venta de energía 71
18.3 CU03: Modificar energía a la venta 71
18.4 CU04: Eliminar energía a la venta 72
18.5 CU05: Comprar energía a la venta 72

18.6 CU06: Realizar solicitud de energía 73
18.7 CU07: Gestionar solicitud de energía 73
18.8 CU08: Modificar solicitud de energía 74
18.9 CU09: Eliminar solicitud de energía 74
18.10 CU010: Satisfacer solicitud de energía 75
18.11 CU11: Acceder a resumen informativo de intercambios de energía 75
19. Diagramas de secuencia 76
19.1 Diagrama de secuencia CU01: Poner a la venta energía 76
19.2 Diagrama de secuencia CU02: Gestionar venta de energía 77
19.3 Diagrama de secuencia CU03: Modificar energía a la venta de energía 78
19.4 Diagrama de secuencia CU04: Eliminar energía a la venta 79
19.5 Diagrama de secuencia CU05: Comprar energía a la venta 80
19.6 Diagrama de secuencia CU06: Realizar solicitud de energía 81
19.7 Diagrama de secuencia CU07: Gestionar solicitud de energía 82
19.8 Diagrama de secuencia CU08: Modificar solicitud de energía 83
19.9 Diagrama de secuencia CU09: Eliminar solicitud de energía 84
19.10 Diagrama de secuencia CU10: Satisfacer solicitud de energía 85
19.11 Diagrama de secuencia CU11: Acceder a resumen informativo de intercambios 86
20. Diagramas de secuencia referenciados 87
20.1 Creación de batch 87
20.2 Actualización de cadena de bloques 88
20.3 Recuperación del estado global 89

Índice de figuras

Fig. 1: Ilustración esquemática de los siete componentes de los mercados en Microgrids. [3] 25
Fig. 2: Diagrama Gantt de la planificación, Parte I 32
Fig. 3 Diagrama Gantt de la planificación, Parte II 33
Fig. 4: Diagrama de casos de uso 41
Fig. 5: Modelo de dominio de Payload 42
Fig. 6: Modelo de dominio de Global State 45
Fig. 7: Arquitectura de alto nivel de una red Sawtooth (Documentación oficial de Sawtooth 2019) ... 48
Fig. 8: Diagrama de arquitectura de alto nivel de la aplicación 50
Fig. 9: Comando para obtener la dirección IP local en Linux 54
Fig. 10: Comando para obtener la dirección IP local en Windows 56
Fig. 11: Esquema de contenedores de Docker utilizando Portainer 57
Fig. 12: Esquema de contenedores de Docker utilizando el comando “docker ps” 57
Fig. 13: Página principal de la aplicación 58
Fig. 14: Menu superior de la aplicación 58
Fig. 15: Página que muestra las ventas de energía 59
Fig. 16: Modal de compra de energía 60
Fig. 17: Modal de edición / eliminación de una venta de energía 61
Fig. 18: Página de creación de venta de energía 62
Fig. 19: Página que muestra las peticiones de energía 63
Fig. 20: Modal de satisfacer una petición de energía 63
Fig. 21: Modal de edición / eliminación de petición de energía 64
Fig. 22: Página de creación de petición de energía 65
Fig. 23: Página del historial de compras de energía 66
Fig. 24: Página del historial de peticiones de energía satisfechas 66
Fig. 25: Diagrama de secuencia CU01: Poner a la venta energía 76
Fig. 26: Diagrama de secuencia CU02: Gestionar venta de energía 77
Fig. 27: Diagrama de secuencia CU03: Modificar energía a la venta 78
Fig. 28: Diagrama de secuencia CU04: Eliminar energía a la venta 79
Fig. 29: Diagrama de secuencia CU05: Comprar energía a la venta 80
Fig. 30: Diagrama de secuencia CU06: Realizar solicitud de energía 81
Fig. 31: Diagrama de secuencia CU07: Gestionar solicitud de energía 82
Fig. 32: Diagrama de secuencia CU08: Modificar solicitud de energía 83
Fig. 33: Diagrama de secuencia CU09: Eliminar solicitud de energía 84

Fig. 34: Diagrama de secuencia CU10: Satisfacer solicitud de energía 85
Fig. 35: Diagrama de secuencia CU11: Acceder a resumen informativo de intercambios de energía 86
Fig. 36: Diagrama de secuencia de creación de batch 87
Fig. 37: Diagrama de secuencia de actualización de cadena de bloques 88
Fig. 38: Diagrama de secuencia de recuperación del estado global 89

Índice de cuadros

Cuadro 1: Tabla comparativa de tecnologías Blockchain 16
Cuadro 2: Tabla desglose de tareas de la planificación 30
Cuadro 3: Tabla de requisitos funcionales 39
Cuadro 4: Tabla de requisitos no funcionales 40
Cuadro 5: Tabla de requisitos de información 40
Cuadro 6: Descripción de CU01: Poner a la venta energía 70
Cuadro 7: Descripción de CU02: Gestionar venta de energía 71
Cuadro 8: Descripción de CU03: Modificar energía a la venta 71
Cuadro 9: Descripción de CU04: Eliminar energía a la venta 72
Cuadro 10: Descripción de CU05: Comprar energía a la venta 72
Cuadro 11: Descripción de CU06: Realizar solicitud de energía 73
Cuadro 12: Descripción de CU07: Gestionar solicitud de energía 73
Cuadro 13: Descripción de CU08: Modificar solicitud de energía 74
Cuadro 14: Descripción de CU09: Eliminar solicitud de energía 74
Cuadro 15: Descripción de CU10: Satisfacer solicitud de energía 75
Cuadro 16: Descripción de CU11: Acceder a resumen informativo de intercambios de energía 75

I. Introducción

1. Motivación

Las tecnologías Blockchain son relativamente jóvenes, siendo el primer ejemplo aplicado el de Bitcoin en 2008. Esto significa que tienen un gran potencial esperando a ser descubierto.

Previamente a comenzar este proyecto, ya tenía un interés en las tecnologías Blockchain, principalmente por su grado de desconocimiento. Las descubrí durante mi cuarto año en la carrera, y aún después de hacerlo, no comprendía del todo cómo funcionaban. Esto hizo que mi interés aumentara e investigara más por mi cuenta.

La idea de este proyecto surge de una necesidad de la división de Energías del Centro Tecnológico CARTIF, lugar donde realicé mis prácticas como becario y, al ver la oportunidad, decidí convertir esa necesidad en mi TFG.

En abril 2019 se aprobó el Real Decreto que habilita la figura del autoconsumo colectivo de energía en España [9]. Esto abre la puerta a que comunidades con consumidores y productores de energía puedan crear sus propios mercados locales para poder comprar y vender excedentes de energía junto a las ventajas que ello conlleva y trataremos más adelante.

2. Objetivos

El objetivo principal del proyecto es elaborar una aplicación basada en Blockchain que permita, a través de Smart Contracts, agilizar el comercio de energía entre particulares sin la intervención de una entidad intermediaria y manteniendo en todo momento un registro inmutable de todas las transacciones en un ledger (libro de cuentas). Por la naturaleza del Blockchain, todos los nodos, o usuarios en la red, tendrán una copia idéntica del estado actual y no podrán modificar transacciones que ya formen parte de la propia cadena, manteniendo así la inmutabilidad y seguridad frente a estafas.

Por lo tanto, la aplicación debe contar con las siguientes funcionalidades:

1. Poner energía a la venta. (Para los productores)
2. Comprar energía que esté a la venta.
3. Modificar o eliminar energía puesta a la venta por uno mismo.

Además de estos 3 pilares elementales de la aplicación, se permitirá también las funcionalidades análogas:

1. Crear una petición de energía.
2. Satisfacer una petición de energía. (Para los productores)
3. Modificar o eliminar una petición de energía creada por uno mismo.

También existirá un apartado informativo en el que se podrá ver los intercambios de energía que se han realizado, y su información correspondiente.

3. Definiciones y acrónimos

A continuación, se enumeran algunos de los acrónimos y definiciones que aparecerán a lo largo del documento:

- **API:** Application Programming Interface. Interfaz o protocolo entre dos partes diferentes de un programa con el propósito de simplificar la comunicación y la implementación del software.
- **Blockhain:** DLT formado por un conjunto de bloques que contienen registros o transacciones y están unidos usando criptografía.
- **Command-line interface:** Elemento encargado de establecer la comunicación entre un usuario y el programa a través de líneas de texto.
- **Consumer:** Participante en una Microgrid que exclusivamente consume energía, ya sea consumo propio o la producida por prosumers.
- **Cross-Origin Resource Sharing:** Política de seguridad que permite el acceso a recursos cuyo origen es distinto al servidor desde donde se realiza la petición.
- **DAG:** Directed acyclic graph, gráfico dirigido sin ciclos.
- **DLT:** Distributed Ledger Technology, Tecnología de libro de cuentas distribuido.
- **EMTS:** Energy management trading system.
- **Energy Storage System:** Sistema de almacenado de energía, por ejemplo, en baterías.
- **Global state:** En Hyperledeger Sawtooth, el estado global es una representación del estado actual en base a las transacciones que se han ido añadiendo a la cadena de bloques. Este estado está almacenado en un Merkle Tree.
- **Island mode:** Modo isla. Sistema eléctrico separado físicamente de la red general.
- **Ledger:** Libro de cuentas, o libro maestro. Es donde quedarán registradas las transacciones.
- **Merkle Tree:** Concepto de criptografía, árbol hash o árbol de Merkle, es una estructura en árbol en la que los nodos hoja están etiquetados con el hash de un bloque de datos, y los nodos no hoja están etiquetados con el hash criptográfico de las etiquetas de sus nodos hijos.
- **Microgrid:** Consiste en un grupo de cargas y recursos energéticos distribuidos interconectados con límites eléctricos claramente definidos que actúa como una única entidad controlable respecto a la red general. En nuestro caso, Microgrid hará referencia a la red de participantes consumers y prosumers que podrán intercambiar energía entre sí.
- **Nodo Sawtooth:** Entidad que contiene los elementos mínimos para ser parte funcional de una Red Sawtooth. Estos elementos mínimos son el Validator y Transaction Processors. Habrá un apartado para cada elemento de Sawtooth.
- **OPF:** Optimal power flow.
- **PBFT:** Practical Byzantine Fault Tolerance.
- **PoET:** Proof of elapsed time.
- **PoW:** Proof of work.
- **Prosumer:** Participante en una Microgrid que produce energía, comúnmente a partir de Photovoltaics (en adelante se llamará PV por simplicidad), pero también consume, ya sea consumo propio, o la energía producida por otros prosumers.
- **Proxy:** Elemento intermediario en las peticiones de recursos entre un cliente y un servidor.
- **PV:** Photovoltaic, energía eléctrica generada a partir de paneles solares.

- **Red general / Grid:** Infraestructura interconectada con el propósito de suministrar electricidad desde las comercializadoras y distribuidoras hasta los consumidores.
- **Red Sawtooth:** Conjunto de nodos Sawtooth interconectados.
- **REST API:** API web que se adhiere a las restricciones REST.
- **REST:** Representational state transfer. Es un estilo de arquitectura de software que define restricciones a la hora de crear servicios web.
- **Smart Contract:** Software que encapsula la lógica de negocio para modificar una base de datos. En Hyperledger Sawtooth esta base de datos se llama “global state”.
- **Smart Meter:** Contador de electricidad inteligente acoplado al contador tradicional de viviendas o empresas y que permite utilizar servicios en internet.
- **TP:** Transaction Processor.
- **TPS:** Transacciones por segundo.
- **UUID:** Universally unique identifier. Sistema de identificación único con 4 versiones que ofrece alternativas para dar identificadores únicos a elementos dentro de un sistema, típicamente software.

4. Estructura del documento.

El documento se estructura de la siguiente manera: El capítulo dos presenta el estado del arte, donde se describirá la situación actual del contexto de trabajo, véase las tecnologías Blockchain más relevantes, y trabajos relacionados ya sea por aplicación directa de un Distributed Ledger Technology (en adelante DLT) a una Microgrid, o ejemplos de Microgrids (mercados de intercambio de energía) en los que hay intercambio directo de energía P2P (sin uso de Blockchain). El capítulo tres contiene los fundamentos de las redes Blockchain y de los mercados de energía Microgrid (sus 7 elementos principales). El capítulo cuatro contiene toda la información relativa a la aplicación per se (análisis, diseño, diagramas, implementación, pruebas y demo), y la definición específica de los 7 elementos del apartado anterior. Finalmente, el capítulo cinco contiene las conclusiones obtenidas de este trabajo y además del trabajo futuro que nos permite este proyecto.

II. Estado del arte

5. Tecnologías relevantes

Las tecnologías Blockchain han sido un boom en los últimos años, con el ejemplo más conocido del Bitcoin. Pero no sirven solo para el minado e intercambio de criptomonedas. Gracias a las características del Blockchain (visibilidad, inmutabilidad, rendimiento, fiabilidad, etc.) se puede aplicar a un número de ámbitos como son las ya mencionadas criptomonedas, gestión de cadenas de suministros, procedencia, derechos de propiedad, finanzas, y asistencia sanitaria entre muchos otros.

La tabla Cuadro 1 tratará las características que se han considerado relevantes para elegir un Blockchain orientado a mercados en Microgrids. En esta tabla, n será el número total de nodos de la red y f el máximo número de nodos defectuosos que el sistema puede hacer frente (ya sea porque no funcionen, o porque intenten dar un resultado erróneo, como nodos adversarios).

Además de ser requisito el uso de una solución open source, junto al principio de Kerckhoffs, que manifiesta que un sistema criptográfico debe ser seguro incluso si todos sus aspectos son conocidos excepto la clave, el Cuadro 1 solo tendrá soluciones open source. Bitcoin es un protocolo especializado en el minado de bloques utilizando PoW, generación de criptomonedas y transacción de estas mismas, y además es open source. Viendo su popularidad actual y a lo largo de los años es evidente que es un protocolo potente y robusto. Sin embargo, su total especialización es también su desventaja, y no es posible su uso para crear soluciones de negocio más personalizadas como es nuestro caso.

Tanto el minado de bloques como el uso de Proof of Work (en adelante PoW) requieren de alta potencia computacional y, en consecuencia, ingentes cantidades de energía, por lo que no son adecuados para aplicaciones verdes. De esta manera se descarta la opción de usar Ethereum.

Por la naturaleza de nuestro problema, va a ser necesaria una red permisionada ya que queremos controlar qué participantes entran y participan en la red, dejando como posibles soluciones Sawtooth, Fabric y Corda.

Llegados a este punto e investigando el funcionamiento de cada una de estas tecnologías se llegó a la conclusión de que era posible usar cualquiera de ellas para dar la solución al problema. Tanto Sawtooth como Fabric forman parte del proyecto Hyperledger de Linux Foundation. Se decidió descartar Corda porque Linux Foundation ofrece un curso gratuito de introducción a las tecnologías Hyperledger, y porque los proyectos de Hyperledger tienen un desarrollo y mantenimiento más activo que Corda.

Nombre Github	Descripción y algoritmo de consenso	Permisionado	Velocidad [TPS]	Resistencia a fallos / nodos adversarios	Notas
Bitcoin https://github.com/bitcoin	Especializado en sistema de pago, PoW	No	~5	f<n/2 (poco probable)	Alto coste energético, no permite creación de Smart Contracts
Ethereum https://github.com/ethereum	PoW, en el futuro pasará a PoS	No	~20	f<n/2 (poco probable)	Máquina virtual de Ethereum (EVM) con un conjunto de instrucciones Turing completas
Hyperledger Sawtooth https://github.com/hyperledger/sawtooth-core	Smart Contracts de propósito general, PoET	Ambos	~1300	Confía en el sistema de seguridad Intel SGX, f<n/2	Funciona con familias de transacciones (grupo de operaciones permitidas en un ledger)
Hyperledger Fabric https://github.com/hyperledger/fabric	Planteado para soluciones industriales, PBFT	Sí	~3500	Alta resiliencia, flexibilidad y escalabilidad, f<n/3	Gran parte de sus componentes son modulares
Corda https://github.com/corda/corda	Sistema de notaria que valida la unicidad y la secuencia de transacciones	Sí	~6300	f<n/3	Blockchain no clásica, Hashed DAG

Cuadro 1: Tabla comparativa de tecnologías Blockchain

Finalmente, entre las 2 opciones restantes entre Sawtooth y Fabric, y tras haber completado el curso introductorio (Blockchain for Business – An Introduction to Hyperledger Technologies, impartido por la propia The Linux Floundation) y programado aplicaciones básicas en ambos frameworks, he decidido utilizar Sawtooth sobre Fabric por las siguientes razones:

- Aunque en esencia ambas tecnologías permiten hacer lo mismo, Fabric está más orientado a **aplicaciones mucho más grandes y heterogéneas**. Al permitir Membership Service Provider modular, estas aplicaciones pueden ser usadas por muchas industrias y modelos de negocio distintos que pueden ser desconocidas al principio y en cualquier momento pueden unirse a la red.
- El resto de **componentes** de Fabric también son **modulares**, enfocados a permitir una gran escalabilidad y cambios a lo largo del tiempo.
- Las **transaction families** de Sawtooth **restringen y simplifican los Smart Contracts**, lo cual los hace mucho más seguros que los contratos escritos en Fabric, que no tienen ninguna restricción.
- Sawtooth permite **transacciones en paralelo**. Aunque esto pueda verse como una única ventaja en lo que respecta a la escalabilidad, el hecho de permitir transacciones en paralelo mejora la velocidad de sistema tanto si la red es pequeña como si es grande.

6. Trabajos relacionados

En esta sección se presentarán una serie de trabajos de investigación y aplicados que ofrecen una visión y soluciones relacionadas con nuestro trabajo.

6.2 Trabajos sobre Microgrids

Una Microgrid es un grupo localizado de fuentes eléctricas y cargas, que típicamente funcionan conectadas a la red general, pero pueden desconectarse y mantenerse funcionando autónomamente dependiendo de las condiciones físicas y/o económicas, como es el caso de las Microgrid en island mode.

Ejemplos de Microgrids desconectadas de la red general lo podemos encontrar en Prinsloo et al. [1], donde se trata de resolver el problema de suministrar electricidad a pequeñas poblaciones rurales en África.

Aproximadamente el 84% de la gente sin electricidad vive en áreas remotas rurales, donde la densidad de población y la demanda de energía es baja. Por eso no siempre es económico expandir la red nacional para llegar a estas zonas rurales por los costes de infraestructura y mantenimiento. Estos factores hacen que los sistemas desconectados de la red general sean una solución atractiva.

El impredecible coste de los combustibles fósiles y la reducción en costes de energías renovables han fomentado la creación de Microgrids basadas en recursos energéticos renovables.

El mayor reto en estos casos es el ineficiente sistema de gestión de la energía. La falta de conocimiento por los diseñadores del sistema sobre el uso de la energía, o un rápido incremento en el uso de la energía puede causar graves problemas. Por esto, en muchas ocasiones se combinan algoritmos de predicción del consumo y generación basados en agentes inteligentes.

Existe también una multitud de trabajos centrados en buscar el flujo de energía óptimo (OPF), como por ejemplo en Carsten et al. [2], cuyo objetivo es obtener un nivel sostenible de producción y consumo de energía. Este caso concreto se gestiona una Microgrid eléctrica y térmica, usando mercados de libro abierto para permitir a los consumidores y productores negociar sus asignaciones de energía. También proponen añadir un agente de arbitraje que decidirá cuándo conectarse a la red general (eléctrica). Adicionalmente se usarían reservas rotativas para estabilizar la Microgrid, cual baterías, permitiendo un acceso rápido de electricidad.

Pero de nuevo, este trabajo está basado en un ejemplo ficticio y su eficiencia está aún por demostrarse.

Lo que está claro es que los mercados de energía en Microgrids tienen una serie de ventajas beneficiosas tanto para las compañías generadoras de energía, las distribuidoras, como para los consumidores finales:

- Disminución de costes de transporte de energía ineficiente y con pérdidas sustanciales al satisfacerse la demanda con recursos locales.
- Se reduce el riesgo de congestión de la red principal.
- Se refuerza la comunidad local en términos de autosuficiencia y con la posibilidad de reducir costes de energía. Los beneficios locales se quedan en la comunidad, promoviendo la generación de energía renovable.

Combinando los aspectos de comunidad, junto con el enfoque de planta de energía virtual, Franke et al. [4] considera más en profundidad todas las implicaciones económicas y ecológicas de los mercados de energía en Microgrids.

6.3 Intercambios de energía con Blockchain

Sin duda, el proyecto más destacado en este aspecto sería Brooklyn Microgrid. Brooklyn Microgrid es un proyecto de demostración en el que los ciudadanos pueden comprar y vender energía producida localmente a partir de PV. El proyecto comenzó en 2015, y en abril de 2016 se realizaron las primeras transacciones de energía peer-to-peer. Actualmente los vecinos pueden ir a su app y especificar cuánto están dispuestos a pagar por energía, pudiendo encontrar así otros vecinos que produzcan energía al precio que propusieron, todo esto en un mercado simulado. El objetivo original del proyecto era conectar los barrios de Gowanus y Boerum, pero esperan crear una red que una los cinco barrios de Brooklyn por completo.

Para participar en el proyecto, simplemente hay que descargarse la app de Brooklyn Microgrid y registrarse como Prosumer o Consumer. Si el registro se realiza como Prosumer, es necesario también disponer de un Smart Meter.

El sistema de Blockchain que utilizó Brooklyn Microgrid en un principio fue Ethereum, pero al realizar pruebas y comprobar que no sería suficientemente escalable pasaron a usar Exergy, una red privada desarrollada por la empresa LO3 Energy. Lo único que se conoce sobre esta tecnología es que está desarrollada específicamente para el almacenamiento de información relacionada con la electricidad desde kWh a vars, ángulo de fase, potencia reactiva. En definitiva, una telemetría completa de la Microgrid. En el futuro pretenden dejar el proyecto en open-source cuando esté más estabilizado.

Otros casos en los que se usa Blockchain para el intercambio de electricidad, investigados por Burger et al. [5], son los de Bankymoon [11], SolarCoin [12] y BlockCharge [13].

Bankymoon es un startup desarrollada en Sudáfrica, que utiliza Bitcoin como criptomoneda para realizar pagos remotos a través de sus Smart Meters compatibles con Bitcoin, por ejemplo, en colegios. Donantes de cualquier lugar del mundo que quieran dar su apoyo pueden enviar Bitcoins directamente al Smart Meter de un colegio de su elección y proveerle de electricidad automáticamente.

El objetivo inicial de SolarCoin es utilizar unidades de energía eléctrica como respaldo económico, similar a las reservas de oro que supuestamente estabilizan monedas “reales”. Actualmente se ha convertido en un sistema de recompensa para instalaciones de generación de energía renovables.

Cualquier propietario puede registrar su instalación PV online con datos que demuestren su existencia y funcionamiento, consiguiendo así sus SolarCoins gratuitamente.

El Proyecto BlockCharge, originado en Alemania, pretende facilitar la carga de vehículos eléctricos utilizando la Blockchain de Ethereum. Los usuarios poseen un Smart Plug, que funciona como un enchufe común, pero tiene un código asociado. El usuario accede a una app instalada en su smartphone para autorizar el proceso de carga. El Smart Plug se conecta a la Blockchain, que gestiona todos los registros de la carga. El objetivo del Proyecto es ofrecer un método de autenticación global, y sistema de carga y facturación sin intermediarios. En el futuro, cuando sea posible la carga por inducción de los vehículos eléctricos, BlockCharge se hará cargo de todo el proceso de carga.

III. Fundamento teórico

7. Blockchain

Todos conocemos, o al menos, hemos oído hablar del Bitcoin, que fue el primer sistema de Blockchain conceptualizado por una persona (o grupo) bajo el pseudónimo de Satoshi Nakamoto en 2008. Pero el primer trabajo sobre una cadena de bloques protegida por criptografía fue descrito en 1991 por Stuart Haber y W. Scott Stornetta [14], quienes buscaban un sistema para almacenar timestamps de documentos que no pudiera ser manipulado. Un año más tarde, Bayer, Haber y Stornetta incorporaron el concepto de Merkle tree al diseño [15], mejorando su eficiencia.

Dependiendo del nivel de visibilidad, las Blockhain pueden ser:

- **Públicas**: La información almacenada en la Blockchain es accesible para cualquier usuario. Por ejemplo, en Bitcoin o Ethereum, cualquier usuario puede acceder y ver el contenido de cada bloque y de cada transacción. Esto ofrece una transparencia total de la que carecen los sistemas muchos sistemas centralizados.
- **Privadas**: Para ser un participante de la red necesitas una invitación, y posteriormente un proceso de validación por el operador de la red o por un protocolo claramente definido por la red. Técnicamente una red privada no está descentralizada, sino que es un ledger que funciona como una base de datos segura basada en conceptos criptográficos ya que no todos los participantes tienen los mismos nodos, y el propietario podría editar o borrar registros en la Blockchain a su gusto. Ejemplos son Bitcoin Private o Monero.
- **Permisionadas**: Este tipo de redes están en el término medio entre públicas y privadas. Se permite a cualquiera ser participante en la red, pero cada participante tiene permisos específicos para realizar las acciones de acceder y escribir información en la Blockhain. Esta opción está siendo la más utilizada por negocios y empresas para sus soluciones por la gran versatilidad. Ejemplos de redes permisionadas son Hyperledger y Corda.

Cada tipo de red Blockchain tiene sus ventajas e inconvenientes, y el decidirse por uno u otro dependerá de los requisitos del caso de uso específico.

Para comprender como funciona una red Blockchain, es necesario entender los siguientes conceptos:

- Hashing, o huella digital.
- Creación de un bloque.
- Añadir un bloque a la cadena.

7.1 Hashing

El hash de una transacción es su huella digital única que sirve para identificarla. Como transacción se puede entender cualquier cosa, desde una palabra, a una imagen, o una cantidad de dinero. El hash se obtiene a través de un algoritmo de generación de hash, habitualmente SHA256 [16]. Este algoritmo toma como entrada una serie de bytes, y devuelve otra cadena de bytes de una longitud fija, en el caso de SHA256 devuelve 32 bytes, independientemente de lo que haya tomado como entrada.

Pero al obtener una salida de menor tamaño que la entrada se está “perdiendo información”. Esto significa que es posible que dos entradas diferentes generen el mismo hash. La posibilidad es muy pequeña pero no es cero. ¿Esto causa algún problema? No. Efectivamente puede haber colisiones de hash, pero esto no significa que este sistema no sea seguro. El problema sería que se pudieran generar colisiones de hash intencionadamente, algo que requeriría años de cómputo de conseguir. Si este sistema no fuera seguro no podríamos realizar las acciones que hacemos cada día a través de internet como comprar, o acceder a nuestra cuenta bancaria.

Una vez sabemos cómo funciona el hashing, veremos cómo se aplica el concepto a las transacciones y los bloques.

7.2 Creación de un bloque

Un bloque es el conjunto mínimo de información en la Blockchain. Son contenedores de transacciones que se van añadiendo a la Blockchain y, una vez forman parte de la cadena, son inmutables.

Hay redes que generan bloques a un tiempo fijo, como por ejemplo Bitcoin (aproximadamente uno cada 10 minutos). Si se mina un bloque en menos de 10 minutos desde el último, aumentará la dificultad del minado. Si sucede lo contrario, se disminuirá la dificultad, manteniendo el equilibrio.

Otras redes generan un bloque cuando es necesario añadir una transacción a la red, por ejemplo, Hyperledger Sawtooth. En estos casos la red puede no tener nuevos bloques durante años, o crearse 10 bloques en 1 minuto.

