Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Investigaciones en Comunicaciones y Redes (CICORE)

DESARROLLO DE SERVIDOR DE AUTENTICACIÓN Y MOTOR DE PAGOS EN CRIPTOMONEDAS UTILIZANDO IDENTIFICACIÓN POR RADIOFRECUENCIA (RFID)

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por la bachiller
Raquel Jeleitza Escalante Salazar,
C.I.N° 19.371.273
para optar al título de
Licenciado en Computación

Tutor:
Prof. Antonio Russoniello

Caracas, octubre 2019

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Investigaciones en Comunicaciones y Redes (CICORE)

ACTA DEL VEREDICTO

Quienes suscriben, miembros del Jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por la Bachiller Raquel Jeleitza Escalante Salazar, CI: 19.371.273, titulado: "**Desarrollo de un Servidor de Autenticación y Motor de Pagos en Criptomonedas utilizando Identificación por Radiofrecuencia (RFID)**", a los fines de cumplir con el requisito legal para optar por el título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 21 de octubre de 2019 a las 10:30 AM en Laboratorio Internet 2, Galpón 10, para que su autor lo defendiera en forma pública, mediante una presentación oral de su contenido, luego de lo cual respondió las preguntas formuladas, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente Acta, en Caracas a los 21 días del mes de octubre del año dos mil diecinueve (2019) dejándose también constancia de que actuó como coordinador del Jurado el Profesor Tutor Antonio Russoniello.

Prof. Antonio Russoniello (Tutor)

Prof. Miguel Astor (Jurado)

Prof. Andrés Sanoja (Jurado)

RESUMEN

Título:

Desarrollo de Servidor de Autenticación y Motor de Pagos en Criptomonedas utilizando la tecnología de Identificación por Radiofrecuencia (RFID)

Autor:

Br. Raquel Jeleitza Escalante Salazar

Tutor:

Prof. Antonio Russoniello

Resumen:

Con la expansión de las tecnologías basadas en internet, son diversos los ámbitos de la sociedad que han implementado el uso de las mismas para su funcionamiento. En particular, el mundo de las finanzas ha dado un vuelco con la aparición de las criptomonedas, un esquema distribuido y descentralizado que se ha levantado como el medio financiero de la Internet en los últimos años. Por ello, las criptomonedas se presentan como un elemento importante en la sociedad actual y altamente compatible con tecnologías asociadas a las telecomunicaciones, tal como puede ser la tecnología de Identificación por Radiofrecuencia. En este Trabajo Especial de Grado, se plantea el desarrollo de un Servidor de Autenticación y un Motor de Pagos que sea parte de un Mecanismo de Pago en Criptomonedas que tenga como base o entrada de datos un sistema de Identificación por Radiofrecuencia, a través de la elaboración de un proyecto web basado en Python Django que proporcione un medio para la realización de transacciones y su emisión a la *blockchain*, o cadena de bloques de una criptomoneda. Un mecanismo de este estilo, pone a disposición de la población general la capacidad de realizar transacciones rápidas y de baja denominación en criptomonedas, facilitando su adopción en un entorno en el que sea necesaria la implementación de nuevos esquemas económicos. Durante el desarrollo del presente trabajo, a través de la creación de una Interfaz de Programación de Aplicaciones (API) para la realización de transacciones con fracciones de criptomonedas, y las pruebas realizadas en cuanto a tiempos de transacción y éxito en la transmisión de transacciones a una cadena de bloques de la criptomoneda Dash, fue posible determinar algunos parámetros importantes para la implementación de sistemas de pago para criptomonedas, relacionados tanto a la construcción de transacciones, sus montos asociados, tiempos de ejecución, y analizar características importantes en una solución de este tipo, como la disponibilidad, escalabilidad y costos de desarrollo, implementación y despliegue que pueden surgir dependiendo de las herramientas a utilizar en la solución.

Palabras Clave: Mecanismos de Pago, Pagos, Transacciones, Bitcoin, Dash, Criptomonedas, Identificación por Radiofrecuencia, Autenticación, Aplicaciones Web, Python, Django

AGRADECIMIENTOS

A Dios, por otorgarme la fuerza y el valor necesario en esta etapa, por su presencia a través de cada una de las personas que me han apoyado a lo largo de la carrera y la vida.