El algoritmo de consenso determina qué nodo en la red publicará el próximo bloque en el ledger. A continuación, se describen los más comunes:

- **Proof of Work (PoW)**: Es el algoritmo de consenso de la red Bitcoin. A través de este algoritmo se decide qué usuario va a añadir el próximo bloque a la red. En PoW se requiere que un usuario forme un bloque a partir del hash del último bloque. Si el hash del nuevo bloque tiene un cierto número de ceros por la izquierda, significa que ha superado la dificultad o nonce de la red en ese momento, y ese bloque es válido. El proceso de generar este nuevo hash que supere el nonce es lo que se llama minado de bloques, y es un proceso altamente costoso computacionalmente y, en consecuencia,

energéticamente. Actualmente la recompensa por minar un bloque es de 12.5 Bitcoins, aproximadamente 78000 Euros.

- **Proof of Stake (en adelante PoS)**: Se creó en 2011 como alternativa al PoW. Actualmente Ethereum utiliza PoW, pero planean cambiar a PoS en el futuro. Los nodos que deseen minar "apuestan" o invierten criptomoneda para aumentar su probabilidad de ser seleccionados. Este algoritmo es una selección pseudoaleatoria que tiene en cuenta la cantidad que se ha apostado, cuánto tiempo lleva apostado y un factor de aleatorización. Esto quiere decir que, aunque no se tenga una máquina potente computacionalmente (un factor necesario en PoW), aún existe una pequeña posibilidad de ser el minador elegido, aunque no se invierta criptomoneda.
- **Practical Byzantine Fault Tolerance (PBFT)**: Ofrece tolerancia a fallos bizantinos. Esto significa que el sistema puede seguir operativo incluso si un porcentaje de la red falla o actúa maliciosamente. En este caso, se permite que hasta un tercio del número total de nodos se comporte de manera maliciosa. Es un sistema con líder, a diferencia de otros sistemas basados en lotería (como PoET), por lo que no puede haber forks accidentales. También requiere que todos los nodos estén conectados entre sí, lo que puede causar problemas de escalado. Es el algoritmo de consenso que usa Hyperledger Fabric.
- **Proof of Elapsed Time (en adelante PoET)**: Es el utilizado por Hyperledger Sawtooth. Cuando existe un nuevo bloque para publicar, cada nodo en la red solicita un tiempo de espera a un enclave, o una función de confianza (en este caso, el entorno de confianza es el uso de Intel SGX, un conjunto de instrucciones incluidas en los procesadores de Intel). Cada nodo recibe un tiempo de espera aleatorio, y el que obtenga el menor es seleccionado como el líder, que crea el nuevo bloque que será añadido al ledger. Como resultado de este algoritmo, un líder totalmente aleatorio es elegido, y la cantidad de poder o tipo de hardware que se posea no supone una ventaja. Usando funciones simples la red puede verificar que el ganador, es en realidad el que ha ganado, probando que tuvo el tiempo de espera más corto antes de asumir la posición de líder.

Aunque PoET posee muchas ventajas y escala muy bien, hay una desventaja, y es el problema de los forks.

Cuando se habla de forks en una red Blockchain, es lo que sucede cuando una Blockchain diverge en dos potenciales caminos. Puede haber "hard forks", que son cambios en las reglas de la Blockchain, como el que sucedió en Ethereum en 2016 para solucionar un hackeo a la organización The DAO, que habría provocado una pérdida de 50 millones de dólares. Los "soft forks" suceden cuando nodos antiguos no está actualizados con las nuevas reglas de la Blockchain.

El algoritmo PoET puede llevar a un fork cuando dos nodos "ganadores" proponen un bloque, ya que han obtenido el mismo tiempo de espera. Estos forks deben ser resueltos por los Validators que, como consecuencia, resulta en un retraso a hora de alcanzar consistencia en la red.

7.3 La cadena de bloques

El hash de un bloque se genera a partir de su contenido, es decir, las transacciones que contiene, junto con el hash del bloque anterior. De esta manera se “encadenan” los bloques y si en algún momento se intenta modificar la información de alguno de ellos, el resto de bloques consiguientes cambiarían su hash, algo que es muy fácilmente detectable. Esta es una de las características más destacables de las tecnologías Blockchain.

El nuevo bloque, después de ver cómo se crea en el apartado anterior, se añade a la cadena. Cada nodo tiene una copia de la cadena, que actualizará cada vez que se añade un bloque. Esto evita que se pueda manipular la información, ya que, si se cambia el contenido de algún bloque, se modificarán los hashes de los bloques siguientes. Esto es muy fácilmente verificable por el resto de nodos de la red, haciendo que la Blockchain modificada quede invalidada.

Tras ver el funcionamiento, podemos derivar las siguientes características distintivas de las redes Blockchain:

- **Descentralizadas**: No existe una autoridad que gobierne o ejerza control total. El contenido de la red está replicado en todos los nodos por lo que, aunque un porcentaje de nodos dejara de funcionar, la red seguirá operativa, a diferencia de los sistemas centralizados.
- **Transparencia**: Como cada nodo posee una copia de la red, puede acceder a toda su información en cualquier momento. Esto asegura que no pueda haber estadas.
- **Seguridad**: Es imposible modificar un registro ya existente sin que el resto de participantes lo vean, como hemos visto anteriormente. Entre los métodos de seguridad de Blockchain se encuentra el uso de criptografía de clave pública. Cada usuario tiene una clave pública, conocida por todos los miembros de la red, y una clave privada. Esto hace que, mientras no se pierda la clave privada, sea imposible una suplantación de identidad, y en todo momento se asegura que los registros pertenecen a quien los haya firmado (con su clave privada)
- **Velocidad y eficiencia**: Al no existir elementos centrales, las transacciones se realizan instantáneamente sin la intervención de seres humanos, a través de los Smart Contracts. Este sistema es mucho más rápido que, por ejemplo, los sistemas bancarios tradicionales donde una acción puede llevar días.

7.4 Smart Contracts

Un Smart Contract es un protocolo cuyo propósito es facilitar, verificar y asegurar el cumplimiento de un contrato. En el contexto de Blockchain, un Smart Contract es cualquier tipo de computación que se realice en el Blockchain o ledger distribuido, sin ser necesariamente algo similar al concepto de un contrato clásico.

Actualmente la inmensa mayoría de Smart Contracts se utilizan como la parte lógica de las aplicaciones Blockchain. Estos Smart Contracts varían desde los programas más sencillos hasta aplicaciones realmente complejas.

La implementación de los Smart Contracts varía respecto a la tecnología que se use. Por ejemplo, en Bitcoin se utiliza un Smart Contract para el traspaso de una criptomoneda, y lo que comprueba es:

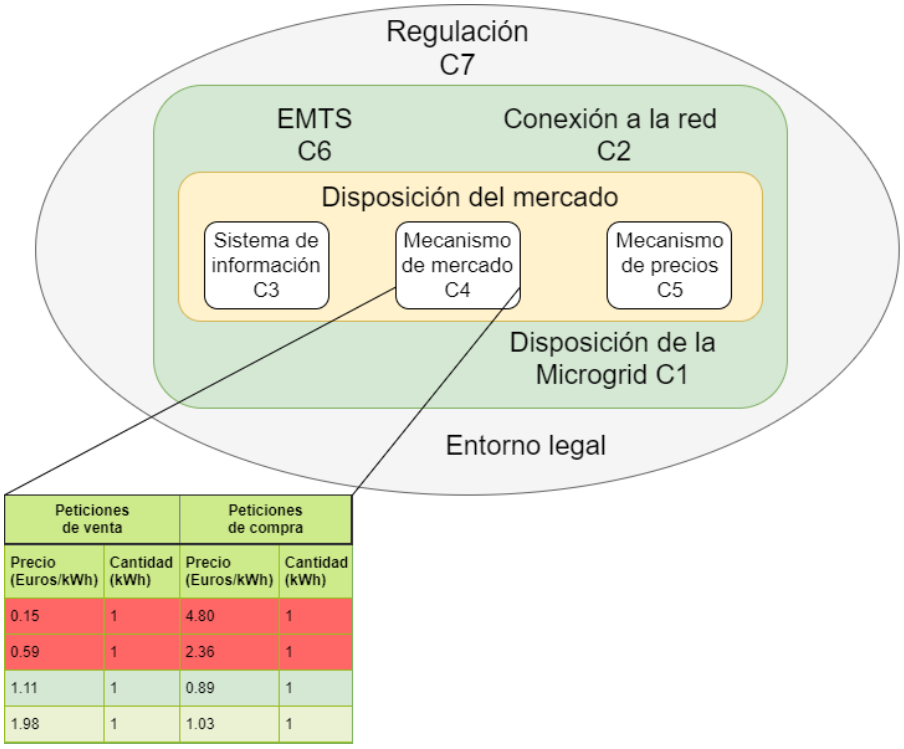
- La dirección de la cartera de salida sea la misma que el que intenta realizar el traspaso.
- La cartera de salida tenga fondos suficientes para realizar el traspaso.

Por otro lado, la plataforma Ethereum utiliza un lenguaje Turing-completo. Esto hace que sea funcionalmente diferente a Bitcoin, obteniendo mucha más flexibilidad, pero a coste de que el sistema pueda llegar a ser impredecible e inseguro.

En el caso de Hyperledger Sawtooth, los Smart Contracts están implementados en los Transaction Processors. Al igual que en Ethereum, pueden estar escritos en una variedad de lenguajes Turing-completos, pero para poder trabajar con ellos, todos los nodos de la red deben tener exactamente el mismo Transaction Processor. Este método, al ser más restrictivo, junto a componentes que ofrecen distintos grados de confidencialidad, hacen que Sawtooth sea una opción más adecuada para nuestro negocio que Ethereum.

8. Mercados de energía en Microgrids

De acuerdo al Departamento de Energía de EE. UU., una Microgrid consiste en un grupo de cargas y recursos energéticos distribuidos interconectados con límites eléctricos claramente definidos que actúa como una única entidad controlable respecto a la red general. Una Microgrid puede conectarse y desconectarse de la red general, lo que la permite operar en modo conectado a la red general(grid-connected) o modo isla, independiente de la red general.



Peticiones de venta		Peticiones de compra	
Precio (Euros/kWh)	Cantidad (kWh)	Precio (Euros/kWh)	Cantidad (kWh)
0.15	1	4.80	1
0.59	1	2.36	1
1.11	1	0.89	1
1.98	1	1.03	1

Fig. 1: Ilustración esquemática de los siete componentes de los mercados en Microgrids. Fuente: [3]

Se pueden derivar siete componentes clave para el funcionamiento eficiente de mercados Micgorid basados en Blockchain basados en Block et al. [6] y Ilic et al. [7]. Se explicará brevemente cada uno de ellos para poder aplicarlos más adelante al caso de uso que nos ocupa (10.3). Se ha realizado una ilustración en la que aparecen cada uno de los componentes. (Fig. 1). En esta figura las dos peticiones superiores serán emparejadas, ya que existen dos compradores que están dispuestos a pagar más que el precio propuesto por los vendedores. El precio será determinado por el mecanismo de precios.

8.1 Disposición de la Microgrid (C1)

Es necesario definir un objetivo claro, los participantes del mercado y las formas de energía que se van a intercambiar. Es posible que existan varios objetivos e incluso que éstos sean contradictorios entre sí. Por ejemplo, se puede buscar maximizar el intercambio de energías renovables y que además se desee minimizar el coste monetario de esta energía. Esto puede funcionar mientras el precio de

mercado de la energía en la Microgrid sea menor que el de la red general. En este caso los consumidores deben elegir entre comprar energía renovable a un alto precio, o energía “marrón” a un bajo precio. La implementación de los precios aparecerá además más adelante en el mecanismo de precios (C5). También es necesario que haya un número mínimo de participantes. Se requiere que un subgrupo de los participantes pueda producir energía, comúnmente a través de sistemas PV. Solamente se permitirá el acceso a ciertos participantes de comunidades o grupos preestablecidos. El tipo de energía también debe ser definido, ya sea electricidad, calor, o una combinación de ambas. Se supondrá que los participantes son interesados en sí mismos (self-interested) y racionales [8]. Esto significa que buscarán siempre la opción que ofrezca mayor beneficio económico. Además, se necesita definir si se va a usar la red general para el transporte de energía (Microgrid virtual), o si se va a construir una Microgrid física.

8.2 Conexión a la red general (C2)

Si la Microgrid está conectada a la red general, es necesario definir los puntos de unión para poder medir precisamente el flujo de energía y rendimiento de la Microgrid. Si se va a tener una Microgrid virtual, será imposible desacoplarse de la red general, y los participantes estarán unidos entre sí a través de un sistema de información (C3). Si la Microgrid es física, debe estar conectada a la red general a través de un número limitado de puntos de enlace para tener una conexión eficiente y permitir un desacople rápido en el caso de cortes de energía.

Para permitir un desacople continuo o modo isla, la Microgrid necesita una gran cantidad de energía propia para poder proveer a los usuarios y asegurar un buen nivel de seguridad y resiliencia. Normalmente esto se consigue a través de un Energy Storage System y flexibilidad a la hora de producir y consumir energía.

8.3 Sistema de información (C3)

Es necesario un sistema de información de alto rendimiento para conectar a todos los participantes del mercado. Este sistema debe estar disponible prácticamente todo el tiempo, y permitirá el mismo acceso a todos los participantes. Un protocolo Blockchain basado en Smart Contracts puede cumplir estos requerimientos. El protocolo Blockchain ofrece una estructura global descentralizada que permite la creación de aplicaciones software (Smart Contracts) sin una plataforma central. Aunque la consistencia y seguridad de los datos son rasgos inherentes de la tecnología Blockchain, es necesaria una conexión segura a los Smart Meters de los participantes. Una vez conseguido esto, los Smart Meters podrán mandar datos energéticos directamente al Blockchain. El mecanismo de identificación del Blockchain dependerá de la disposición del Microgrid (C1). Si solo pueden participar miembros de confianza, bastará con utilizar un mecanismo de identificación basado en identidad por hash. De esta manera se puede evitar que agentes dañinos entren al sistema.

8.4 Mecanismo de mercado (C4)

El mecanismo de mercado abarca la distribución del mercado, reglas de pago, lenguaje y formato de las pujas. El mecanismo de mercado está implementado por el sistema de información (C3). Su objetivo principal es la distribución eficiente de energía a partir de las órdenes de compra y venta de los participantes. Óptimamente esto se hará en (casi) tiempo real. También podrían existir restricciones sobre el mínimo y máximo de energía producido y/o comprado en distintas franjas temporales (e.g. intradía, semanal). El mecanismo puede ser una simple lista de órdenes, un sistema de subasta o un mercado P2P en el que todos los participantes se comunican y pueden acordar precios.

8.5 Mecanismo de precios (C5)

El mecanismo de precios está implementado por el mecanismo de mercado (C4) y su objetivo es la distribución eficientemente la oferta y demanda de energía. En mercados de energía se usan frecuentemente subastas con precios de equilibrio uniformes o individuales como mecanismo de precios. Mientras que una gran parte del precio de la energía tradicional consiste en impuestos y sobrecargas, la energía de una Microgrid puede tener distintas cuotas, e.g. en los casos de Microgrids físicas. Debería haber tanto señales de precio que indiquen la escasez de energía, como bajadas en el precio si hay un superávit de energía.

Económicamente hablando, los mercados locales son beneficiosos para los participantes mientras que el precio de equilibrio sea menor que el precio de la red externa. Sin embargo, por razones socio-económicas (e.g. apoyar la generación local de energía renovable), el precio de la energía local puede superar el de la red general. También podría limitarse el precio máximo de venta al precio actual de la electricidad propuesto por el Gobierno.

8.6 Sistema de administración e intercambio de energía (C6)

El objetivo del Energy Management Trading System (en adelante EMTS) es asegurar automáticamente el suministro de energía para un participante mientras implementa una estrategia de subasta. El EMTS necesita acceder en tiempo real a la oferta y demanda de su participante. Basándose en estos datos, el EMTS predice la consumición y generación, y desarrolla la estrategia de subasta en consecuencia. Un EMTS simple siempre compraría energía cuando el precio de esta cae por debajo del precio de la red general. Sin embargo, el EMTS tiene que tener también en cuenta factores socio-económicos (e.g. preferiblemente comprar energía renovable generada localmente, o comprar la más cercana geográficamente), y las estrategias de cada agente que puede implicar cambiar los precios a lo largo del tiempo, como por ejemplo aumentar el precio de su energía repentinamente.

8.7 Regulación (C7)

La regulación determina cómo los mercados de energía de Microgrids se ajustan a la política de energía actual. De esta manera, las leyes determinan que diseños de mercado están permitidos, los impuestos, y cómo el mercado se puede integrar en el sistema de energía actual. Por lo tanto, el gobierno puede fácilmente apoyar mercados de energía de Microgrids aumentando el uso de recursos locales y disminuyendo el impacto ambiental a través de cambios regulatorios, e.g. ofreciendo subsidios. Pero de la misma manera pueden rechazar la implementación de mercados Microgrid si estos resultan en impactos negativos al sistema tradicional de energía.

En España a día 05/04/2019 se aprobó el Real Decreto que hace viable el autoconsumo individual y colectivo conectado a una red [9]. Este mecanismo es aplicable a instalaciones con una potencia no superior a 100 kilovatios, y siempre que se produzca electricidad a partir de energía de origen renovable. Un consumidor podrá utilizar energía excedente de un vecino si este no está consumiendo su energía proporcional y se encuentra a menos de 500 metros de distancia. Con esta norma, junto a la derogación del “impuesto al sol” en octubre de 2018, se abre la puerta a las posibilidades de mercados de energía locales en España, que hasta ahora no reconocía la figura del autoconsumo colectivo.

Aunque estos sean los elementos que son considerados clave en este trabajo, no son todos los componentes de una Microgrid. No se van a considerar otros más específicos como pueden ser el manejo técnico de la Microgrid o el tratamiento de las pérdidas de energía en forma de potencia reactiva en los intercambios.

IV. Proyecto software

9. Plan de desarrollo

Teniendo en cuenta que la asignatura Trabajo de Fin de Grado está valorada en 12 créditos ECTS (European Credit Transfer and Accumulation System), y que un crédito equivale a 25 horas, podemos estimar que la duración total del proyecto será de unas 300 horas, por lo que se ha adaptado el alcance del proyecto para tratar de ajustarme a dicha duración.

Se han repartido las 300 horas estimadas entre 35 semanas, con lo que quedarían aproximadamente 8.5 horas semanales. Pero debido a la carga de trabajo de otros proyectos realizados en la empresa, la distribución de horas ha sido muy variable, en ocasiones siendo necesarias también varias horas fuera del trabajo.

9.1 Actividades y planificación

En esta sección se presentarán las actividades que se han propuesto para el estudio, diseño e implementación del sistema. Las actividades se han estructurado de forma secuencial para facilitar tanto la planificación como el desarrollo, excepto las tareas sobre la memoria que se van a ir haciendo paralelamente junto al resto de actividades del proyecto, y las pruebas, que se irán haciendo a medida que se avance con la implementación.

A continuación, se presenta la tabla (Cuadro *2*) en la que se puede ver el desglose de tareas y actividades, junto a sus horas estimadas y horas reales.

Actividad	Id Tarea	Tarea	Duración estimada (horas)	Duración real (horas)
Estudio previo	T-01	Documentación sobre Blockchain	10	9
	T-02	Documentación sobre aplicaciones de Blockchain a mercados de energía	10	9
	T-03	Documentación sobre Hyperledger	15	27
	T-04	Prueba de prototipos con Hyperledger	10	10
Modelo de negocio	T-05	Definición de miembros, activos y transacciones, normas, canales	10	15
Análisis	T-06	Requisitos	5	3
Diseño	T-07	Diagrama de casos de uso	5	1
	T-08	Diagramas de arquitectura	5	8
	T-09	Diagramas de secuencia	15	20
Implementación	T-10	Preparar el entorno de desarrollo	5	3
	T-11	Implementar las normas de negocio en el Smart Contract	10	5
	T-12	Implementar puesta a la venta de energía	20	30
	T-13	Implementar compra de energía	15	40
	T-14	Implementar petición de energía	10	8
	T-15	Implementar satisfacción de petición	10	5
	T-16	Implementar edición y borrado de venta de energía	10	9
	T-17	Implementar edición y borrado de venta de petición de energía	5	4
	T-18	Implementar histórico de intercambios de energía	7	5
Despliegue	T-19	Despliegue manual en una máquina	5	10
	T-20	Despliegue por Docker en distintas máquinas	10	20
	T-21	Demo	5	10
Pruebas **A lo largo de la implementación**	T-22	Pruebas de Sawtooth	5	2
	T-23	Pruebas de cliente	10	15
	T-24	Pruebas de Proxy	5	10
	T-25	Pruebas de cliente con API y Smart Contract	15	15
	T-26	Pruebas de sistema completo	15	15
Memoria **A lo largo de todo el proyecto**	T-27	Índice de memoria	2	2
	T-28	Estado del arte	10	20
	T-29	Análisis y diseño	1	1
	T-30	Documentación sobre la implementación	10	15
	T-31	Documentación sobre el despliegue	10	10
	T-32	Documentación sobre las pruebas	5	5
	T-33	Corrección de errores y finalización	15	15

Cuadro 2: Tabla desglose de tareas de la planificación

Se puede apreciar que el total de horas reales (376) ha sido bastante mayor al total de horas estimadas (300), sobre todo en la parte de implementación. Esto ha sido a causa de la poca documentación a la hora de desarrollar una aplicación Sawtooth, y menos aún, enlazarla con un cliente web. Sin embargo, una vez finalizado este trabajo y con la experiencia adquirida de resolver los problemas encontrados, realizar un trabajo nuevo similar supondrá un total de horas más cercano al estimado inicialmente.

En Fig. 2 y Fig. 3 se presenta el diagrama de Gantt correspondiente a dicha distribución.

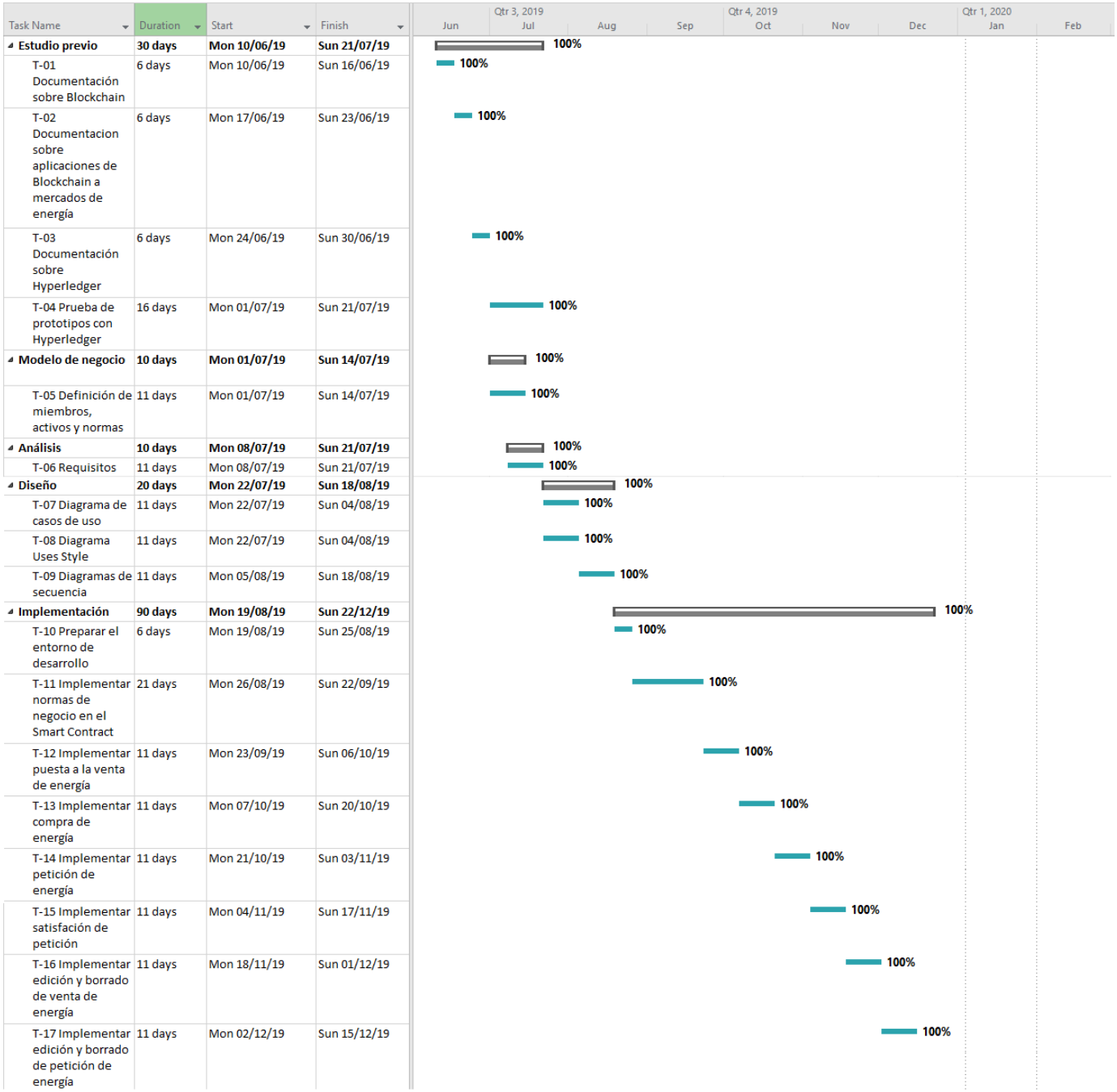
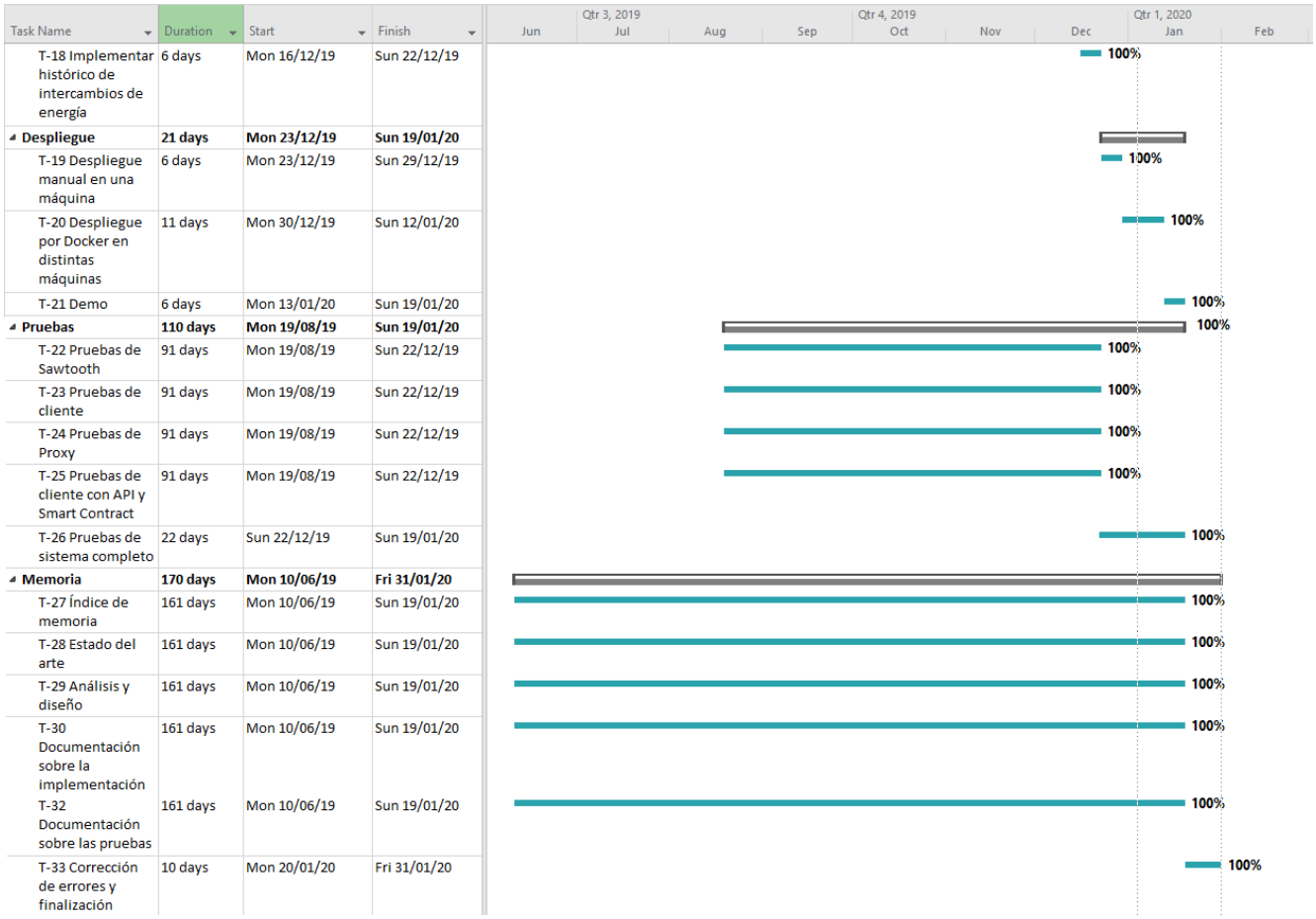


Fig. 2: Diagrama Gantt de la planificación, Parte I



Task Name	Duration	Start	Finish
T-18 Implementar histórico de intercambios de energía	6 days	Mon 16/12/19	Sun 22/12/19
Despliegue	**21 days**	**Mon 23/12/19**	**Sun 19/01/20**
T-19 Despliegue manual en una máquina	6 days	Mon 23/12/19	Sun 29/12/19
T-20 Despliegue por Docker en distintas máquinas	11 days	Mon 30/12/19	Sun 12/01/20
T-21 Demo	6 days	Mon 13/01/20	Sun 19/01/20
Pruebas	**110 days**	**Mon 19/08/19**	**Sun 19/01/20**
T-22 Pruebas de Sawtooth	91 days	Mon 19/08/19	Sun 22/12/19
T-23 Pruebas de cliente	91 days	Mon 19/08/19	Sun 22/12/19
T-24 Pruebas de Proxy	91 days	Mon 19/08/19	Sun 22/12/19
T-25 Pruebas de cliente con API y Smart Contract	91 days	Mon 19/08/19	Sun 22/12/19
T-26 Pruebas de sistema completo	22 days	Sun 22/12/19	Sun 19/01/20
Memoria	**170 days**	**Mon 10/06/19**	**Fri 31/01/20**
T-27 Índice de memoria	161 days	Mon 10/06/19	Sun 19/01/20
T-28 Estado del arte	161 days	Mon 10/06/19	Sun 19/01/20
T-29 Análisis y diseño	161 days	Mon 10/06/19	Sun 19/01/20
T-30 Documentación sobre la implementación	161 days	Mon 10/06/19	Sun 19/01/20
T-32 Documentación sobre las pruebas	161 days	Mon 10/06/19	Sun 19/01/20
T-33 Corrección de errores y finalización	10 days	Mon 20/01/20	Fri 31/01/20

Fig. 3 Diagrama Gantt de la planificación, Parte II

10. Modelo de negocio

10.1 Modelo de negocio planteado

En este apartado se van a detallar los componentes del modelo de negocio que definido, los diferentes mecanismos de mercado que se han planteado y la elección final.

Participantes:

- **Prosumer**: Usuarios que consumen energía propia y de otros Prosumers, pero además producen energía, típicamente a través de PV. Tienen permitido realizar todas las operaciones en la red Blockchain.
- **Consumer**: Usuarios que consumen energía propia, y de los participantes Prosumers. Tienen permitido realizar las mismas operaciones que los Prosumers, excepto las relacionadas con vender energía, ya que estos usuarios no producen.

Activos:

- **Energía**: La unidad de energía con la que se va comercializar es el Kilovatio hora (kWh). 1 kWh es el equivalente a 1 kW de potencia sostenida durante 1 hora. Tomamos esta unidad como referencia porque es de las más utilizadas a la hora de expresar el consumo de electrodomésticos domésticos, y total de energía consumida y producida mensualmente.
- **Unidad monetaria**: Existen dos opciones. Utilizar una moneda virtual, definida en el propio sistema, o utilizar una moneda real. Ya que la realización de este trabajo se ha llevado a cabo en España, se ha decidido que la unidad monetaria sea el Euro (€). Sin embargo, en el futuro será algo totalmente configurable y abierto.

Operaciones:

- **Poner a la venta energía**: Los Prosumers podrán poner a la venta una cantidad de energía (kWh) durante un periodo de tiempo, a un precio determinado (€).
- **Comprar energía a la venta**: Prosumers y Consumers podrán comprar el total o una parte de la energía ofertada por un Prosumer, siempre y cuando lo haga dentro del periodo de tiempo válido. El precio de la energía es el que decide el vendedor. Si se compra solo una parte de la ofertada, el resto de energía sigue estando disponible para comprar para otro usuario.
- **Crear petición de energía**: Operación análoga a “Poner a la venta energía”. Prosumers y Consumers pueden solicitar una cantidad de energía a un precio determinado durante un periodo de tiempo.
- **Satisfacer petición de energía**: Un Prosumer puede satisfacer una petición de energía, ofertando el total o una parte de la energía solicitada, siempre y cuando lo haga dentro del periodo de tiempo válido. Si satisface solo una parte, el resto de energía podrá seguir siendo satisfecha por otro Prosumer.

- **Modificar energía a la venta**: Un Prosumer que haya puesto energía a la venta puede modificar la cantidad de energía que tenía a la venta (aumentando o disminuyendo la cantidad), y el precio.
- **Modificar petición de energía**: Un Consumer o Prosumer que haya creado una petición de energía puede modificar la cantidad de energía que solicita, y el precio.
- **Eliminar energía a la venta**: Un Prosumer que haya puesto energía a la venta puede eliminar la oferta. Esto no afecta a los usuarios que hayan comprado de esta petición previamente.
- **Eliminar petición de energía**: Un Consumer o Prosumer que haya creado una petición de energía puede eliminar la petición. Esto no afecta a los usuarios que hayan satisfecho previamente esta petición.
- **Ver historial de compras y ventas**: Cualquier participante puede ver las compras y ventas de energía que se han llevado a cabo en el sistema, pudiendo ver datos relevantes como comprador, vendedor, el precio, la cantidad de energía y la fecha en la que se ha realizado la transacción.

Normas y restricciones:

Actualmente, los usuarios son libres de poner a la venta la cantidad de energía que decidan, siempre y cuando sea una cantidad positiva. Esto puede variar en el futuro, si la normativa decide imponer algún tipo de restricción sobre la cantidad máxima energía que se puede vender a lo largo del día, o si es necesaria una cantidad mínima de energía por transacción para minimizar pérdidas en el transporte de la energía.

Lo mismo sucede con el precio. Al ser un mercado libre, los usuarios son libres de poner el precio que les parezca oportuno. Este precio puede ser inferior al coste de energía que ofrecen las compañías, y superior al que ofrecen estas compañías a la hora de comprar un excedente de energía, haciendo que tanto el vendedor como el comprador obtengan beneficios. Pero también puede ser superior, y permitir que los usuarios compren libremente, aun no siendo la mejor opción económicamente hablando, puede haber razones sociales o ecológicas para pagar más, como promover el comercio local o la generación de energía renovable.

Canales:

El sistema de Blockchain estará conectado a través de la red de internet, sin embargo, es necesario que los participantes estén físicamente próximos para poder cumplir con la normativa (Apartado 8.7).

La energía irá por la red eléctrica ya existente, ya que nuestra Microgrid será virtual y no física (Apartado 8.2). La energía podrá ser intercambiada directamente entre los participantes a través de los Smart Meters, o directamente vertida a la red general.

El intercambio de unidades monetarias podrá realizarse a través de transferencias bancarias, como por ejemplo PayPal. Si en el futuro se añade la funcionalidad de una moneda virtual / saldo, el canal sería la propia Blockchain.

10.2 Mecanismos de mercado

La idea inicial era hacer un sistema de subastas para realizar los intercambios de energía. En este sistema los vendedores pondrían a la venta una cantidad de energía a un precio determinado, y los compradores, sin conocer la energía que está a la venta en el sistema, solicitarían una cantidad y pujarían, ofreciendo el precio que están dispuesto a pagar. Idealmente en este sistema, las demandas de los vendedores que hayan puesto los precios más bajos serán las primeras en ser satisfechas por los compradores que hayan pujado más alto, es decir, estén dispuestos a pagar un mayor precio por unidad de energía.

Este sistema tendría un alto nivel de automatismo. Sin embargo, puede haber situaciones no deseadas, por ejemplo, un comprador desee comprar energía, pero el precio que está dispuesto a pagar queda ligeramente por debajo del precio solicitado por los vendedores. Al ser una subasta a ciegas, no tendría forma de saberlo, y los vendedores tampoco sabrían que este usuario desea comprar y puede darse el caso de que acabe el periodo de validez de la energía y no se realice ningún intercambio, “perdiendo” por tanto esta energía. Técnicamente no se perdería, sino que se vertería a la red o se vendería a la compañía eléctrica a su precio fijo, pero es una situación que podría resolverse en un sistema de compra directa, en el que los vendedores podrían haber disminuido el precio o el comprador aumentado el precio dispuesto a pagar para realizar el intercambio.

Pero independientemente de las ventajas y desventajas de este sistema, también nos encontramos con el problema técnico de transportar este método a un sistema Blockchain.

En un caso práctico existirán varios vendedores y compradores. Un sistema Blockchain está basado en un estado global, y transacciones, que modifican este estado. El hecho de tener elementos tan variables como un sistema automático que reciba las peticiones de venta, las de compra, gestione las subastas y elija quién satisface a quién requiere de un elemento central. Suponiendo en este caso real que se han creado varias pujas, y pongamos que la duración de una puja es de 1 hora, no hay forma de que el sistema pueda “cerrar” la puja automáticamente en el rango de tiempo de 1 hora. El sistema Blockchain no puede generar transacciones automáticamente. Las dos opciones serían:

- Que el vendedor cierre la puja manualmente a la hora, una solución bastante manual y poco práctica teniendo en cuenta que buscamos un sistema automático.
- Crear una capa superior al Blockchain, donde exista un sistema central que será el que reciba las peticiones de venta y pujas de los usuarios, y sea este propio sistema el que después se comunique con el ledger, escribiendo las pujas satisfechas. Esta solución, además de requerir un sistema central (filosofía contraria a los sistemas distribuidos que ofrece Blockchain), consistiría en una red con un solo nodo, que utilizaría el ledger como sistema complementario simplemente para apuntar los intercambios de energía que ha

llevado a cabo un sistema central, perdiendo así todo el potencial que tanto Blockchain como los Smart Contracts ofrecen.

Si el “matching” o unión de pujas y ventas las realizaran los nodos particularmente, existirían otros problemas, como que habría que elegir un nodo único para que decidiera las pujas y ventas satisfechas en un periodo de tiempo, y puede darse el caso de que el nodo del vendedor esté apagado en el momento de cierre de la subasta.

Es posible que esta solución hubiera sido posible con otro tipo de Smart Contracts, utilizando otro framework como Ethereum, pero por las razones dadas en el Apartado 5, se ha decidido utilizar Hyperledger Sawtooth y se alcanzará la mejor solución posible utilizando este framework.

Tras analizar todo lo comentado anteriormente respecto al sistema de pujas, se decidió definir una solución totalmente diferente, y que además requiere de la participación del usuario, pero que mantiene la naturaleza distribuida del Blockchain, y un uso razonable de Smart Contracts.

En esta solución los vendedores añadirán sus ofertas de energía al ledger, indicando la cantidad, precio, fecha de creación de la venta y fecha de validez de la venta. Los compradores podrán acceder a todas las ofertas de energía que estén en el rango de tiempo válido y que no estén completamente satisfechas. Entonces un comprador elegirá la oferta que le interese, e indicará la cantidad de energía que desea comprar de ese vendedor, al precio indicado por él. De esta manera se ha creado un “contrato” que es aceptado automáticamente por ambos participantes. La lógica que comprueba la validez de las fechas, o de que no se compre más cantidad de la ofertada se encontrará en el Smart Contract, o en el caso de Sawtooth, en el Transaction Processor. Con este sistema se asegura que solo se realicen compras válidas, y además han desaparecido todos los problemas que traía el sistema de subastas automático mencionado anteriormente.

Como un añadido final, se hará este sistema bidireccional. Esto significa que además de permitir que los vendedores pueden poner a la venta energía, análogamente los compradores pueden realizar peticiones de energía al precio que deseen por si no les interesa los precios que hay en el mercado actualmente, o por cualquier otra razón, permitiendo a los productores satisfacer su necesidad.

Por las razones expuestas anteriormente, finalmente el proyecto implementará un mecanismo de compra y venta one-on-one en la que cada intercambio de energía tiene un comprador y un vendedor, y ambas partes conocerán en todo momento la identidad del otro participante.

10.3 Revisión del caso de uso planteado frente a la definición de mercado Microgrid (Apartado 8)

En este apartado se volverá sobre los conceptos mencionados en el Apartado 8, definiendo todos y cada uno de los elementos en él mencionados para el caso concreto que se plantea en este proyecto.

C1 – Disposición de la Microgrid: La energía producida que con la que se va a comercializar será originada por los usuarios Prosumers de la red, principalmente por PV, pero puede ser de otros orígenes como eólica. Sin embargo, esto es indiferente para el sistema planteado. Al ser una red permisionada, será necesaria la autorización de un administrador de la red para poder entrar y participar en ella. El objetivo de la Microgrid es crear un mercado libre de energía en el que los participantes puedan comprar y vender al precio que ellos deseen, sin intermediarios.

C2 – Conexión a la red general: La Microgrid que utilizará este sistema será una Microgrid virtual. Los participantes no estarán conectados físicamente en un sistema diferente al de la red, si no que se utilizará este mismo. El elemento de unión de los participantes es “virtual”, será la participación a la red Blockchain.

C3 – Sistema de información: El sistema de información será la red Blockchain implementada utilizando el framework Hyperledger Sawtooth.

C4 – Mecanismo de mercado: Un sistema de compra one-on-one, explicado detalladamente en el Apartado 10.2.

C5 – Mecanismo de precios: Al ser un mercado libre, los precios no tienen ningún tipo de restricción y cada usuario puede poner el precio que desee a la hora de vender y hacer una petición de compra de energía.

C6 – Sistema de administración e intercambio de energía: Al utilizar un mecanismo de mercado one-on-one, no es necesario un sistema que gestione la energía. Un usuario sabe la cantidad de energía que va a recibir, el precio que paga y cuándo la va a recibir en todo momento.

C7 – Regulación: Se espera que el futuro se especifique más la definición y legislación sobre los mercados en Microgrids. Sin embargo, el sistema es totalmente aplicable en el ámbito de la regulación actual existente en España [9], pero si en el futuro hubiera cambios, se modificaría el sistema acorde con la ley. Al final, este sistema solo sirve para facilitar el intercambio, y cumplirá con todo lo que sea posible, pero la responsabilidad final y las mayores limitaciones las tendrán los usuarios participantes.

11. Análisis

11.1 Requisitos funcionales

ID	Nombre	Descripción
RF01	Poner a la venta energía	El sistema debe permitir a un usuario Prosumer poner a la venta energía.
RF02	Modificar energía a la venta	El sistema debe permitir a un usuario Prosumer modificar la cantidad de energía y el precio de la energía que ha puesto a la venta.
RF03	Eliminar energía a la venta	El sistema debe permitir a un usuario Prosumer que ha puesto energía a la venta, eliminar esa venta de energía.
RF04	Comprar energía a la venta	El sistema debe permitir a cualquier usuario comprar energía a la venta.
RF05	Realizar petición de energía	El sistema debe permitir a cualquier usuario hacer una petición de energía, indicando cantidad y precio.
RF06	Modificar petición de energía	El sistema debe permitir a un usuario cualquiera modificar la cantidad de energía y precio de la petición de energía que ha hecho.
RF07	Eliminar petición de energía	El sistema debe permitir a un usuario cualquiera que ha creado una petición de energía eliminar esta petición.
RF08	Acceder a resumen informativo	El sistema debe permitir acceder a un resumen informativo que mostrará un historial de todos los intercambios de energía, junto a información correspondiente de dichos intercambios.

Cuadro 3: Tabla de requisitos funcionales

11.2 Requisitos no funcionales

ID	Nombre	Descripción
RNF01	Aplicación distribuida y descentralizada	El sistema utilizará Blockchain, estando formado por una red de nodos haciendo que la aplicación sea distribuida y carente de un elemento central.
RNF02	Red permisionada	Es necesario permiso del administrador para poder participar en la red con un nodo.

RNF03	Entorno de uso	La aplicación debe funcionar en cualquier máquina que pueda ejecutar Docker. En el caso de Windows, Docker Desktop. En caso de Linux, Docker y Docker Compose. El sistema operativo recomendado para el uso y desarrollo es Ubuntu 16.04.
RNF04	Lenguajes de programación	El cliente web estará desarrollado utilizando HTML 5, CSS 3 y JavaScript. La lógica de los Smart Contracts estará desarrollada en Python 3.
RNF05	Librerías utilizadas	Para la interfaz del cliente web se utilizarán estilos como Bootstrap. Para la lógica del cliente se usarán una serie de módulos de NodeJS como Ajax, jQuery, el SDK de Sawtooth para las comunicaciones con el ledger, y otros varios para utilidades.
RNF06	Facilidad de despliegue y uso	El sistema (de prueba) será fácilmente desplegable por usuarios con un conocimiento básico de uso de consola de comandos, y usable por cualquier tipo de usuario con conocimientos básicos de compra y venta.
RNF06	Solución open source	La tecnología utilizada deberá ser open source, permitiendo la colaboración entre desarrolladores, y el añadido de modificaciones en el código para personalizarlo y usarlo en proyectos propios.

Cuadro 4: Tabla de requisitos no funcionales

11.3 Requisitos de información

ID	Nombre	Descripción
RI01	Tipo de información	El sistema almacenará toda la información sobre las compras y ventas que se produzcan.
RI02	Visibilidad de la información	Cualquier usuario con un nodo dentro de la red podrá acceder a toda la información en el ledger.
RI03	Inmutabilidad de la información	La información de las transacciones que formen parte del ledger no podrá ser modificada.

Cuadro 5: Tabla de requisitos de información

11.4 Casos de uso

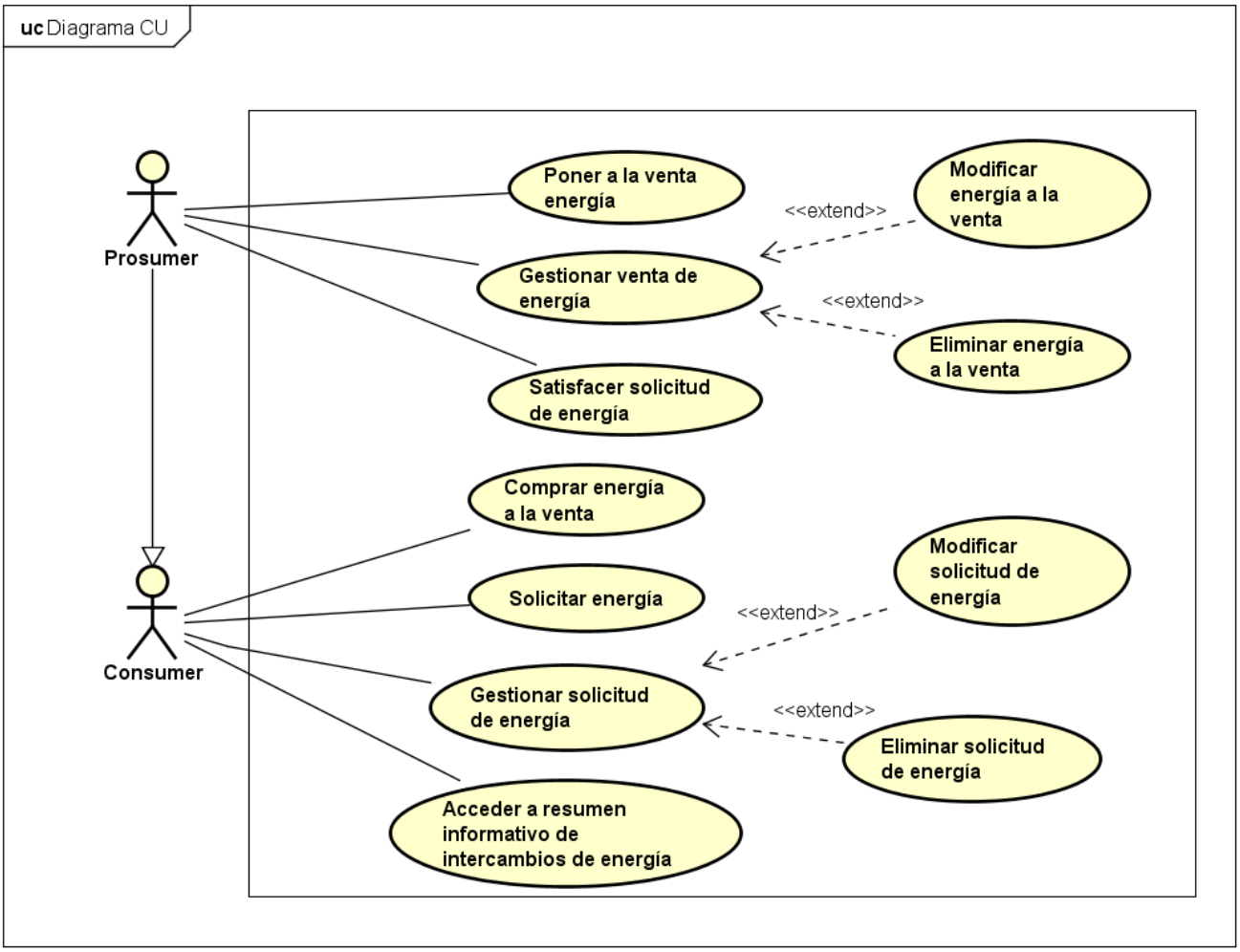


Fig. 4: Diagrama de casos de uso

11.5 Descripción de los casos de uso

Esta sección se encuentra en el VI. Anexo, 18. Descripción de los casos de uso.

11.6 Modelo de dominio

En nuestro sistema hay dos tipos de modelos. El primero sería el modelo del contenido de las transacciones, o payload. Todas las transacciones que gestione un Transaction Processor deben tener los mismos campos, es decir, utilizar el mismo modelo de payload. El segundo tipo de modelo utilizado es que usa el ledger para almacenar los states, o estado global del ledger. Hay que tener en cuenta que este state que almacena el ledger no es lo mismo que las transacciones introducidas, sino que es un modelo creado específicamente para esta solución.

11.6.1 Modelo de las transacciones

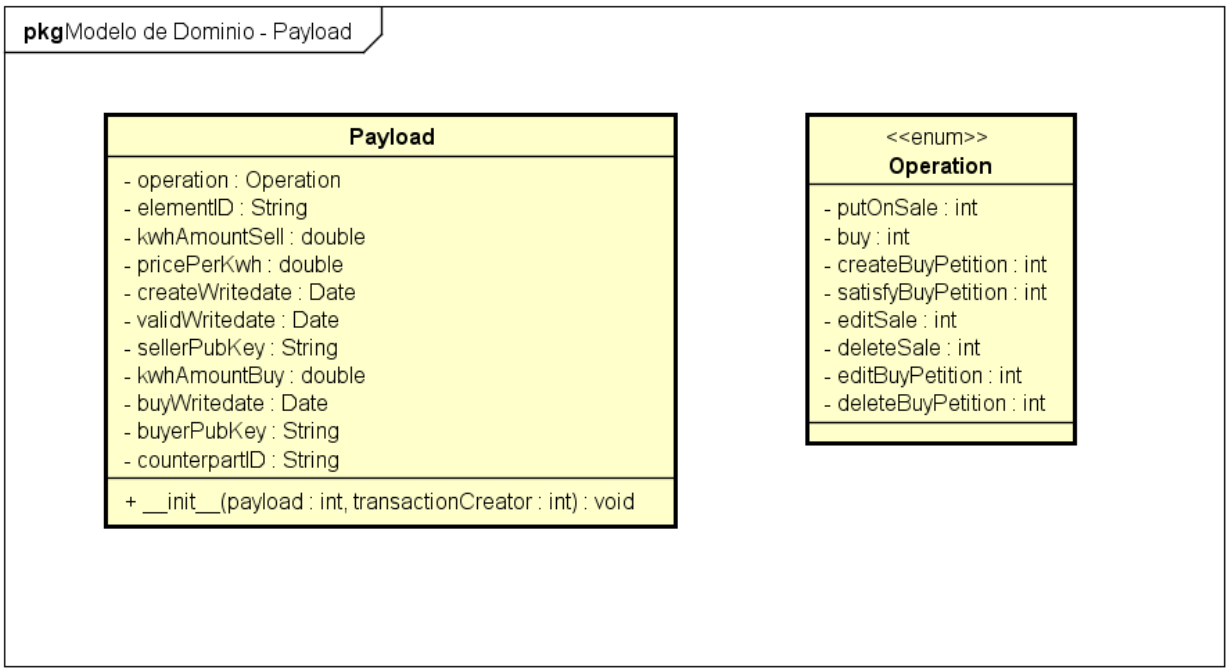


Fig. 5: Modelo de dominio de Payload

A continuación, se detalla brevemente cada atributo del modelo.

operation: Indicará que tipo de operación será realizada en el sistema al procesar la transacción. Las operaciones son:

- *putOnSale*: Poner energía a la venta.
- *buy*: Comprar energía que está a la venta.
- *createBuyPetition*: Crear una petición de compra de energía.
- *satisfyBuyPetition*: Satisfacer una petición de compra de energía.

- *editSale*: Modificar una energía a la venta.
- *deleteSale*: Eliminar una energía a la venta.
- *editBuyPetition*: Modificar una petición de compra de energía.
- *deleteBuyPetition*: Eliminar una petición de compra de energía.

kwhAmountSell: Cantidad de energía relacionada con la venta. Es decir, que este campo se utiliza tanto a la hora de poner energía a la venta, como para indicar también la cantidad de energía que ofrece un Prosumer en la operación de satisfacer una petición de compra.

pricePerKwh: Precio en Euros por kWh para el intercambio de energía, ya sea el que propone el vendedor de energía, o el creador de la petición de compra.

createWriteDate: Fecha de creación, ya sea de la energía a la venta, como de la petición de compra de energía.

validWriteDate: Fecha hasta la cuál la operación es válida, ya sea de la energía a la venta, como de la petición de compra de energía.

elementID: Identificador único de la puesta a la venta de energía o petición de compra. Este identificador está generado con el estándar UUID versión 4, ya que las otras 3 versiones no han sido aplicables a este problema.

sellerPubKey: Clave pública del vendedor de energía, o usuario que ofrece energía en una petición de compra.

kwhAmountBuy: Análogamente a kwhAmountSell, será el campo que representará la cantidad de energía relacionada con la compra.

buyWriteDate: Fecha en la cual se ha realizado un intercambio de energía.

counterpartID: Análogo a elementID. Identificador único de una compra o satisfacción de petición de energía.

buyerPubKey: Clave pública del comprador de energía, o usuario que satisface una petición de compra de energía.

11.6.2 Modelo del estado global

Como hemos visto anteriormente, el modelo que usará el ledger puede no tener nada que ver con la información de las transacciones, y cada desarrollador es libre de implementarlo como desee.

Esta información se almacena en un Merkle tree. En concreto, en los nodos hoja, que pueden ser accedidos usando un esquema de direcciones de 35 bytes, representados como 70 caracteres hexadecimales.

Estas direcciones deben empezar con un prefijo de 6 caracteres hexadecimales representando el namespace, o nombre identificador del Transaction Processor.

El resto de la dirección puede ser calculado de múltiples maneras, siendo libre totalmente a elección del desarrollador. En este caso, se decidió que la dirección estuviera formada de la siguiente manera:

- Los primeros 3 bytes, o 6 caracteres hexadecimales serán los 6 primeros caracteres resultado de pasar el nombre de la aplicación, “enerblock”, por el algoritmo de hashing SHA-512. El resultado será **5a45ce**.
- El siguiente byte, o carácter hexadecimal, representará que elemento está almacenado. En este caso ‘00’ representa energía a la venta, ‘01’ una compra de energía, ‘02’ una petición de compra de energía, y ‘03’ una satisfacción de petición de compra.
- Los siguientes 32 bytes o 62 caracteres hexadecimales serán los primeros 62 caracteres resultantes de pasar el UUID del elemento.

Un ejemplo práctico sería la creación de una venta de energía, cuyo elementID, o UUID es “d72c9225-630d-496b-8378-11bcc23f403c”. En este caso, este elemento se almacenará en la dirección: 5a45ce + 00 + 78f560930f69a6a71a1ab812a8fa9e82eed9e46237523003848678a29bdebb.

Igual que la generación de la dirección, la información almacenada es decisión del desarrollador en cuestión. En este caso se ha decidido almacenar la información de la siguiente manera:

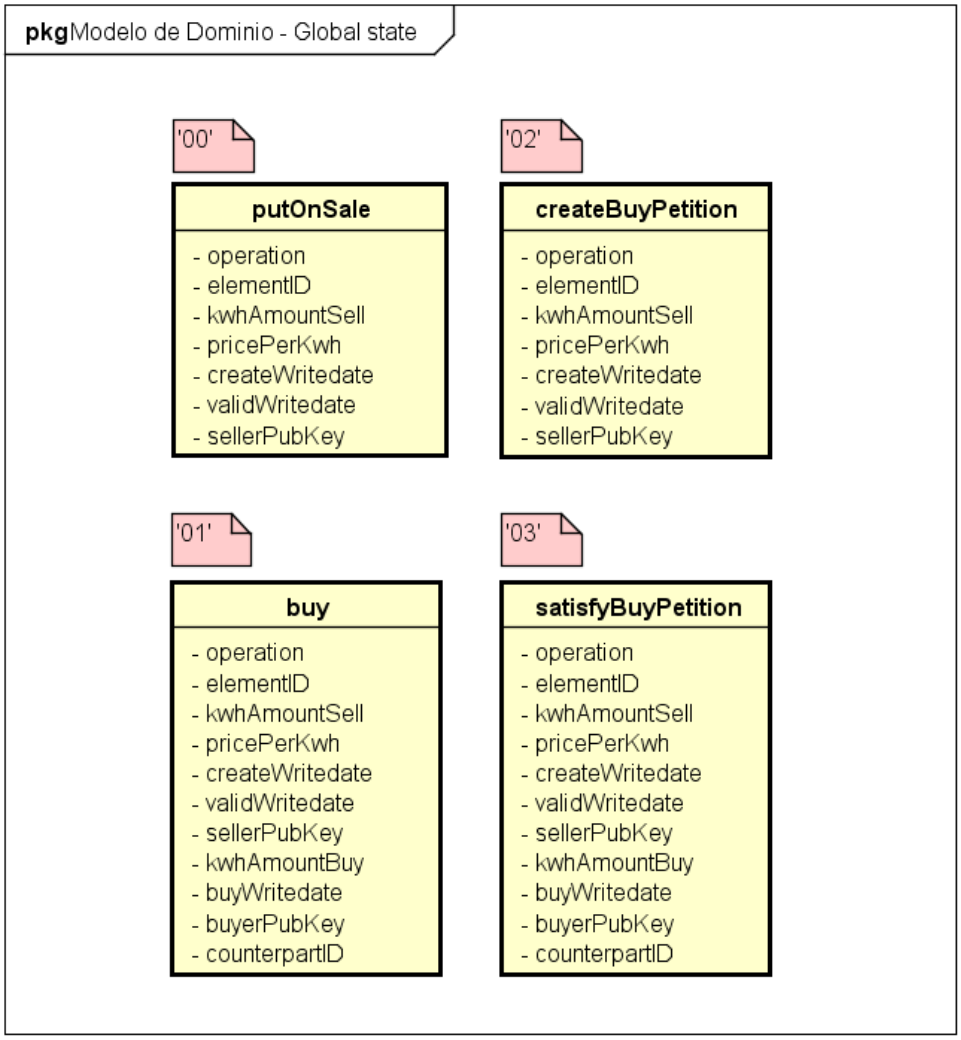


Fig. 6: Modelo de dominio de Global State

En este caso, se han utilizado los mismos nombres que en el modelo de payload, por simplicidad en el cliente. Los apartados que interesan en cada caso son los siguientes:

En putOnSale

- **kwhAmountSell** es la cantidad de energía que se pone a la venta.
- **pricePerKwh** es el precio en Euros por kWh que propone el vendedor.
- **createWriteDate** es la fecha de creación de la venta.
- **validWritedate** es la fecha hasta la cual es válida la venta.
- **elementID** es el identificador único de la venta.
- **sellerPubKey** es la clave pública del vendedor.

En createBuyPetition

- **kwhAmountSell** es la cantidad de energía que solicita el creador de la petición.
- **pricePerKwh** es el precio en Euros por kWh que está dispuesto a pagar por la energía.
- **createWriteDate** es la fecha de creación de la petición de compra.
- **validWritedate** es la fecha hasta la cual es válida la petición de compra.
- **elementID** es el identificador único de la petición de compra.
- **sellerPubKey** es la clave pública del creador de la petición de compra.

En buy

- **kwhAmountSell** es la cantidad de energía que estaba a la venta al realizarse la compra.
- **pricePerKwh** es el precio en Euros por kWh que propuso el vendedor.
- **createWriteDate** es la fecha de creación de la venta.
- **validWritedate** es la fecha hasta la cual es válida la venta.
- **elementID** es el identificador único de la compra.
- **sellerPubKey** es la clave pública del vendedor.
- **kwhAmountBuy** es la cantidad de energía que compra el comprador.
- **buyWritedate** es la fecha en la que se ha realizado la compra.
- **counterpartID** es el identificador único de la venta.
- **buyerPubKey** es la clave pública del comprador.

En satisfyBuyPetition

- **kwhAmountSell** es la cantidad de energía que estaba solicitando el creador de la petición.
- **pricePerKwh** es el precio en Euros por kWh que propuso el creador de la petición.
- **createWriteDate** es la fecha de creación de la petición.
- **validWritedate** es la fecha hasta la cual es válida la petición.
- **elementID** es el identificador único de la satisfacción de la petición.
- **sellerPubKey** es la clave pública del creador de la petición.
- **kwhAmountBuy** es la cantidad de energía que ha ofrecido el Prosumer al creador de la petición.
- **buyWritedate** es la fecha en la que se ha realizado la satisfacción de la petición.
- **counterpartID**es el identificador único de la petición de energía.
- **buyerPubKey** es la clave pública del Prosumer que ha satisfecho la petición.

Sawtooth utiliza en estándar secp256k1 ECDSA para firmar transacciones. Lo que significa que casi cualquier conjunto de 32 bytes es una clave válida. Se han generado dos claves privadas de prueba utilizando openssl en Linux, pero también existe la opción de utilizar el módulo secp256k1 de Python. Es posible sacar la clave pública a partir de la privada, pero obviamente no lo contrario.

Respecto a las fechas de creación y de validez, es necesario indicar ambas. En un ejemplo real, cuando se desea vender energía, no es energía que se disponga de ella actualmente, sino que se trata

de energía que se prevé que se tendrá de excedente dentro de un tiempo. Por esta razón el tiempo de validez es de libre introducción para el usuario. Es posible que un Prosumer solo sepa lo que va a producir en el rango de tiempo de una hora, y otro lo conozca con días de antelación porque usa otro sistema de predicción.

Al llegar el momento límite de validez de la energía a la venta, no se podrá comprar más de esa oferta y los se enviará la energía desde el vendedor a los compradores correspondientes consultando el historial de compras.

El sistema de este trabajo funciona como un mercado de compromisos. Los usuarios que compran obligan al vendedor a comprometerse a enviar la energía ofrecida. Si el vendedor no tuviera energía suficiente para satisfacer a todos sus compradores, deberá ofrecer energía propia comprada a una empresa eléctrica, u ofrecer una compensación monetaria equivalente a la energía por el precio que debió haber enviado, pero esto es un problema fuera del ámbito de la aplicación que podrá resolverse de diferentes maneras.

12. Diseño

12.1 Arquitectura

Se pasa a describir la arquitectura genérica del framework utilizado, junto con la arquitectura del desarrollo concreto realizado en el ámbito de este proyecto.

Como podemos ver en Fig. 7, una red Sawtooth está compuesta de un conjunto de nodos validadores interconectados. En Sawtooth todos los nodos son validadores, por lo que me referiré a ellos como nodos por simplicidad.

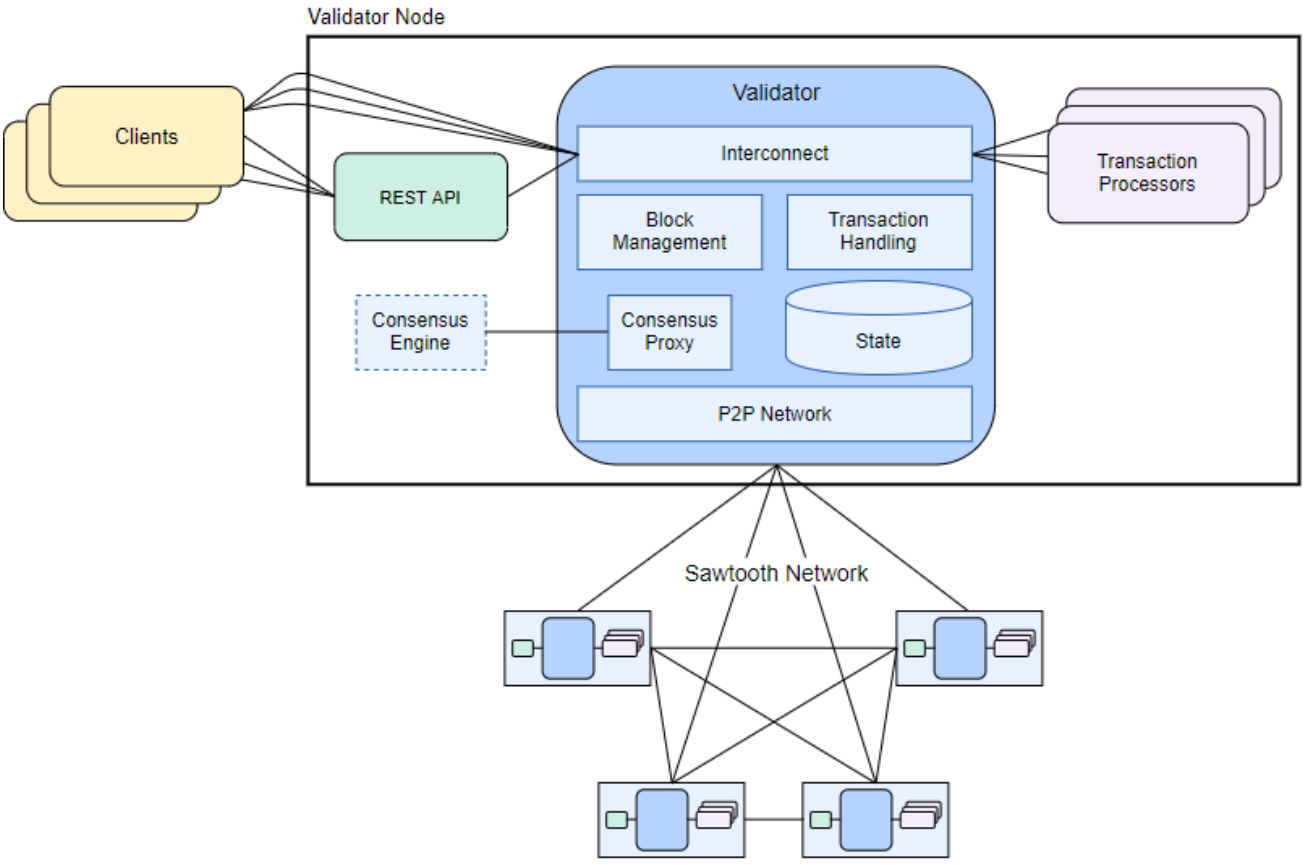


Fig. 7: Arquitectura de alto nivel de una red Sawtooth (Documentación oficial de Sawtooth 2019)

Un nodo está compuesto de los siguientes elementos:

- **Clientes**: Un nodo puede tener varios clientes, por ejemplo, en una vivienda existe un nodo, pero se han diseñado diferentes interfaces para interactuar con la red, añadiendo o limitando alguna funcionalidad. También es posible que un nodo no tenga ningún

cliente, por lo consiguiente, los usuarios no podrán interactuar con la red. Pero este nodo sigue siendo funcional, con una copia del estado global sincronizada con el resto de la red, e incluso puede participar para validad nuevos bloques.

- **REST API**: La API REST de Sawtooth permite la comunicación entre un cliente y la red, ya sea ofreciendo información contenida en la red a través de un GET, o permitiendo añadir nuevas transacciones a través de un POST. Este elemento es también opcional, por ejemplo, en el caso de que el nodo no tenga un cliente.
- **Validador**: Es el núcleo del nodo y se encarga de la mayor parte de la funcionalidad. Se encarga de conectarse a la API REST, a los Transaction Processors, a otros nodos en la red, la gestión de los bloques, poner en funcionamiento el algoritmo de consenso y mantener el estado sincronizado con el resto de participantes.
- **Transaction Processor**: Elemento equivalente a un Smart Contract. Es el encargado de recibir transacciones, validarlas y actuar dependiendo de cómo esté implementado, por ejemplo, modificando el estado global. Un nodo puede tener tantos Transaction Processors como quiera.

Los elementos de la API y el Validator están ya desarrollados por el propio framework de Sawtooth, ya que son exactamente igual para cualquier aplicación. En este proyecto se ha desarrollado lo siguiente:

- **Cliente**: En este caso un cliente Web encargado de interactuar con la API.
- **Transaction Processor**: Contiene la lógica del Smart Contract.

La imagen Fig. 7 muestra un diagrama de alto nivel que representa esta aplicación en concreto, en una red que tuviera dos nodos.

Concretando los elementos respecto al esquema de arquitectura general (Fig. 7), el cliente en este proyecto es un cliente web. La vista está desarrollada en HTML5 y el controlador en un conjunto de ficheros JavaScript. Por el hecho de utilizar un cliente web, es necesario añadir un elemento entre el cliente y la API REST, un proxy. La razón es que la API REST de Sawtooth no soporta Cross-Origin Resource Sharing. Llegados a este punto, tenía dos opciones: Optar por desarrollar un cliente command-line interface, pero el proyecto quedaría bastante insuficiente; o solucionar este problema, montando la API tras un proxy, como sugiere la documentación oficial de Sawtooth[10]. Se decide documentarse sobre los proxys y su funcionamiento y finalmente se consigue montar un proxy con Apache.

Finalmente, se explicarán los Transaction Processors. Enerblock Transaction Processor es el Smart Contract encargado de gestionar todo lo relacionado con la funcionalidad de intercambio de energía, ya se explicó en el Apartado 11.6.1 de las operaciones que era capaz de realizar.

El Settings Transaction Processor es un Transaction Processor que viene ya definido por Sawtooth. Es el encargado de crear el bloque génesis en una red, es decir, el primer nodo deberá tener este TP funcionando para poder poner la red en marcha. El resto de nodos que se unan a la red posteriormente no están obligados a tenerlo, ya que realizarán una copia del estado global de los nodos que ya estén

previamente en la red. Este TP también permite realizar cambios en las opines de la Blockchain sin necesidad de detener la ejecución. Estos cambios se llaman “proposals”. Por ejemplo, se puede modificar el tiempo de espera de un nodo antes de proponer un nuevo bloque (por si hay otros nodos añadiendo nuevas transacciones), o el máximo de transacciones por bloque. Estos proposals pueden llevarse a cabo de dos maneras. La primera, configurando que una sola clave pueda realizar los cambios (que sería la clave del operador o administrador de la red). La segunda es permitir un conjunto de claves. En este segundo caso, los cambios se llevarán a cabo a través de un sistema de votación, y los usuarios que tienen permitido votar deciden si se aplican o no los cambios de un proposal. Por defecto, el número de votos para aprobar o denegar un proposal es 1, pero esta también es una opción a cambiar a través del Settings TP.

Los números escritos después de “:” representan los puertos donde está ejecutándose cada proceso. Son de libre elección para el usuario, pero son los recomendados por defecto de Sawtooth, y si se deciden cambiar, habría que modificar la configuración de cada elemento que se vea afectado.

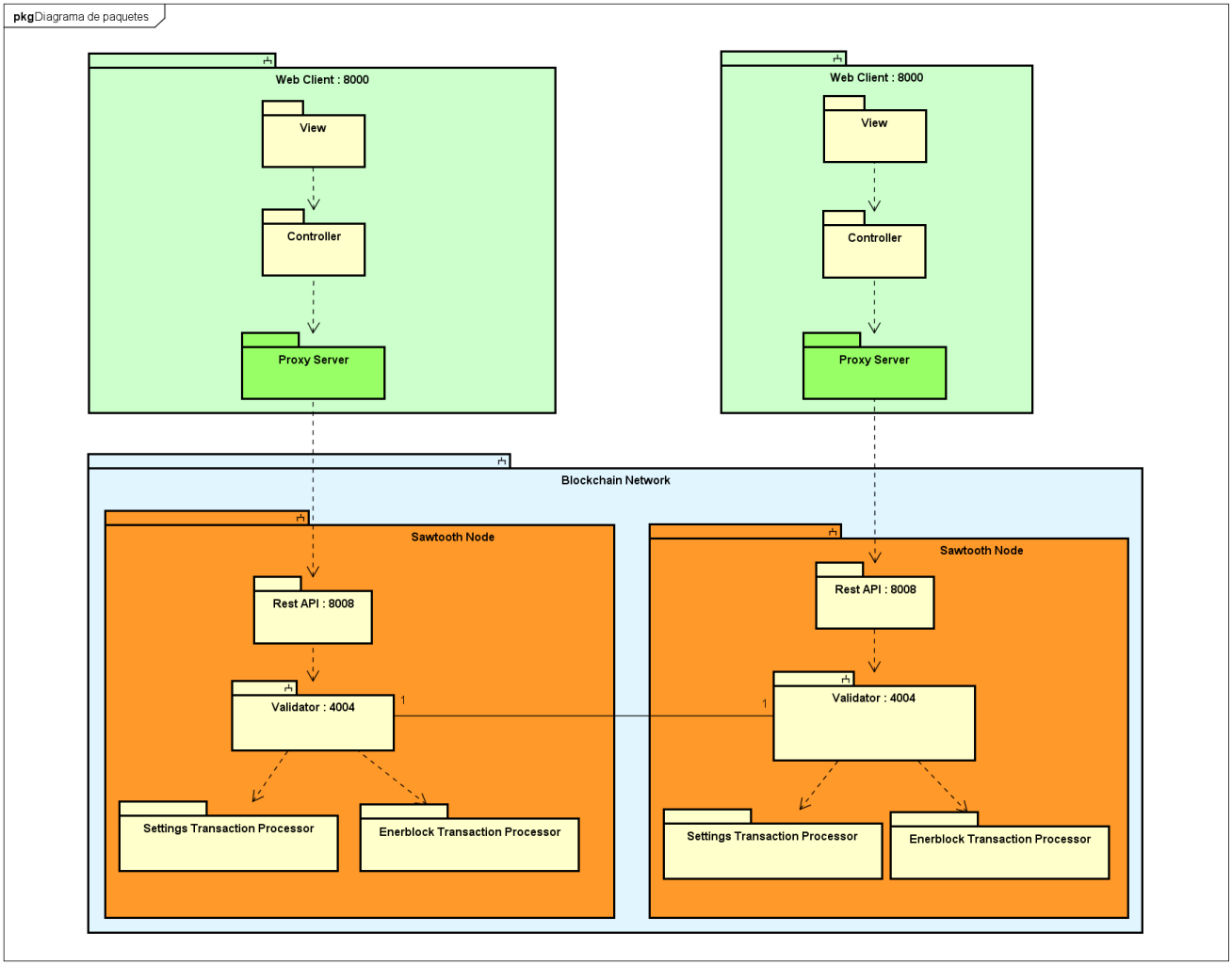


Fig. 8: Diagrama de arquitectura de alto nivel de la aplicación

12.2 Diagramas de secuencia

Esta sección se encuentra en el VI. Anexo, 19. Diagramas de secuencia.

13. Despliegue

En las primeras fases del proyecto se desplegaban los elementos del nodo manualmente, utilizando una consola por cada proceso del nodo: (cliente web, el Validator, la API y los Transaction Processors). Sin embargo, esta forma de trabajar era muy poco eficiente por el tiempo de inicio de los procesos, más la limitada portabilidad ya que las versiones estables de Sawtooth están únicamente disponibles en Ubuntu 16.04.

Tras conocer la posibilidad de Docker y sus ventajas, se migró el proceso de despliegue del sistema a Docker.

Docker es un sistema de virtualización de servicios que permite un despliegue rápido y fácil de aplicaciones, y además una gran portabilidad ya que lo único que se necesita para utilizar una aplicación con despliegue en Docker es otra máquina con Docker instalado.

De esta manera se ha mejorado potencialmente la productividad del desarrollo, y la portabilidad, siendo posible desplegar el proyecto en cualquier Ubuntu y Windows reciente.

El proceso de despliegue de Docker comienza desde el fichero docker-compose.yaml. Este fichero contiene información que utilizará Docker para crear Containers, que son los encargados de ejecutar los servicios. Desde el fichero yaml se definen los siguientes cinco contenedores:

- **enerblock-tp:** Es el servicio que ejecutará el Transaction Processor de la aplicación de intercambio de energía.
- **settings-tp:** Es el Transaction Processor para modificar opciones de la Blockchain y generación del bloque génesis, explicado en el Apartado 12.1.
- **my-web-pag:** Este servicio ejecutará a la vez el cliente web, y el proxy.
- **sawtooth-rest-api:** El servicio de la API REST de Sawtooth, explicada en el Apartado 12.1.
- **validator:** Es el elemento central del nodo, y es el encargado de intercomunicar los cuatro servicios anteriores.

Además de que los contenedores de Docker pueden ser pausados y reanudados a elección, manteniendo el estado que tenían al pausarse. También pueden ser eliminados, borrando así toda la información que podían tener.

Más información sobre el despliegue en el documento Guía de instalación.

14. Guía de instalación

Para poder instalar y desplegar el proyecto, son necesarios los siguientes requisitos:

- Docker
- Docker Compose
- (Opcional) Git

Una visualización gráfica del árbol de directorios del proyecto es la siguiente:

```
./
├── client
│   ├── Dockerfile
│   └── public
│       ├── dist
│       │   └── bundle.js
│       ├── favicon.png
│       ├── index.html
│       ├── keys
│       ├── reloadbundle.sh
│       ├── src
│       │   ├── main.js
│       │   └── state.js
│       └── node_modules
├── docker-compose.yaml
├── processor
│   ├── Dockerfile
│   ├── enerblock_processor
│   │   ├── enerblock_payload.py
│   │   ├── enerblock_state.py
│   │   ├── handler.py
│   │   ├── main.py
│   └── enerblock-tp
└── README.md
```

En el directorio raíz tenemos las dos carpetas principales del proyecto: El cliente y el Transaction Processor. También hay otros archivos como un README y el fichero docker-compose.yaml. El propósito de este último fichero es construir y desplegar todos los servicios en contenedores de Docker.

14.1 Instalación en un sistema Linux

Docker está disponible en una variedad de plataformas Linux. Se recomienda el uso de Ubuntu 16.04, ya que es la única versión que permite el desarrollo de la última versión estable de Sawtooth, pero en el caso de desplegar el proyecto no es un requisito necesario y puede utilizarse también Ubuntu 18.04. En Linux es necesario instalar:

- Docker versión 17.03.1-ce o superior.
- Docker Compose versión 1.9.0 o superior.

Se instalará también curl para facilitar la descarga, a través del siguiente comando:

```
sudo apt install curl
```

Para comprobar que se ha instalado curl correctamente:

```
curl -V
```

Hay múltiples opciones para descargar Docker, pero la más sencilla es usar el script original:

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

```
sudo sh get-docker.sh
```

Por defecto, solo el usuario root puede usar Docker. Para utilizarlo como un usuario no root, se procede a crear un grupo para utilizar Docker e introducir al usuario actual.

```
sudo groupadd Docker
```

```
sudo usermod -aG Docker $USER
```

Para comprobar que se ha realizado todo correctamente, ejecutar una imagen de prueba de Docker:

```
Docker run hello-world
```

El siguiente paso es instalar Docker Compose, una herramienta de Docker para facilitar la creación y gestión de múltiples containers simultáneamente.

```
sudo apt update
```

```
sudo apt install docker-compose
```

Finalmente, para comprobar que Docker y Docker Compose se han instalado correctamente:

```
docker --version && docker-compose --version
```

Hasta aquí, la instalación de requisitos en Linux. Para ejecutar el proyecto, es necesario descargarlo de git. Se puede acceder directamente a la página, descargar el proyecto como zip y descomprimirlo, o utilizar git. Para instalarlo:

```
sudo apt-get install git
```

Para descargar el proyecto a través de git:

```
git clone https://github.com/elrito96/TFG_enerblock_saw.git
```

Una vez descargado el proyecto, es necesario acceder a la carpeta client, y modificar el archivo Dockerfile en este directorio:

```
FROM httpd:2.4



RUN echo "\

LoadModule proxy_module modules/mod_proxy.so\n\

LoadModule proxy_http_module modules/mod_proxy_http.so\n\

ProxyPass /api http://localhost:8008\n\

ProxyPassReverse /api http://localhost:8008\n\

RequestHeader set X-Forwarded-Path \"/api\"\n\

" >>/usr/local/apache2/conf/httpd.conf
```

Es necesario modificar los “localhost” resaltados y sustituirlos por la dirección IP dentro de la red local a la que se esté conectada la máquina. En Linux se puede conocer esta dirección, como muestra la Fig. 9, utilizando el comando:

```
hostname -I
```

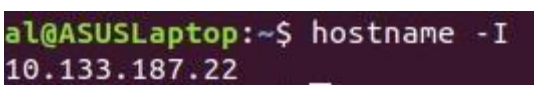


Fig. 9: Comando para obtener la dirección IP local en Linux

Una vez realizada esta modificación, el sistema está listo para ejecutarse. Acceder al directorio del proyecto, donde se encuentran las carpetas de client, processor, README, gitpush.sh y docker-compose.yaml. Ejecutar el siguiente comando:

```
docker-compose up
```

Una vez ejecutado este comando, el proyecto estará ejecutándose. Para acceder al cliente basta con abrir un navegador web y acceder a la página localhost:8000. Para pausar la ejecución, presionar control + C en la consola donde se está ejecutando el comando “docker-compose up”. Para eliminar los contenedores, y junto a ellos, la información en la Blockchain, ejecutar el comando:

```
docker-compose down
```

14.2 Instalación en un sistema Windows

La versión de Docker a instalar en Windows se llama Docker Desktop, e incluye tanto Docker como Docker Compose. Para descargar Docker Desktop, es necesario acceder al siguiente enlace: https://www.docker.com/products/docker-desktop, registrarse en Docker, descargar el instalador y ejecutarlo siguiendo las instrucciones del asistente de instalación (la mayoría son pulsar “Siguiente").

Una vez instalado, es posible que sea necesario activar la virtualización del sistema a través de la BIOS. Cada pc lo hace de diferente manera, en el caso de ser necesario consultar Google.

Después de instalar Docker Desktop, hay que descargar el proyecto a través del enlace: https://github.com/elrito96/TFG_enerblock_saw.git. Una vez descargado, hay que modificar el fichero como se ha comentado en el apartado anterior. En Windows, la forma de obtener tu IP local es abrir la consola de comandos, o PowerShell y ejecutar el siguiente comando:

```
ipconfig
```

Un ejemplo en la máquina de prueba se encuentra en la Fig. 10.

Una vez realizada esta modificación, el sistema está listo para ejecutarse. Acceder al directorio del proyecto, donde se encuentran las carpetas de client, processor, README, gitpush.sh y docker-compose.yaml. Ejecutar el siguiente comando mediante la consola Windows PowerShell:

```
docker-compose up
```

Una vez ejecutado este comando, el proyecto estará ejecutándose. Para acceder al cliente basta con abrir un navegador web y acceder a la página localhost:8000. Para pausar la ejecución, presionar control + C en la consola donde se está ejecutando el comando “docker-compose up”. Para eliminar los contenedores, y junto a ellos, la información en la Blockchain, ejecutar el comando:

```
docker-compose down
```

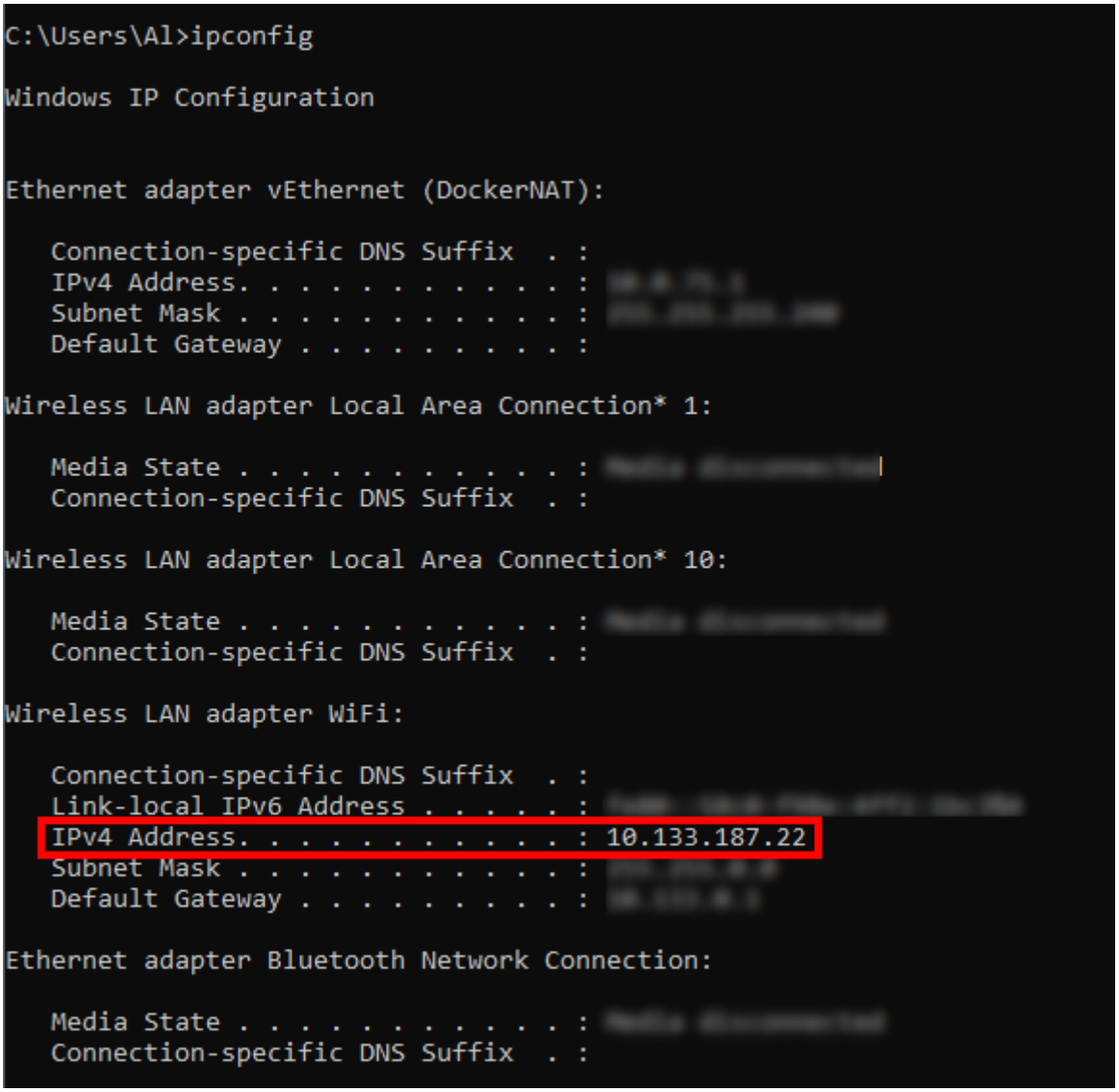


Fig. 10: Comando para obtener la dirección IP local en Windows

Tanto en Linux como en Windows, una vez se tienen instalados todos los requisitos, el proyecto, y se ha llegado a ejecutar una vez con el comando “docker-compose up”, para ejecutar el proyecto una vez más solo basta con volver a ejecutar este comando de nuevo.

Si se utiliza el servicio de Portainer, que ofrece una interfaz gráfica del estado de los containers de Docker, se puede observar los contenedores con los servicios (Fig. 11) que se han explicado previamente en el Apartado 13.

Name	State Filter	Quick actions	Stack	Image	Created	IP Address	Published Ports
web-client-proxy	running		tfg_enerblock_saw-master	enerblock-client	2020-01-09 11:06:44	172.21.0.6	8000:80
sawtooth-settings-tp	running		tfg_enerblock_saw-master	hyperledger/sawtooth-settings-tp:1.0	2020-01-09 11:06:42	172.21.0.3	-
enerblock-python-tp	running		tfg_enerblock_saw-master	enerblock-tp-image	2020-01-09 11:06:42	172.21.0.4	-
sawtooth-rest-api	running		tfg_enerblock_saw-master	hyperledger/sawtooth-rest-api:1.0	2020-01-09 11:06:42	172.21.0.5	8008:8008
sawtooth-validator	running		tfg_enerblock_saw-master	hyperledger/sawtooth-validator:1.0	2020-01-09 11:06:41	172.21.0.2	4004:4004

Fig. 11: Esquema de contenedores de Docker utilizando Portainer

Otra forma, menos visual (Fig. 12), es ejecutar el comando:

```
docker ps
```

```
PS C:\Users\Al> docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS
113c2c595fb2        enerblock-client                     "httpd-foreground"       2 weeks ago         Up 6 minutes        0.0.0.0:8000->80/tcp
9041785b6698        hyperledger/sawtooth-settings-tp:1.0 "settings-tp -v --co…"   2 weeks ago         Up 6 minutes        4004/tcp
baf902e2a5bd        enerblock-tp-image                   "/bin/sh -c 'python3…"   2 weeks ago         Up 6 minutes        4004/tcp
0bcac8a27584        hyperledger/sawtooth-rest-api:1.0    "sawtooth-rest-api -…"   2 weeks ago         Up 6 minutes        4004/tcp, 0.0.0.0:8008->8008/tcp
776fc5eaedf0        hyperledger/sawtooth-validator:1.0   "bash -c '\n  if [ ! …"   2 weeks ago         Up 6 minutes        0.0.0.0:4004->4004/tcp
0d70e7747c03        portainer/portainer:latest           "/portainer"             2 months ago        Up 11 minutes       0.0.0.0:9000->9000/tcp
```

Fig. 12: Esquema de contenedores de Docker utilizando el comando "docker ps"

15. Manual de usuario

Primero de todo vamos a observar la página inicial de la aplicación en la Fig. 13.

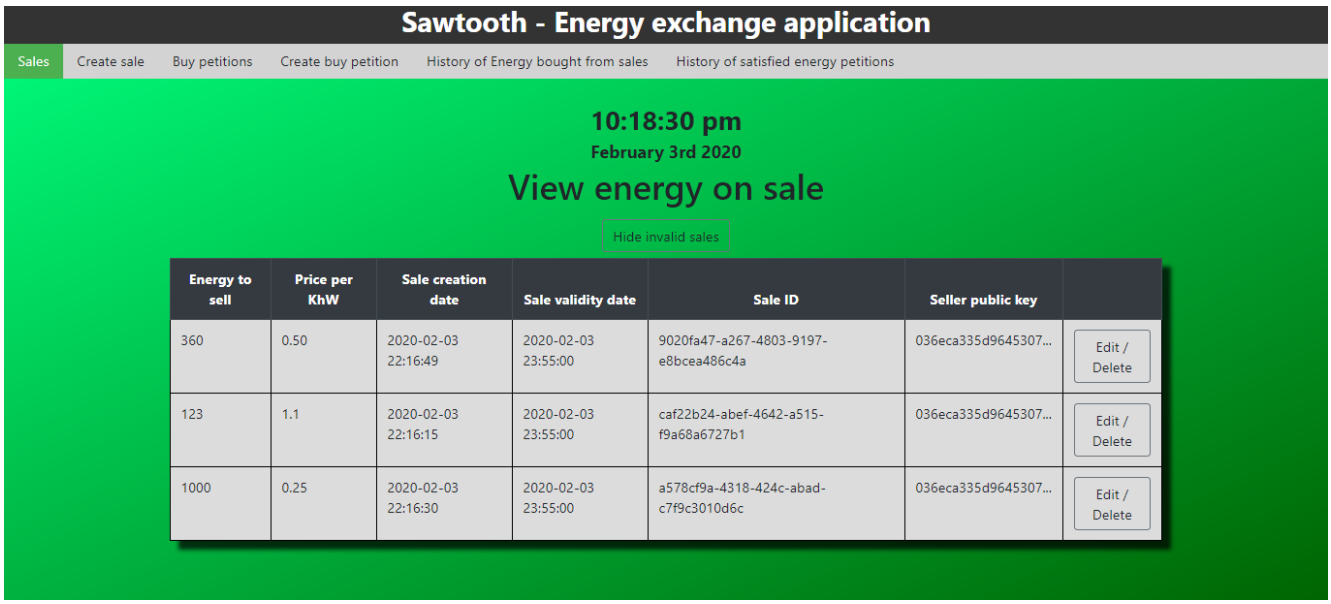


Fig. 13: Página principal de la aplicación

En el menú superior (Fig. 14) se puede observar que hay seis pestañas, cada una lleva a una página que tiene diferentes funcionalidades.

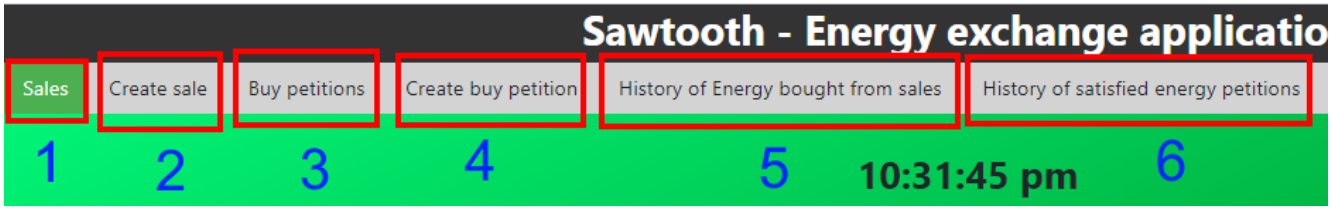


Fig. 14: Menu superior de la aplicación

1. **Sales:** Muestra las ventas de energía actuales en el sistema.
2. **Create sale:** Página que permite crear una venta de energía.
3. **Buy petitions:** Muestra las peticiones de energía actuales en el sistema.
4. **Create buy petition:** Página que permite crear una petición de energía en el sistema.
5. **History of Energy bought from sales:** Muestra el historial de intercambios de energía realizados a través de ventas de energía.
6. **History of satisfied energy petitions:** Muestra el historial de intercambios de energía realizados a través de peticiones de energía.

15.1 Página Sales

Fig. 15: Página que muestra las ventas de energía

Esta página (Fig. 15) muestra las ventas de energía en el sistema. Se puede pulsar el botón “Hide invalid sales”, que ocultará las ventas de energía que ofrezcan una cantidad de 0, o cuya fecha de validez haya superado a la hora y fecha actuales.

Si se hace click en la zona rosada de una fila de la fig, se abrirá el modal que permitirá comprar energía de esa venta (Fig. 16). El usuario que desee comprar energía debe introducir la cantidad que desea comprar, su clave privada y generar un ID de la compra pulsando el botón “Click to gen buy ID”. Finalmente presionará al botón “Buy”.

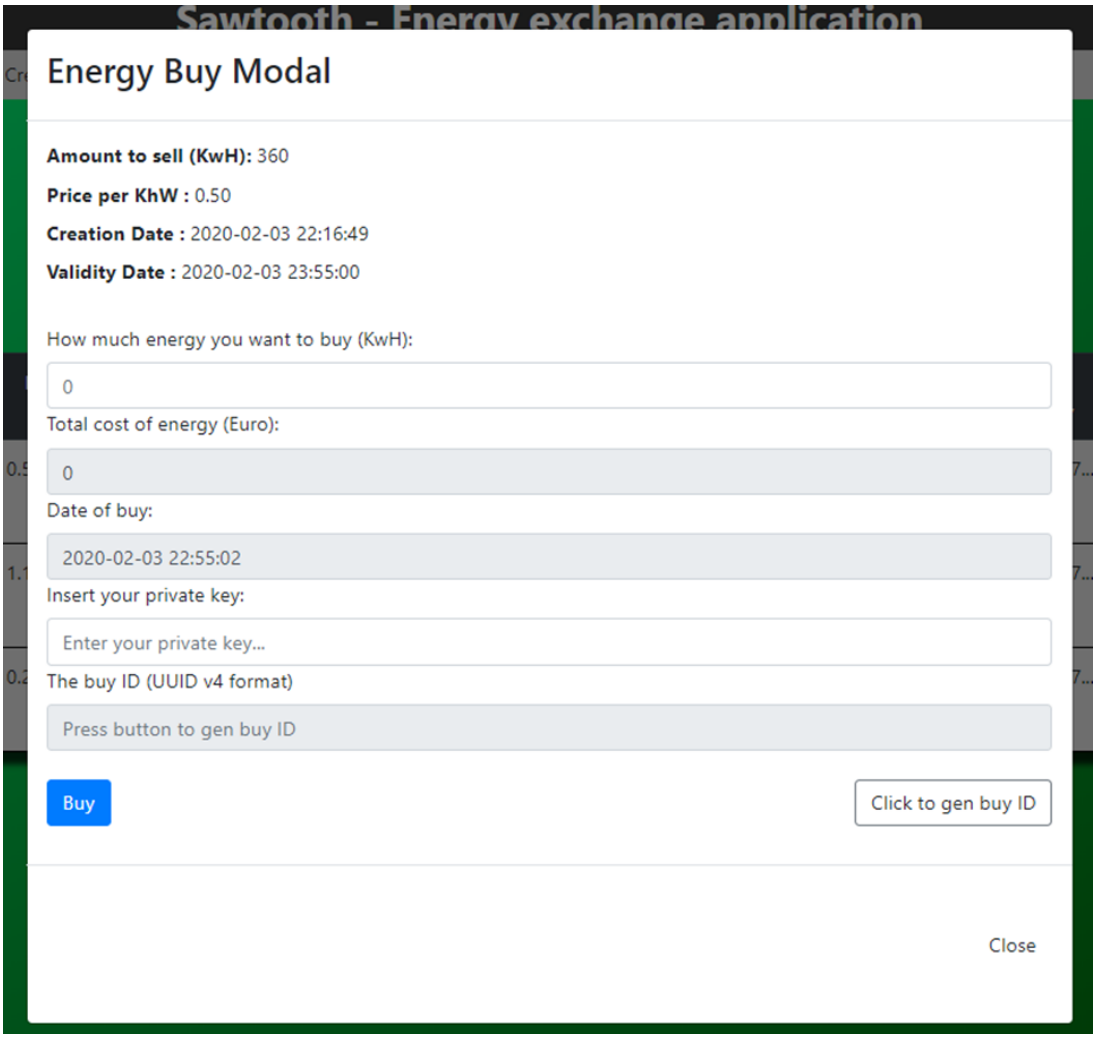


Fig. 16: Modal de compra de energía

Si se hace click en el botón resaltado en azul de la fig, “Edit / Delete”, se abrirá el modal que permitirá modificar la cantidad de energía o el precio de una venta, pulsando en el botón “Save Changes”, así como eliminarla, con el botón “Delete Sale” (Fig. 17).

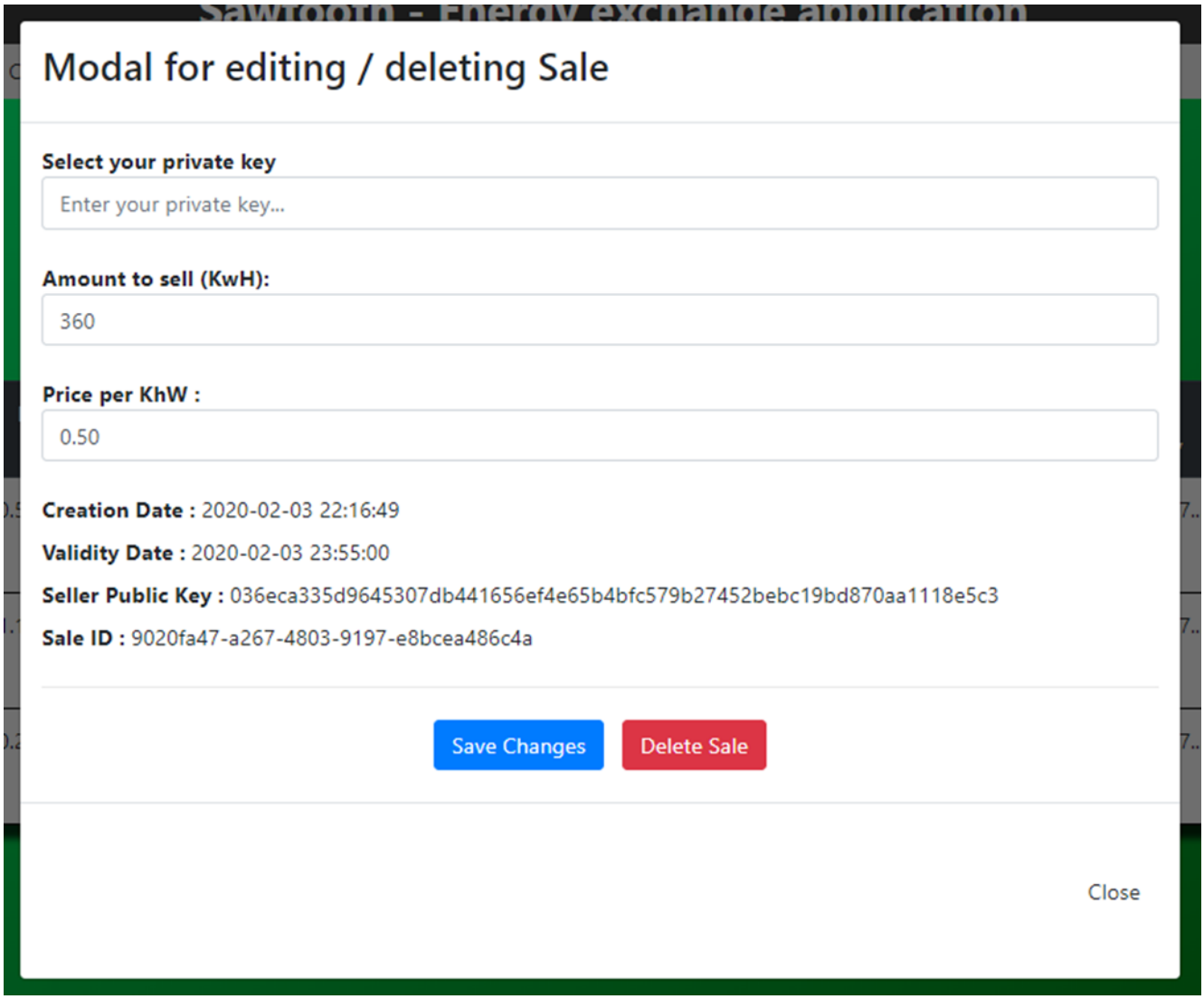


Fig. 17: Modal de edición / eliminación de una venta de energía

15.2 Página Create Sell

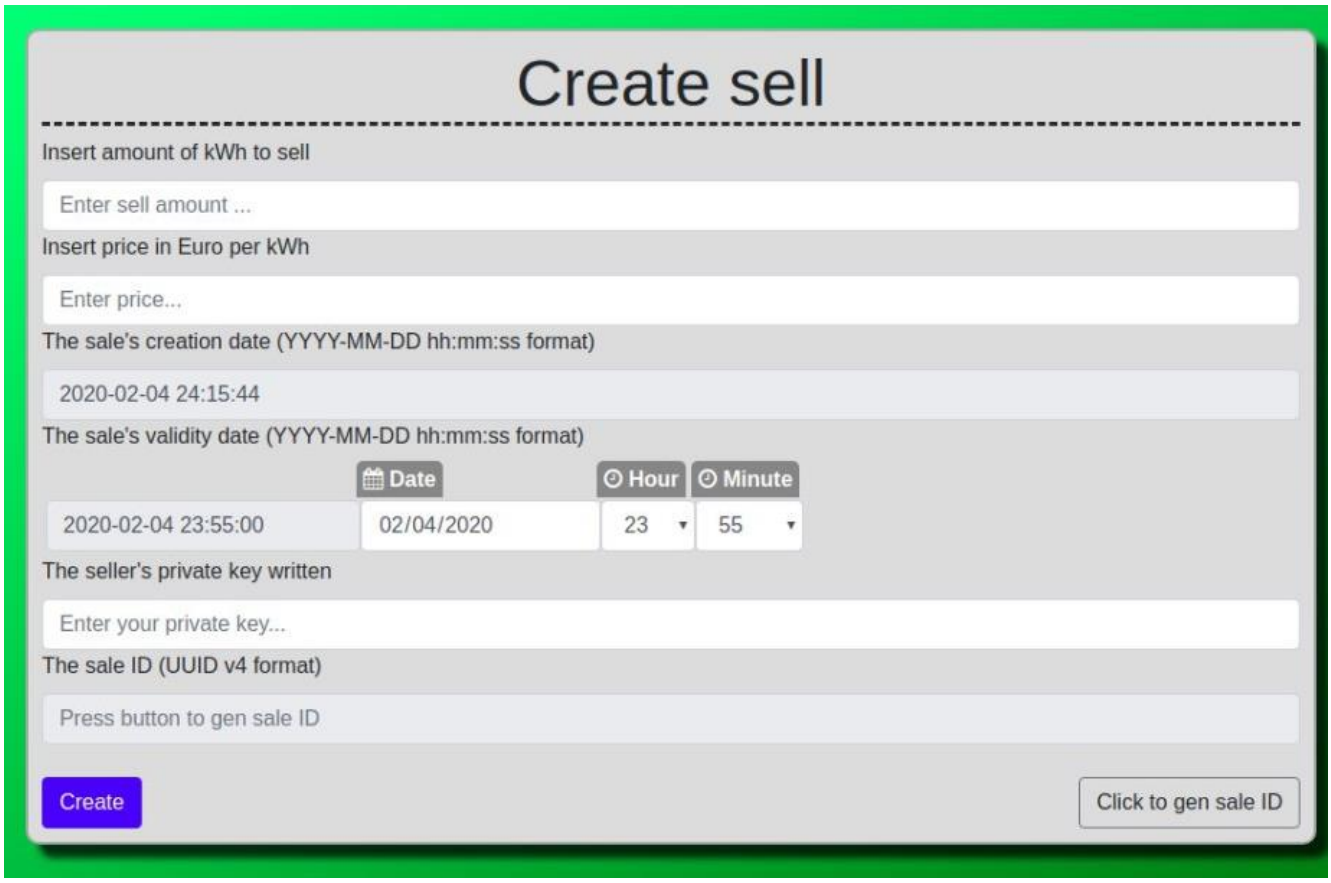


Fig. 18: Página de creación de venta de energía

En esta página (Fig. 18) se permite crear una venta de energía, en la que el usuario deberá introducir la cantidad de energía que desea vender, el precio, la fecha de validez, indicando día, hora y minuto, su clave privada, y generará un ID de la venta haciendo click en el botón “Click to gen sale ID”. Finalmente hará click en el botón “Create”.

15.3 Página Buy Petitions

Similar a la página Sales (Apartado 15.1). Esta página (Fig. 19) muestra las peticiones de compra en el sistema. Si se hace click en la fila, se abrirá el modal que permite a un Prosumer ofrecer energía a un usuario para satisfacer su petición (Fig. 20). Si se hace click en el botón “Edit / Delete”, se podrán modificar la cantidad de energía que se solicita y su precio, o se podrá eliminar la petición del sistema (Fig. 21).

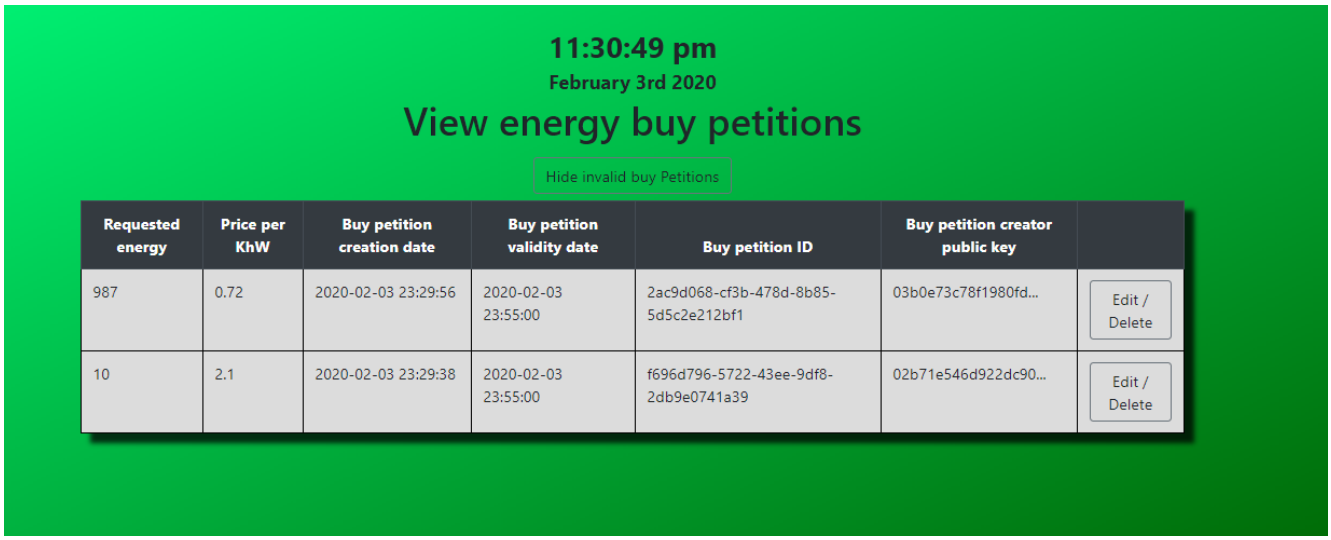


Fig. 19: Página que muestra las peticiones de energía

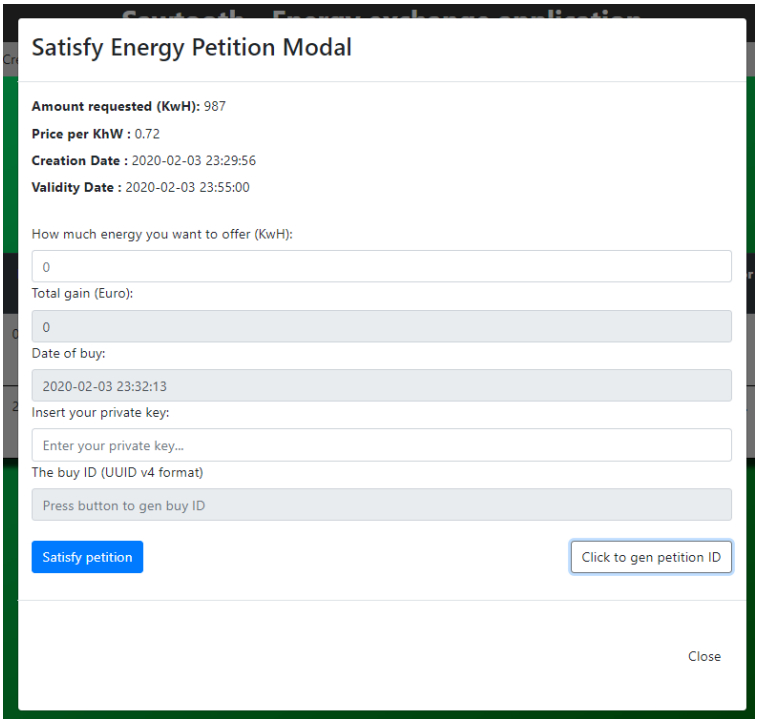


Fig. 20: Modal de satisfacer una petición de energía

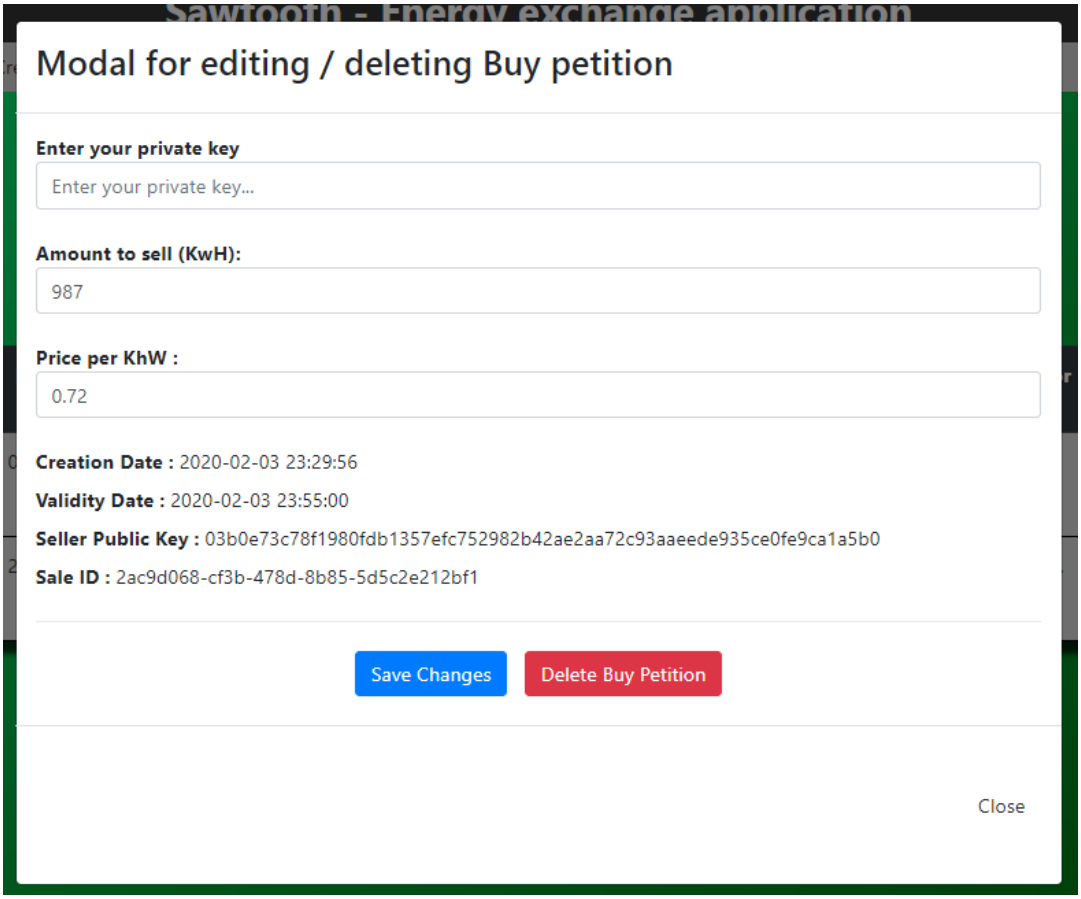


Fig. 21: Modal de edición / eliminación de petición de energía

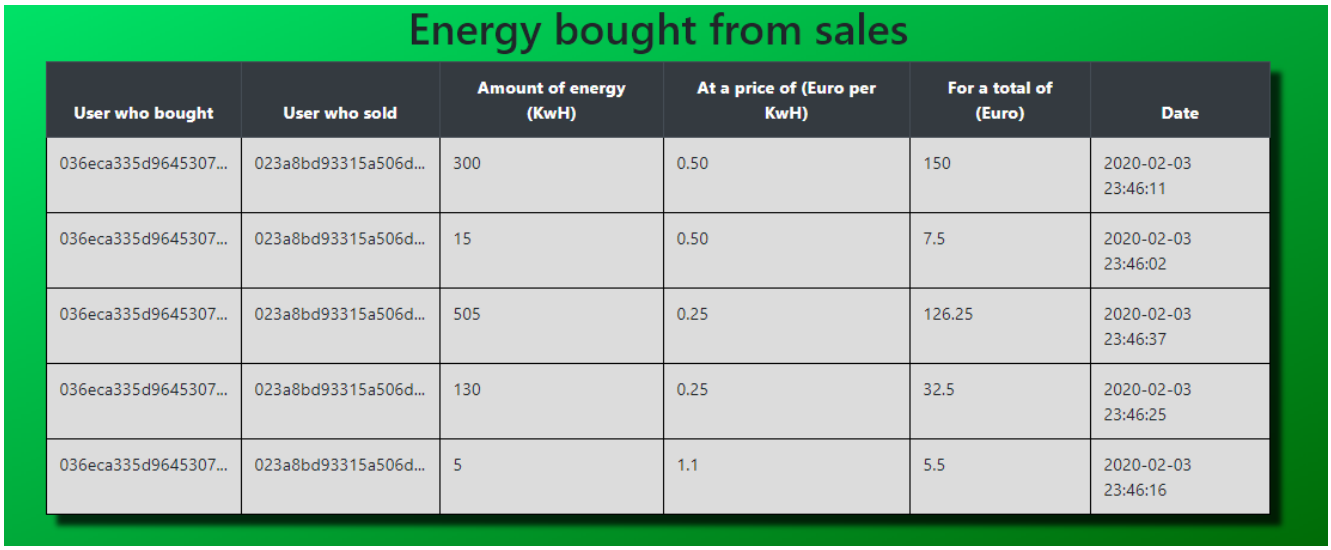
15.4 Página Create buy petition

Similar a la página Create sell (Apartado 15.2). Esta página (Fig. 22) permite crear una petición de energía, en la que el usuario deberá introducir la cantidad de energía que solicita, el precio que pagará a cambio de la energía, la fecha de validez, indicando día, hora y minuto, su clave privada, y generará un ID de la petición de energía haciendo click en el botón “Click to gen buy petition ID”. Finalmente hará click en el botón “Create”.

Fig. 22: Página de creación de petición de energía

15.5 Páginas de History of Energy bought from sales y satisfied energy petitions.

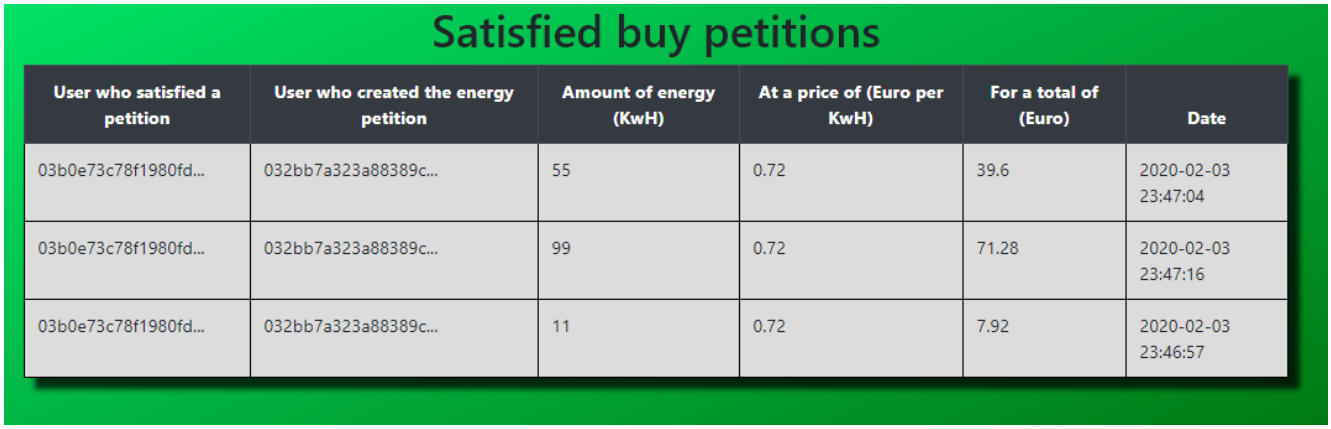
En estas dos páginas, el funcionamiento es muy similar. La página de History of Energy bought from sales (Fig. 23) muestra el historial de compras de usuarios de ventas de energía, junto con las claves públicas del comprador, vendedor, la cantidad, el precio y el momento en el que se realizó la compra.



User who bought	User who sold	Amount of energy (KwH)	At a price of (Euro per KwH)	For a total of (Euro)	Date
036eca335d9645307...	023a8bd93315a506d...	300	0.50	150	2020-02-03 23:46:11
036eca335d9645307...	023a8bd93315a506d...	15	0.50	7.5	2020-02-03 23:46:02
036eca335d9645307...	023a8bd93315a506d...	505	0.25	126.25	2020-02-03 23:46:37
036eca335d9645307...	023a8bd93315a506d...	130	0.25	32.5	2020-02-03 23:46:25
036eca335d9645307...	023a8bd93315a506d...	5	1.1	5.5	2020-02-03 23:46:16

Fig. 23: Página del historial de compras de energía

La página de History of satisfied energy petitions (Fig. 24) muestra el historial de peticiones de energía satisfechas por otros usuarios, junto con las claves públicas del creador de la petición, el que ha satisfecho la petición, la cantidad, el precio y el momento en el que se realizó la acción.



User who satisfied a petition	User who created the energy petition	Amount of energy (KwH)	At a price of (Euro per KwH)	For a total of (Euro)	Date
03b0e73c78f1980fd...	032bb7a323a88389c...	55	0.72	39.6	2020-02-03 23:47:04
03b0e73c78f1980fd...	032bb7a323a88389c...	99	0.72	71.28	2020-02-03 23:47:16
03b0e73c78f1980fd...	032bb7a323a88389c...	11	0.72	7.92	2020-02-03 23:46:57

Fig. 24: Página del historial de peticiones de energía satisfechas

V. Conclusiones y trabajo futuro

16. Conclusiones

En este trabajo se ha realizado el análisis, diseño, implementación y despliegue de un sistema de compra y venta de energía basado en la tecnología Blockchain. Partiendo de los objetivos iniciales, se ha conseguido cumplir satisfactoriamente todos ellos: (1) Uso de una red Blockchain, manteniendo un registro inmutable de las transacciones realizadas; (2) Uso de Smart Contracts para agilizar los intercambios de energía sin necesidad de intermediarios; (3) Creación de una interfaz gráfica para facilitar el uso de la aplicación y poder ver los intercambios que han tenido lugar entre los usuarios. Con todo esto, mi interés por esta tecnología ha aumentado de forma exponencial, tanto que probablemente seguiré su estudio en el futuro.

También he aprendido la utilidad de usar otros sistemas como soporte, en este caso Docker, que ha facilitado en gran medida el desarrollo, portabilidad y customización del proyecto.

Dado que la legislación que permite este tipo de acciones relacionadas con la generación de energías renovables es relativamente nueva, este trabajo constituye un primer paso para la creación de multitud de sistemas de este tipo que faciliten el autoconsumo colectivo y creación de mercados de energía.

A lo largo del proyecto me he encontrado con un gran número de dificultades, la mayoría de ellas causadas por el desconocimiento de utilizar una tecnología con un paradigma bastante diferente a lo que se enseña en el Grado en Ingeniería Informática. Pero a base de buscar información por internet, preguntar a compañeros y contactar con los propios encargados de mantener el framework (Sawtooth), se ha conseguido superar todos los obstáculos hasta obtener una versión totalmente funcional.

Como he comentado anteriormente, con la realización de este trabajo no solo he reforzado mis conocimientos sobre planificación, análisis e implementación, sino que también he aprendido a resolver problemas y he adquirido una valiosa experiencia de trabajo y personal.

17. Trabajo futuro

Las sugerencias de posibles funcionalidades y cambios futuros en esta aplicación son:

- La utilización de un sistema de pago, ya sea externo, en forma de transferencias bancarias u otro sistema como Paypal; o como parte de la propia Blockchain. En el caso de ser un sistema on-chain, la idea es desarrollar otro Transaction Processor encargado de crear cuentas con “saldo” para cada usuario, y hacer que el Transaction Processor existente de intercambio de energía, compruebe también el saldo del usuario que compra, y se encargue de traspasar el saldo de una cuenta a otra.
- La inclusión de sistemas de predicción de generación y consumo de energía. Estos sistemas pueden mejorar la eficiencia de los intercambios, ya que ofrecen información sobre la cantidad de energía que se prevé que se generará y consumirá para un usuario. De esta manera se conoce de antemano la cantidad de energía excedente que se producirá y que es conveniente poner a la venta en el mercado.
- La implementación de un sistema de subastas. Este sistema se encargará de repartir la energía entre los participantes a través de un sistema de pujas. Hay múltiples posibilidades a la hora de implementar este sistema, desde subastas a ciegas, por intervalos de tiempo, o un sistema completamente automatizado, pero que debe conocer previamente la cantidad de energía que desea cada usuario y la cantidad de precio que están dispuestos a pagar. Hay que tener en cuenta que este sistema sería un EMTS (Apartado 8.6), siendo independiente del sistema Blockchain.
- En base a la legislación, añadir restricciones al sistema. Por ejemplo, no permitir a un usuario que esté a más de 500 metros del nodo más cercano participar, o añadir límites en los precios de la energía y cantidad puesta a la venta, aunque actualmente no hay legislación que prohíba estos aspectos.
- Añadir a la Blockchain los datos de lectura de los Smart Meters, que indicarán las cantidades de energía que salen y entran en una vivienda. El Smart Meter de una vivienda se comunicará con su nodo a través de la API de Sawtooth, como si de otro cliente se tratase. Si se añade, se facturará a los usuarios en base a estas lecturas.

Referencias

[1] Prinsloo G, Mammoli A, Dobson R. Customer domain supply and load coordination: A case for smart villages and transactive control in rural off-grid microgrids; 2017. p.430-441.

[2] Block C, Neumann D, Weinhardt C. A Market Mechanism for Energy Allocation in Micro-CHP Grids. In: IEEE Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008). p.1-9.

[3] Mengelkamp E, Gärttner J, Rock K, Kessler S, Orsini L, Weinhardt C. Designing Microgrid energy markets: A case study: The Brooklyn Microgrid. In: Applied Energy, vol. 210; January 2018. p. 870-880.

[4] Franke M, Rolli D, Kamper A, Dietrich A, Geyer-Schulz A, Lockemann P, et al. Impacts of distributed generation from virtual power plants. In: Proceedings of the 11th annual international sustainable development research conference, vol. 1; 2005. p. 12.

[5] Burger C, Kuhlmann A, Richard P, Weinmann J. Blockchain in the energy transition: A survey among decision-makers in the German energy industry. Berlin: Deutsche Energie-Agentur GmbH (dena); 2016. p. 13.

[6] Block C, Neumann D, Weinhardt C. A market mechanism for energy allocation in micro-chp grids. In: Proceedings of the 41st annual Hawaii international conference on system sciences.IEE; 2008. 172-172.

[7] Ilic D, Da Silva PG, Karnouskos S, Griesemer M. An energy market for trading electricity in smart grid neighbourhoods. In: 2012 6th IEEE international conference on digital ecosystems technologies (DEST). IEEE; 2012. p.1-6.

[8] Lamparter S, Becher S, Fischer J-G. An agent-based market platform for smart grids. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems; 2010. p. 1689-96.

[9] BOE-A-2018-13592

[10] https://sawtooth.hyperledger.org/faq/rest/#i-am-getting-this-error-cross-origin-request-blocked-the-same-origin-policy-disallows-reading-the-remote-resource-at-http-localhost-8008-batches-wait-reason-cors-header-access-control-allow-origin-missing

[11] http://bankymoon.co.za/

[12] https://solarcoin.org/

[13] https://www.linkedin.com/pulse/blockcharge-blockchain-based-solution-charging-cars/

[14] Haber, Stuart; Stornetta, W. Scott (January 1991). "How to time-stamp a digital document". Journal of Cryptology. 3 (2): p. 99–111.

[15] Bayer, Dave; Haber, Stuart; Stornetta, W. Scott (March 1992). Improving the Efficiency and Reliability of Digital Time-Stamping. Sequences. 2. p. 329–334

[16] https://en.wikipedia.org/wiki/SHA-2

VI. Anexo

18. Descripción de los casos de uso

18.1 CU01: Poner a la venta energía

Actor	**Prosumer**
Precondición	**-**
Postcondición	**El usuario Prosumer habrá creado una oferta de energía en el sistema**
Flujo principal	**1.** El usuario elige la opción de crear venta. **2.** El sistema muestra la información sobre los campos necesarios para poner energía a la venta. **3.** El usuario introduce los datos necesarios para crear una venta, que son: • Cantidad de energía a vender. • Precio por unidad de energía. • Fecha hasta la cual es válida la energía que pone a la venta. • ID único que identificará esta venta de energía. • Clave privada del usuario. Después de introducir los datos, el usuario selecciona la opción de crear venta. **4.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**3.a** El usuario introduce algún dato inválido, o intenta utilizar un ID ya existente. **4.a** El sistema detecta la transacción como inválida y muestra un mensaje de error.

Cuadro 6: Descripción de CU01: Poner a la venta energía

18.2 CU02: Gestionar venta de energía

Actor	Prosumer
Precondición	**El usuario Prosumer tiene que haber creado una puesta a la venta de energía**
Postcondición	**-**
Flujo principal	**1.** El usuario elige la opción de mostrar ventas de energía. **2.** El sistema muestra las ventas de energía actuales. **3.** El usuario selecciona la opción de modificar en una venta de energía creada por él. **4.** El sistema mostrará la información de la venta de energía.
Excepciones	**3.a** El usuario selecciona una puesta a la venta que no ha creado él mismo. **4.a** El sistema no le permitirá eliminarla ni guardar cambios si la modifica.

Cuadro 7: Descripción de CU02: Gestionar venta de energía

18.3 CU03: Modificar energía a la venta

Actor	Prosumer
Precondición	**Haber realizado el CU02: Gestionar venta de energía**
Postcondición	**El usuario Prosumer habrá modificado una puesta a la venta de energía**
Flujo principal	**1.** El usuario modificará la cantidad que desea vender y el precio, y se identificará. Finalmente selecciona la opción de guardar. **2.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**1.a** El usuario introduce algún dato inválido. **2.a** El sistema detecta la transacción como inválida y muestra un mensaje de error. **1.b** El usuario cancela la edición. El caso de uso termina sin efecto.

Cuadro 8: Descripción de CU03: Modificar energía a la venta

18.4 CU04: Eliminar energía a la venta

Actor	**Prosumer**
Precondición	**Haber realizado el CU02: Gestionar venta de energía**
Postcondición	**El usuario Prosumer habrá eliminado una puesta a la venta de energía**
Flujo principal	**1.** El usuario se identificará. Finalmente selecciona la opción de eliminar. **2.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**1.a** El usuario cancela la eliminación. El caso de uso termina sin efecto.

Cuadro 9: Descripción de CU04: Eliminar energía a la venta

18.5 CU05: Comprar energía a la venta

Actor	**Consumer**
Precondición	**Debe haber una petición de venta de energía válida en el sistema**
Postcondición	**El usuario Prosumer habrá comprado energía de un Prosumer**
Flujo principal	**1.** El usuario elige la opción de mostrar ventas de energía. **2.** El sistema muestra las ventas de energía actuales. **3.** El usuario selecciona la venta de energía que le interese. **4.** El sistema mostrará la información de la venta de energía. **5.** El usuario indicará la cantidad de energía que desea comprar, y se identificará. Finalmente selecciona la opción de comprar. **6.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**5.a** El usuario introduce algún dato inválido, o intenta utilizar un ID ya existente. **6.a** El sistema detecta la transacción como inválida y muestra un mensaje de error.

Cuadro 10: Descripción de CU05: Comprar energía a la venta

18.6 CU06: Realizar solicitud de energía

Actor	**Consumer**
Precondición	**-**
Postcondición	**El usuario Consumer habrá creado una petición de energía**
Flujo principal	**1.** El usuario elige la opción de crear solicitud de energía. **2.** El sistema muestra la información sobre los campos necesarios para crear una petición de energía. **3.** El usuario introduce los datos necesarios para crear una petición de energía, que son: • Cantidad de energía que desea comprar. • Precio por unidad de energía. • Fecha hasta la cual es válida la petición de energía. • ID único que identificará esta petición de energía. • Clave privada del usuario. Después de introducir los datos, el usuario selecciona la opción de crear petición de energía. **4.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**3.a** El usuario introduce algún dato inválido, o intenta utilizar un ID ya existente. **4.a** El sistema detecta la transacción como inválida y muestra un mensaje de error.

Cuadro 11: Descripción de CU06: Realizar solicitud de energía

18.7 CU07: Gestionar solicitud de energía

Actor	**Consumer**
Precondición	**El usuario Consumer tiene que haber creado una solicitud de energía**
Postcondición	**-**
Flujo principal	**1.** El usuario elige la opción de mostrar peticiones de energía. **2.** El sistema muestra las solicitudes de energía actuales. **3.** El usuario selecciona la opción de modificar en una petición de energía creada por él. **4.** El sistema mostrará la información de la petición de energía.
Excepciones	**3.a** El usuario selecciona una petición de energía que no ha creado él mismo. **4.a** El sistema no le permitirá eliminarla ni guardar cambios si la modifica.

Cuadro 12: Descripción de CU07: Gestionar solicitud de energía

18.8 CU08: Modificar solicitud de energía

Actor	Consumer
Precondición	**Haber realizado el CU07: Gestionar solicitud de energía**
Postcondición	**El usuario Prosumer habrá modificado una petición de energía**
Flujo principal	**1.** El usuario modificará la cantidad que desea comprar y el precio, y se identificará. Finalmente selecciona la opción de guardar. **2.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**1.a** El usuario introduce algún dato inválido. **2.a** El sistema detecta la transacción como inválida y muestra un mensaje de error. **1.b** El usuario cancela la edición. El caso de uso termina sin efecto.

Cuadro 13: Descripción de CU08: Modificar solicitud de energía

18.9 CU09: Eliminar solicitud de energía

Actor	Consumer
Precondición	**Haber realizado el CU07: Gestionar solicitud de energía**
Postcondición	**El usuario Consumer habrá eliminado una solicitud de energía**
Flujo principal	**1.** El usuario se identificará. Finalmente selecciona la opción de eliminar. **2.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**1.a** El usuario cancela la eliminación. El caso de uso termina sin efecto.

Cuadro 14: Descripción de CU09: Eliminar solicitud de energía

18.10 CU010: Satisfacer solicitud de energía

Actor	Prosumer
Precondición	Debe haber una solicitud de energía válida en el sistema
Postcondición	El usuario Prosumer habrá satisfecho la petición de energía de un Consumer
Flujo principal	**1.** El usuario elige la opción de mostrar solicitudes de energía. **2.** El sistema muestra las solicitudes de energía actuales. **3.** El usuario selecciona la solicitud de energía que le interese. **4.** El sistema mostrará la información de la solicitud de energía. **5.** El usuario indicará la cantidad de energía que desea contribuir a la solicitud, y se identificará. Finalmente selecciona la opción de satisfacer solicitud. **6.** El sistema valida los datos, añade un bloque a la Blockchain con la información y actualiza el estado global.
Excepciones	**5.a** El usuario introduce algún dato inválido, o intenta utilizar un ID ya existente. **6.a** El sistema detecta la transacción como inválida y muestra un mensaje de error.

Cuadro 15: Descripción de CU10: Satisfacer solicitud de energía

18.11 CU11: Acceder a resumen informativo de intercambios de energía

Actor	Consumer
Precondición	-
Postcondición	El usuario Consumer habrá accedido a un resumen informativo sobre los intercambios de energía producidos en el sistema
Flujo principal	**1.** El usuario elige la opción de mostrar historial de energía comprada. **2.** El sistema muestra información sobre las compras de energía a la venta, que será creador de la venta, comprador, cantidad de energía comprada, precio y fecha en la que se ha realizado el intercambio.
Flujo alternativo	**1.** El usuario elige la opción de mostrar historial de peticiones de energía satisfechas. **2.** El sistema muestra información sobre las peticiones de energía satisfechas, que será creador de la petición de energía, usuario que la satisface, cantidad de energía enviada, precio y fecha en la que se ha realizado el intercambio

Cuadro 16: Descripción de CU11: Acceder a resumen informativo de intercambios de energía

19. Diagramas de secuencia

19.1 Diagrama de secuencia CU01: Poner a la venta energía

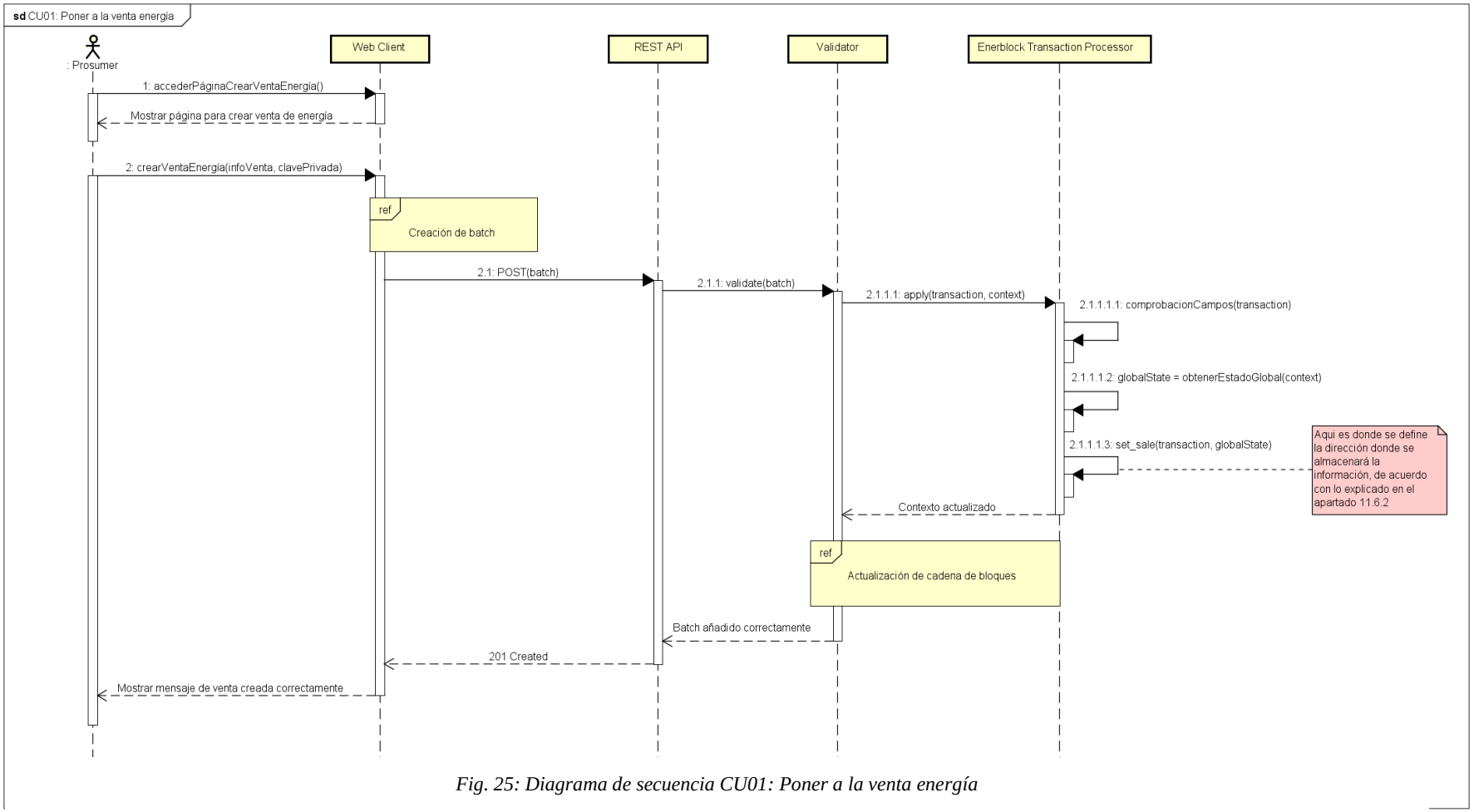


Fig. 25: Diagrama de secuencia CU01: Poner a la venta energía

Los diagramas referenciados se encuentran en el Anexo: Fig. 36 y Fig. 37.

19.2 Diagrama de secuencia CU02: Gestionar venta de energía

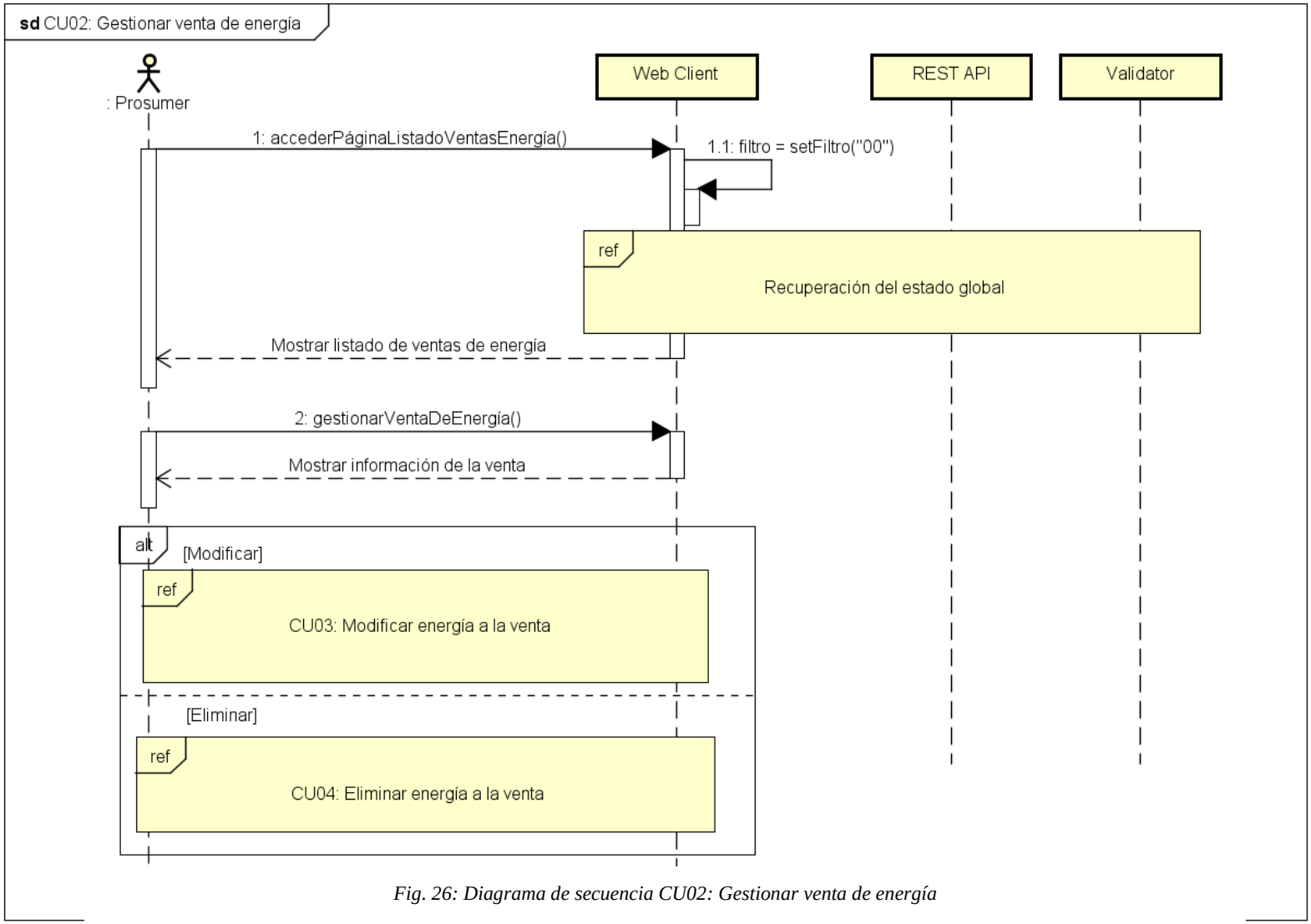


Fig. 26: Diagrama de secuencia CU02: Gestionar venta de energía

Los diagramas referenciados son: Fig. 27, Fig. 28 y Anexo: Fig. 38.

19.3 Diagrama de secuencia CU03: Modificar energía a la venta de energía

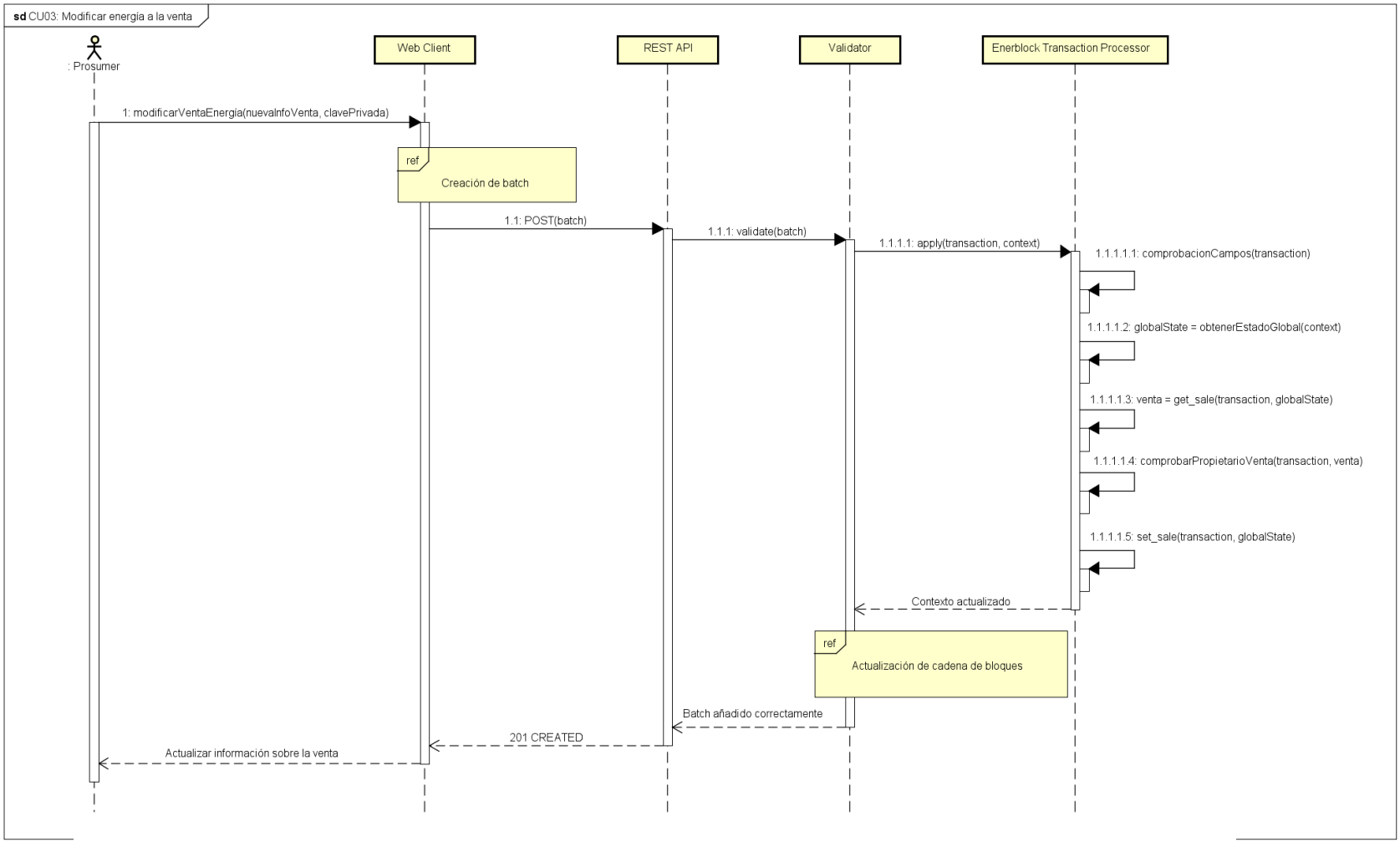


Fig. 27: Diagrama de secuencia CU03: Modificar energía a la venta

Los diagramas referenciados se encuentran en el Anexo: Fig. 36 y Fig. 37.

19.4 Diagrama de secuencia CU04: Eliminar energía a la venta

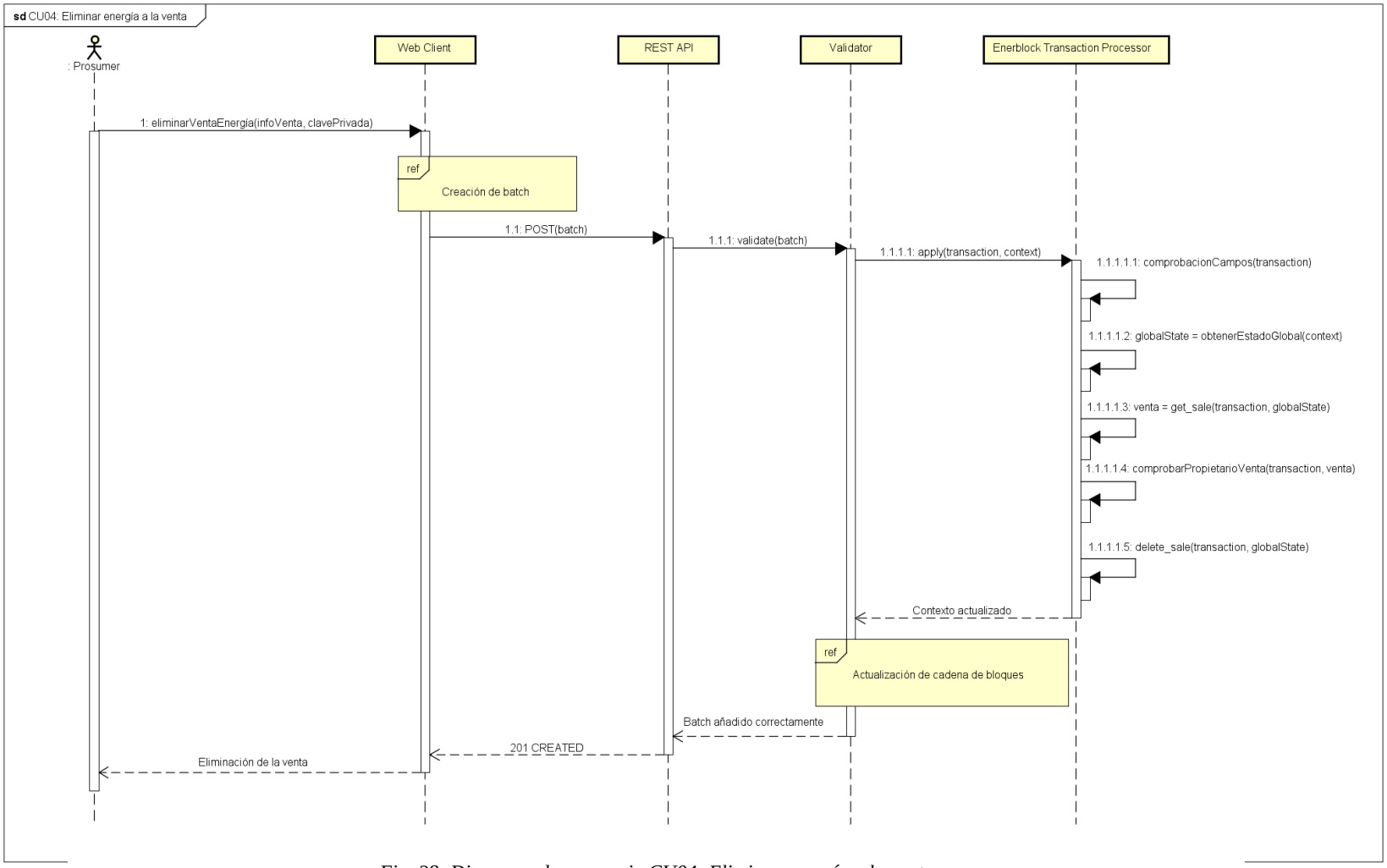


Fig. 28: Diagrama de secuencia CU04: Eliminar energía a la venta

Los diagramas referenciados se encuentran en el Anexo: Fig. 36 y Fig. 37.

19.5 Diagrama de secuencia CU05: Comprar energía a la venta

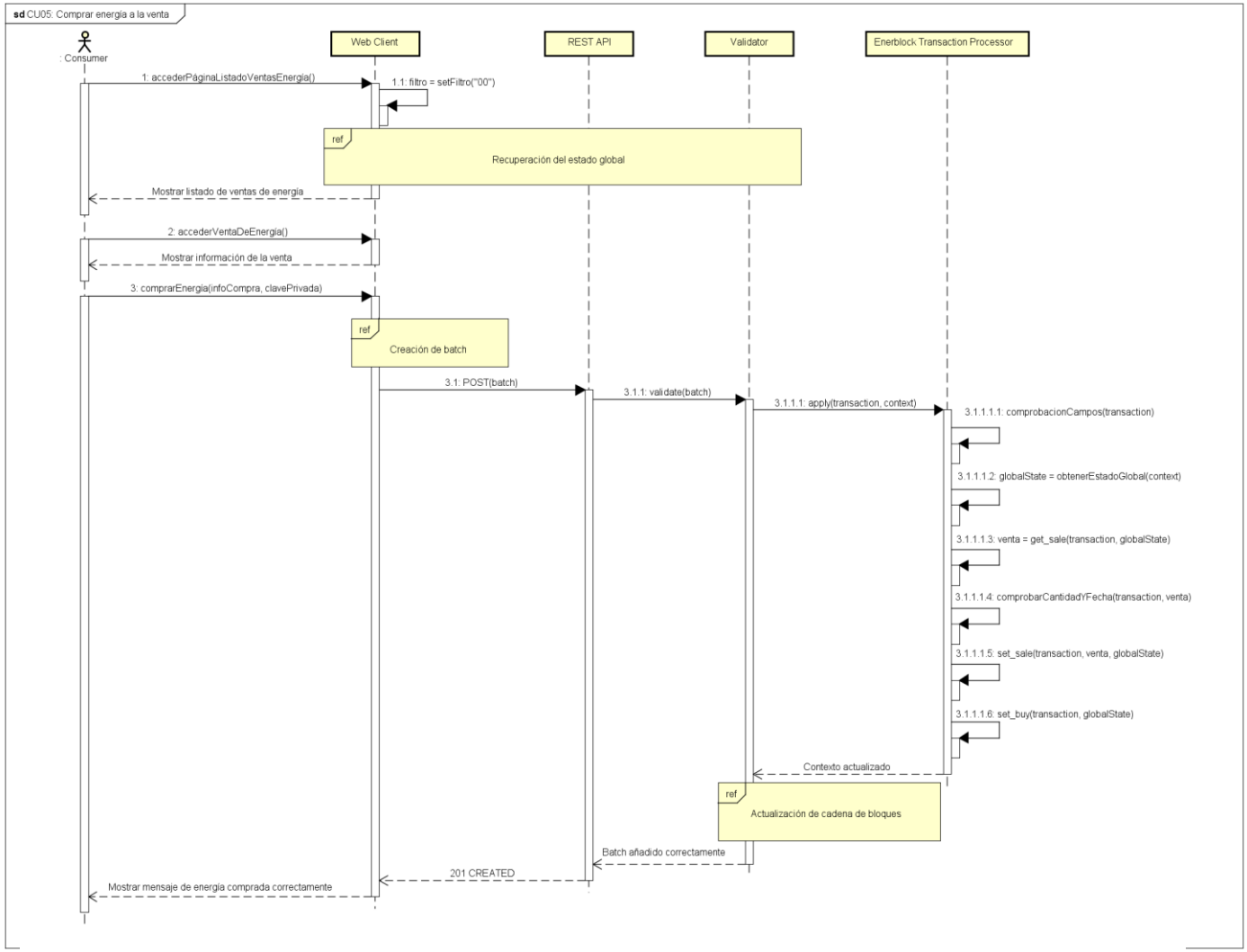


Fig. 29: Diagrama de secuencia CU05: Comprar energía a la venta

Los diagramas referenciados se encuentran en el Anexo: Fig. 36, Fig. 37 y Fig. 38.

19.6 Diagrama de secuencia CU06: Realizar solicitud de energía

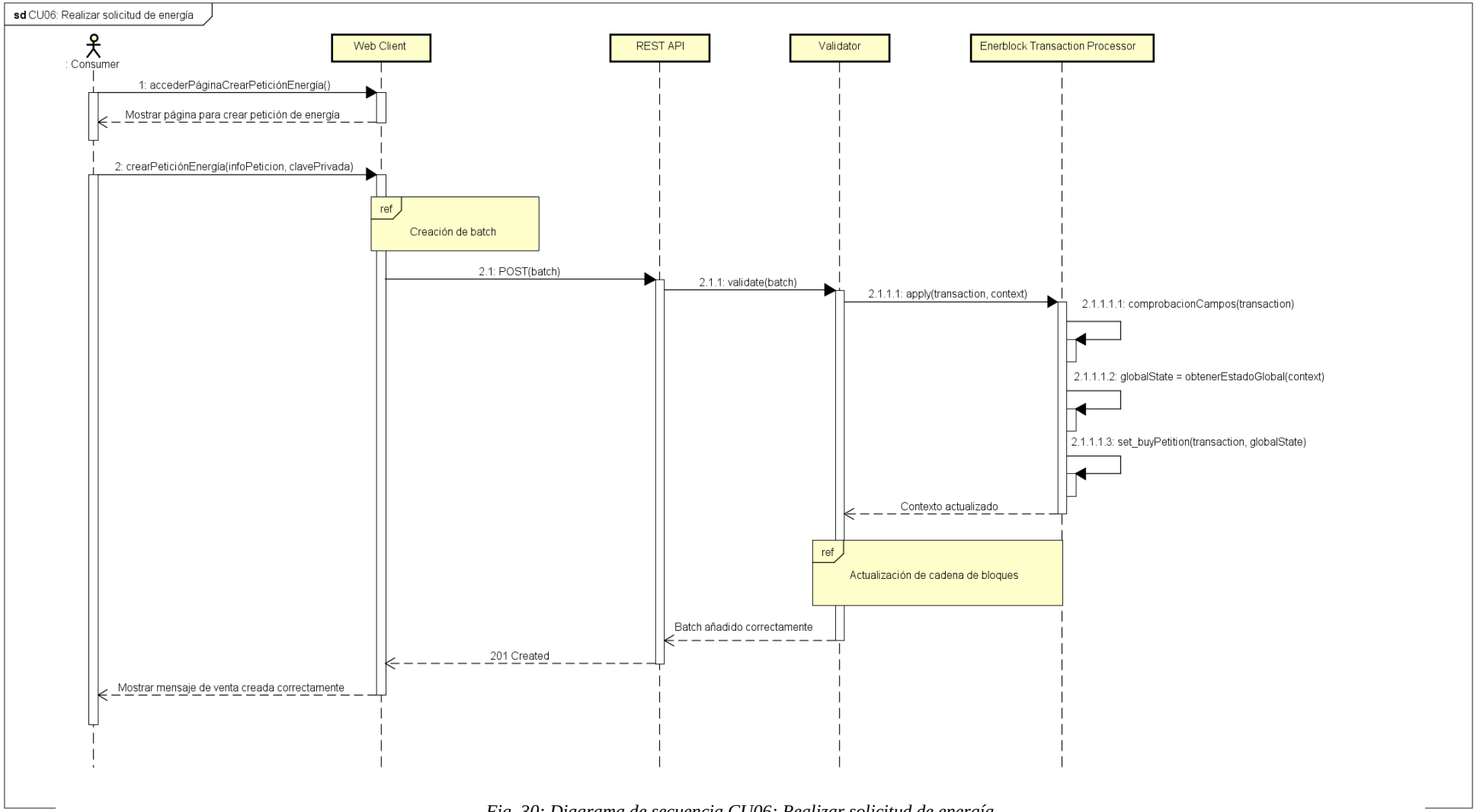


Fig. 30: Diagrama de secuencia CU06: Realizar solicitud de energía

Los diagramas referenciados se encuentran en el Anexo: Fig. 36 y Fig. 37.

19.7 Diagrama de secuencia CU07: Gestionar solicitud de energía

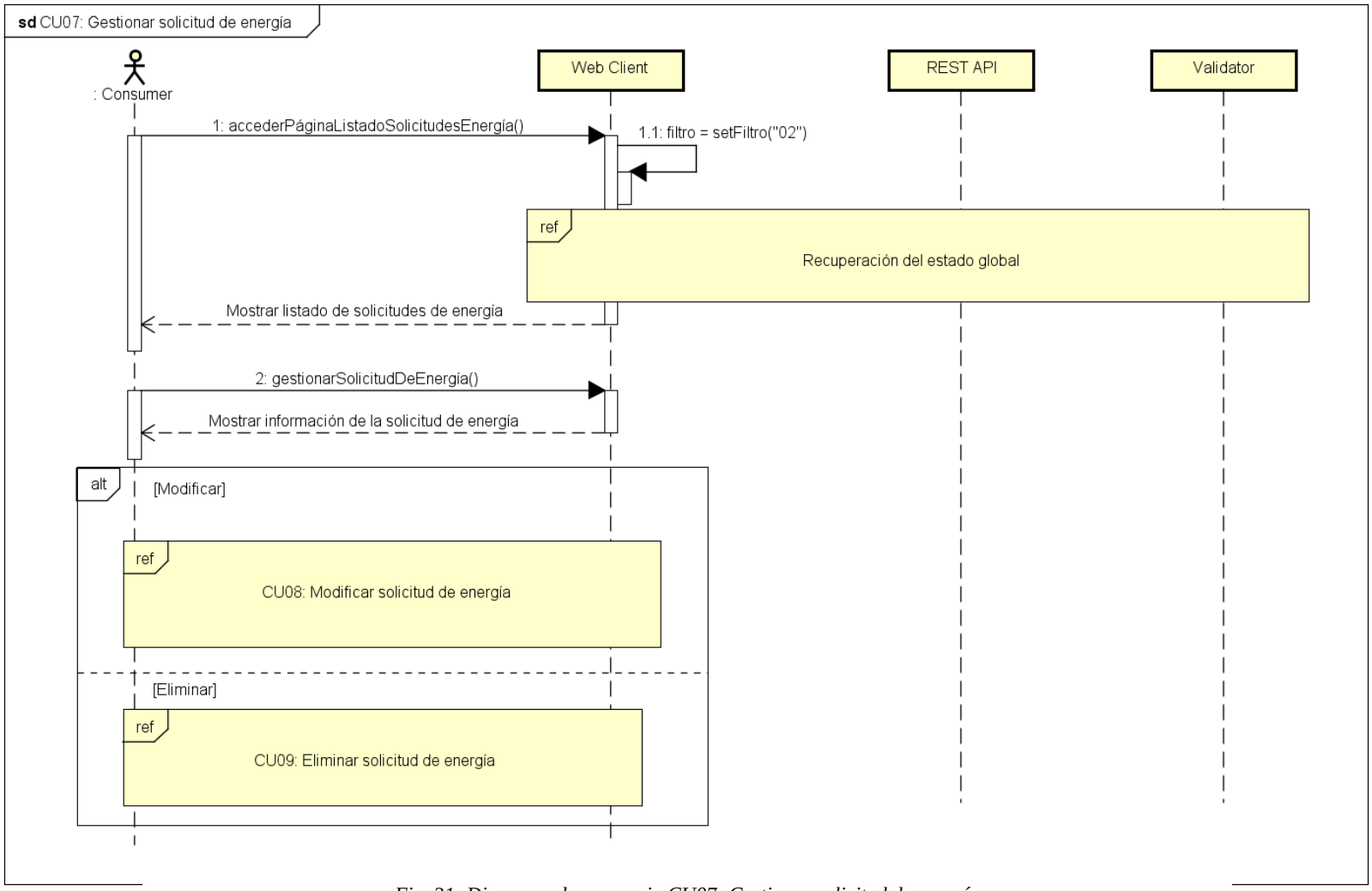


Fig. 31: Diagrama de secuencia CU07: Gestionar solicitud de energía

Los diagramas referenciados son: Fig. 32, Fig. 33 y Anexo: Fig. 38.

19.8 Diagrama de secuencia CU08: Modificar solicitud de energía

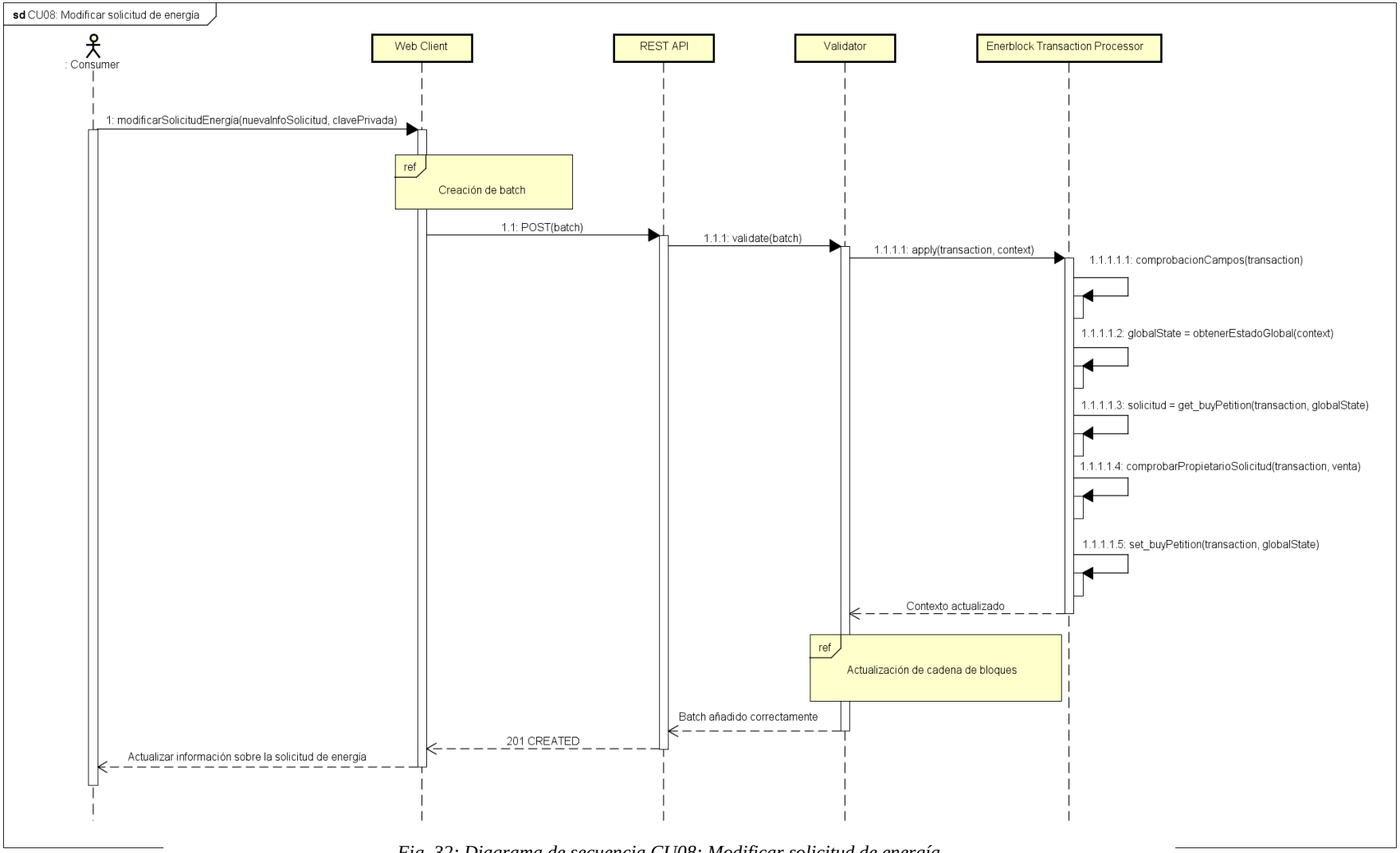


Fig. 32: Diagrama de secuencia CU08: Modificar solicitud de energía

Los diagramas referenciados se encuentran en el Anexo: Fig. 36 y Fig. 37.

19.9 Diagrama de secuencia CU09: Eliminar solicitud de energía

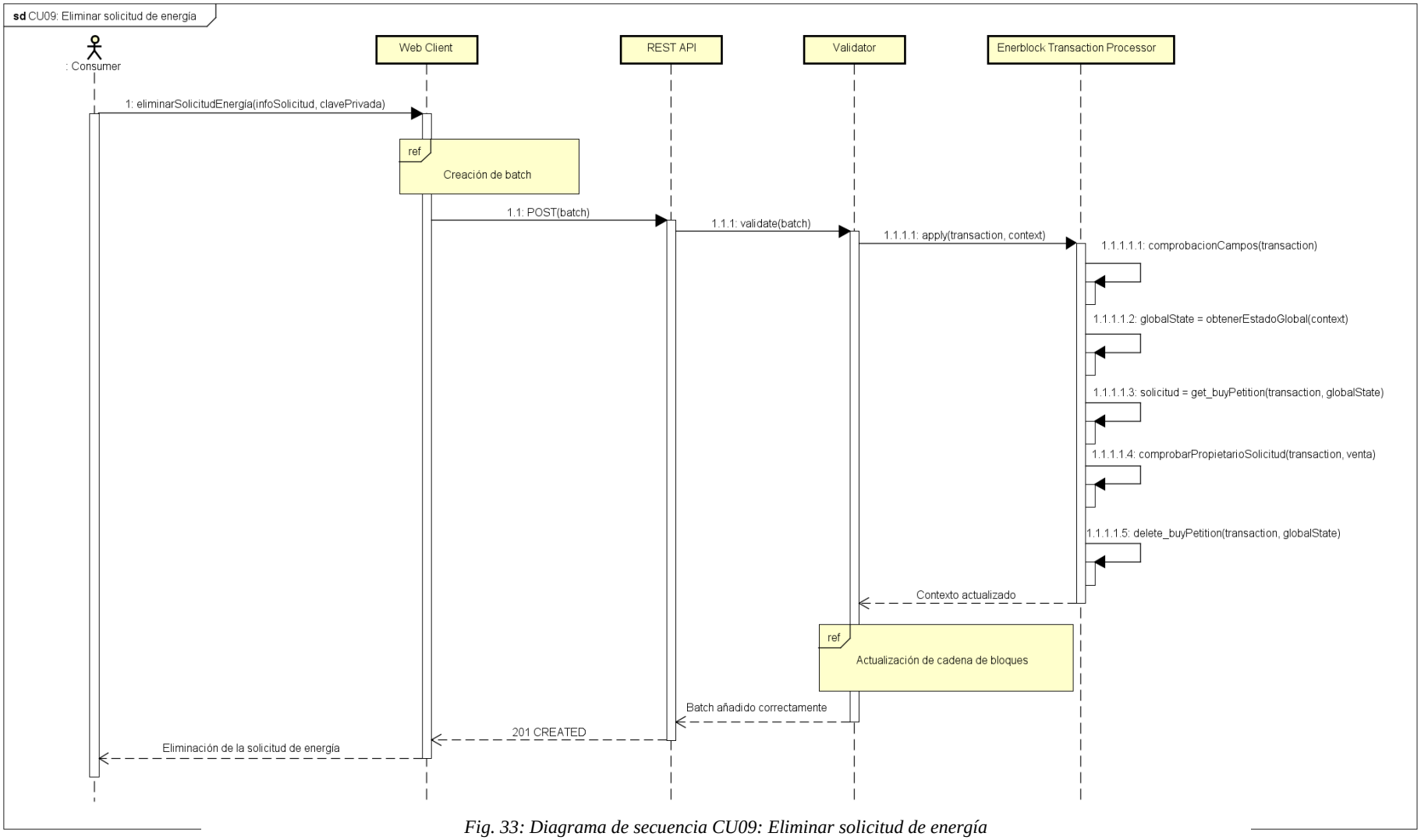


Fig. 33: Diagrama de secuencia CU09: Eliminar solicitud de energía

Los diagramas referenciados se encuentran en el Anexo: Fig. 36 y Fig. 37.

19.10 Diagrama de secuencia CU10: Satisfacer solicitud de energía

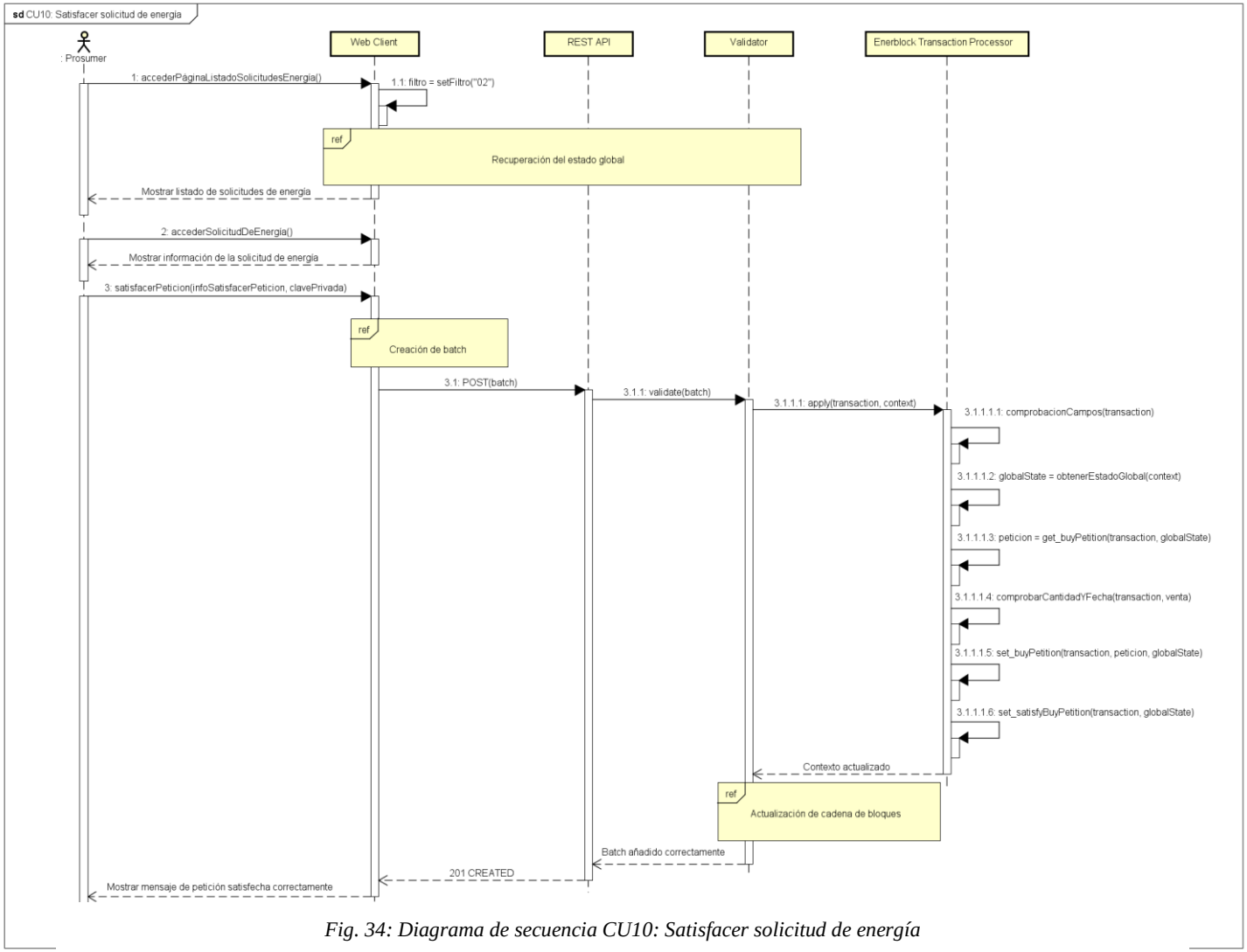


Fig. 34: Diagrama de secuencia CU10: Satisfacer solicitud de energía

Los diagramas referenciados se encuentran en el Anexo: Fig. 36, Fig. 37 y Fig. 38.

19.11 Diagrama de secuencia CU11: Acceder a resumen informativo de intercambios

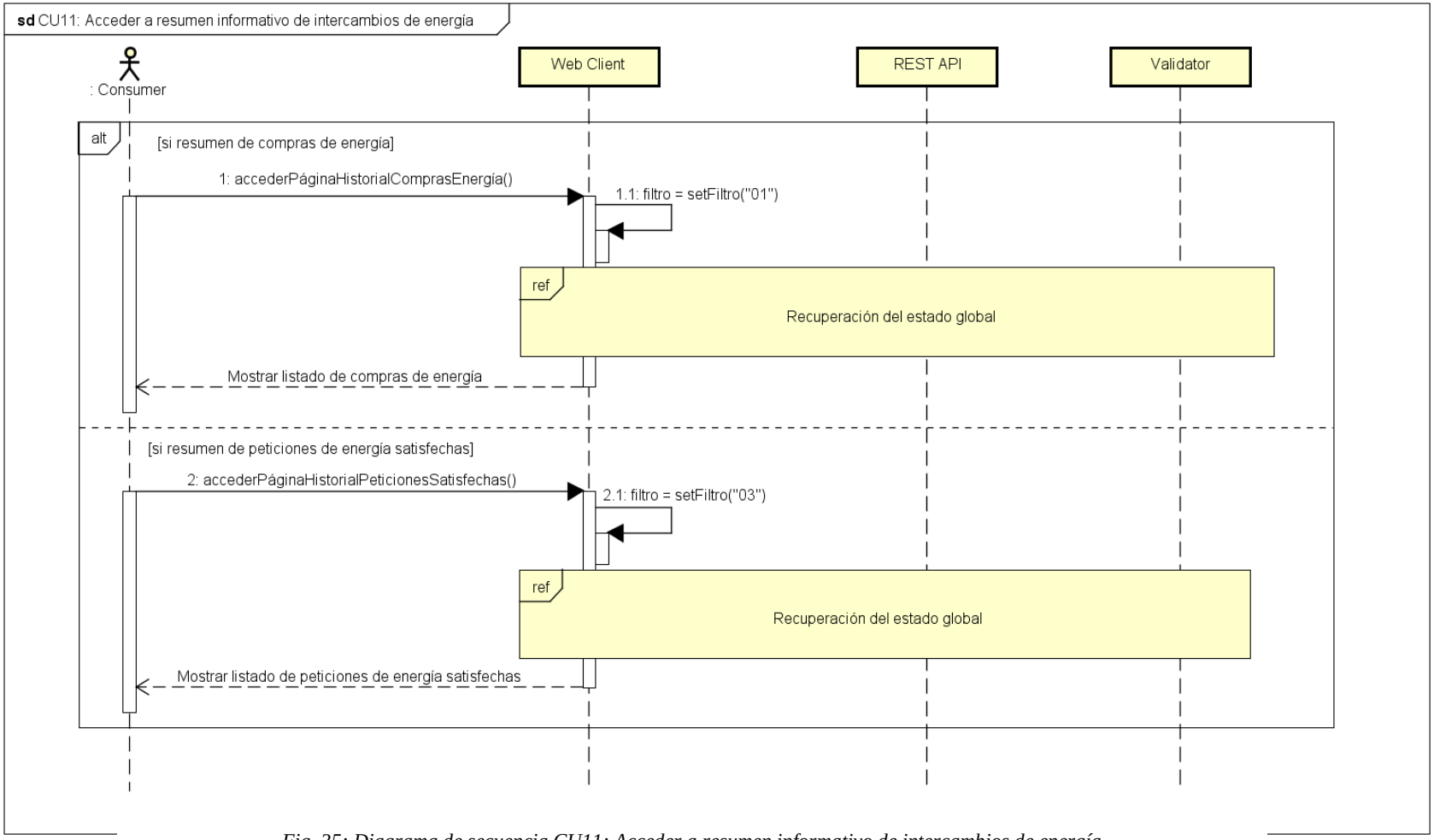


Fig. 35: Diagrama de secuencia CU11: Acceder a resumen informativo de intercambios de energía

El diagrama referenciado se encuentra en el Anexo: Fig. 38.

20. Diagramas de secuencia referenciados

20.1 Creación de batch

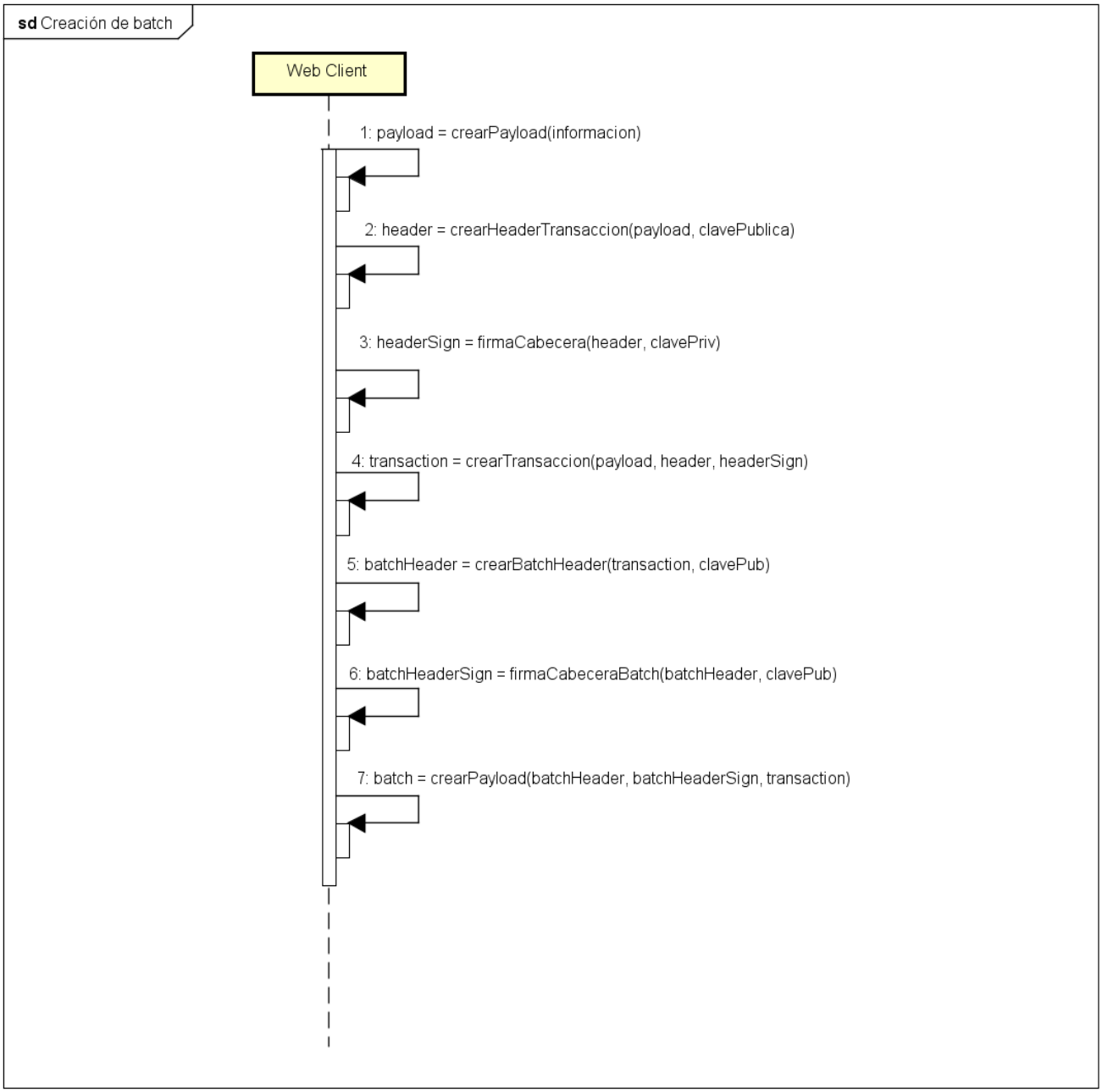


Fig. 36: Diagrama de secuencia de creación de batch

20.2 Actualización de cadena de bloques

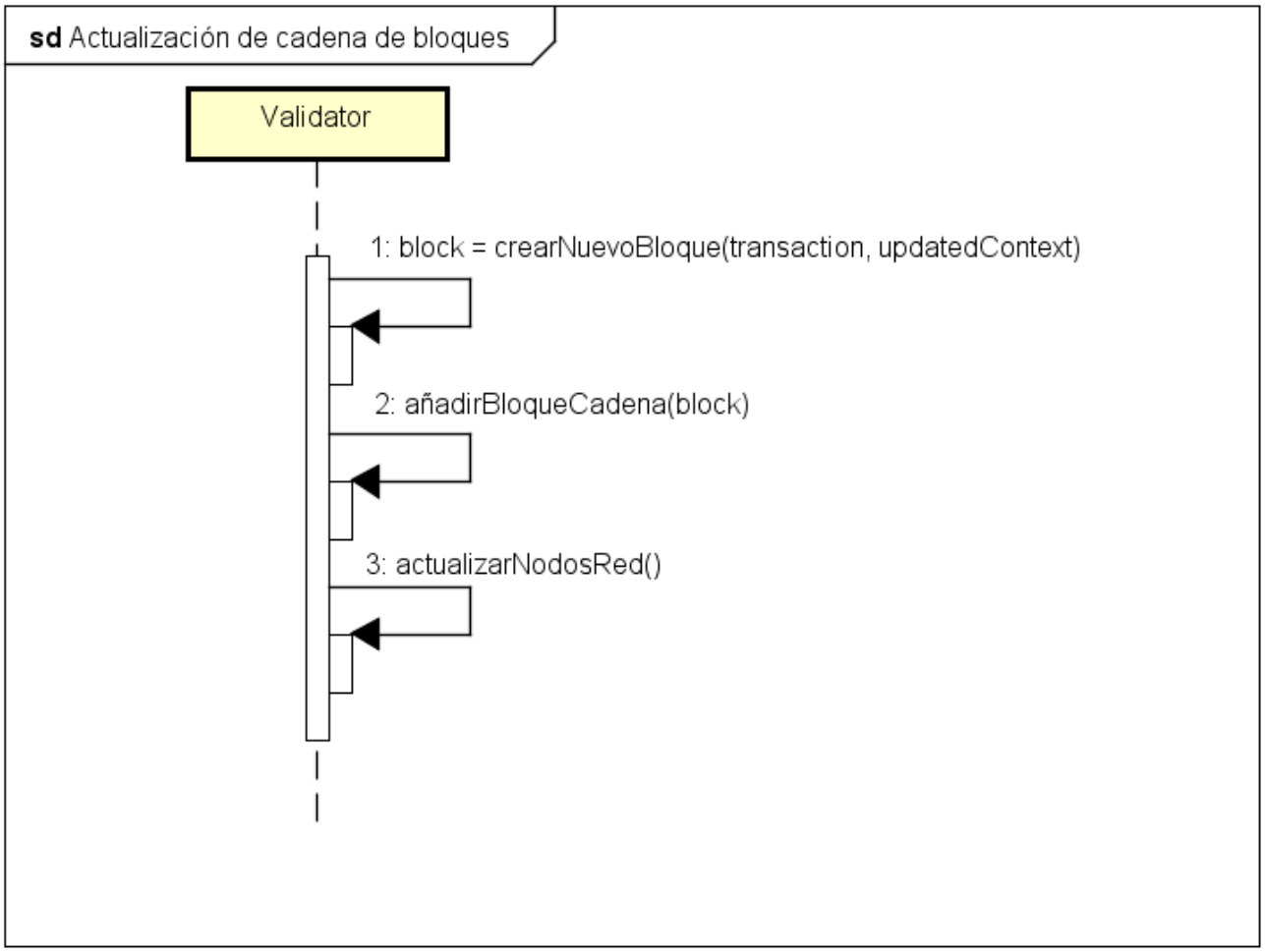


Fig. 37: Diagrama de secuencia de actualización de cadena de bloques

20.3 Recuperación del estado global

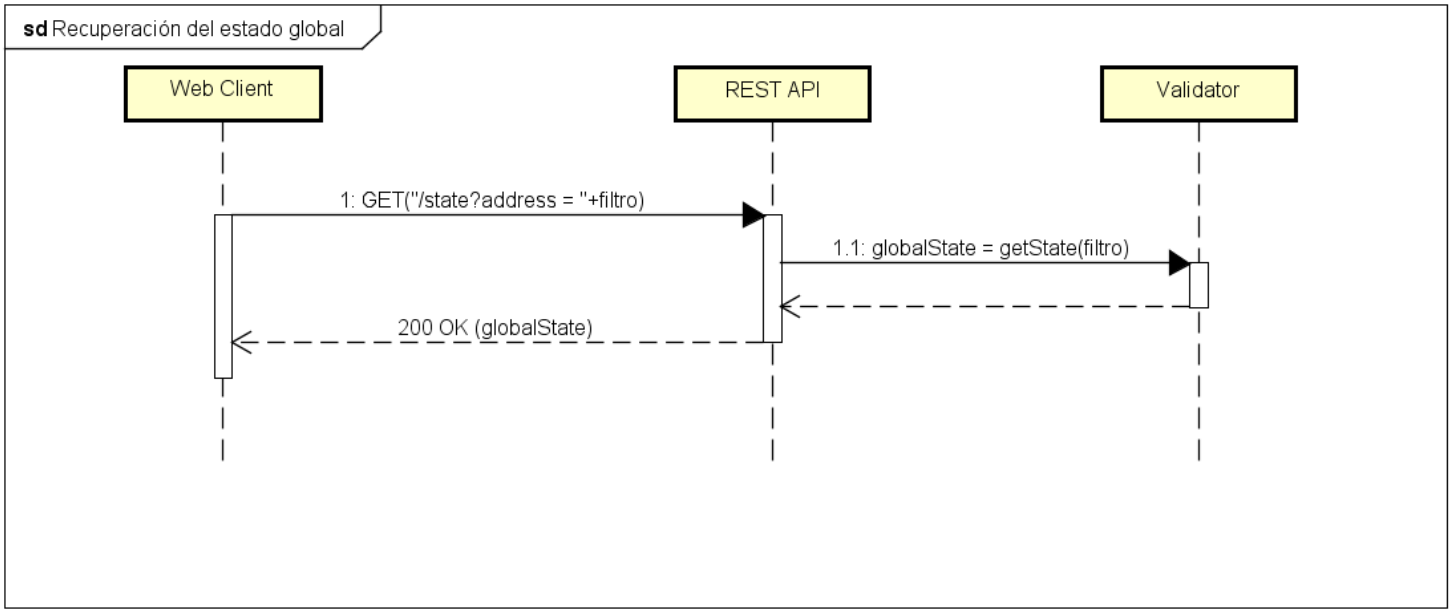


Fig. 38: Diagrama de secuencia de recuperación del estado global