A mis padres, Yoan Salazar y Régulo Escalante, por ser mis guías, mi fuerza, por estar conmigo en cada paso, por haberme forjado como soy, y a Rafael Vásquez, por ser mi pilar, mi estrella del norte, darme la motivación y el amor necesario para seguir adelante con todo y más. Los amo.

A mis amigos de la carrera por la compañía, el esfuerzo y los buenos momentos, y a Luz Ferrer, por ser mi confidente y mi apoyo en los momentos más claves de toda la carrera, no puedo pedir una mejor amiga.

A mi tutor, Antonio Russoniello, por ser uno de los profesores más apasionados y comprensivos que he tenido el honor de atender, por su orientación y guía, y por darme la oportunidad de construir este proyecto. Muchísimas gracias.

Finalmente, a la Ilustre Universidad Central de Venezuela, por las segundas oportunidades, por haberme dado una segunda casa a la que por fin puedo llamar mi Alma Mater.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las tecnologías de comunicaciones e información se han convertido en base fundamental del funcionamiento de la sociedad desde la aparición de la Internet y su constante evolución durante las últimas décadas. Esto ha dado paso a la inclusión de Internet y tecnologías derivadas a diversos aspectos y saberes comunes a la sociedad que no se limitan solo a las ciencias puras, tales como agricultura, infraestructura y finanzas.

En particular, las finanzas han dado un giro significativo desde hace aproximadamente tres décadas con la aparición de diversas propuestas que han mantenido dos objetivos particulares: Crear esquemas con raíces en la descentralización y la colaboración entre pares. Estos esquemas, que poco a poco han ido modificando las reglas establecidas por la banca tradicional, se han visto mayormente representados en la última década con la aparición de la tecnología de Cadena de Bloques (*Blockchain*), presentada en el año 2008 [1], y que ha sido ampliamente adoptada por lo que se conoce en la actualidad como *criptomonedas.*

Las criptomonedas trabajan bajo un esquema descentralizado, asegurado por su funcionamiento bajo algoritmos criptográficos y dependen ampliamente de la colaboración entre pares. Se han levantado como el esquema financiero de la Internet, y por ello, resulta más sencillo incorporar a sus mecánicas elementos propios de las tecnologías de las telecomunicaciones. Entre estas tecnologías destaca particularmente la Tecnología de Identificación por Radiofrecuencia, o *Radio Frecuency Identification* por sus siglas en inglés, un sistema inalámbrico de almacenamiento y lectura de datos que ofrece diversas posibilidades para la creación de sistemas que propicien la adopción más rápida de esquemas financieros basados en criptomonedas. Este trabajo se enfoca principalmente en el estudio de las tecnologías antes mencionadas

El presente Trabajo Especial de Grado surge para documentar el desarrollo de un Servidor de Autenticación y un Motor de Pagos que funciona de soporte para un mecanismo, basado en la tecnología de identificación por radiofrecuencia, que permita registrar y procesar pagos de una determinada criptomoneda. El documento se encuentra organizado como se describe a continuación:

- Capítulo 1: Planteamiento de la problemática a solventar, basado en diversos problemas económicos actuales en el territorio venezolano, y cuya solución se encuentra orientada a la elaboración de un mecanismo de pagos en criptomonedas.
- Capítulo 2: Marco conceptual, en el que se definen conceptos asociados a las Criptomonedas, las diferentes tecnologías de desarrollo asociadas a la solución y la tecnología de Identificación por Radiofrecuencia.
- Capítulo 3: Se establece un Marco Aplicativo, en el que se presentan en detalle las tecnologías y la metodología de trabajo utilizada.
- Capítulo 4: Diseño e implementación de la solución, en el que se expone el diseño del sistema y su implementación de acuerdo a la metodología descrita en el capítulo 3 de este trabajo de investigación

- Capítulo 5: Contribución, Limitaciones y Trabajos futuros.
- Conclusiones, referencias bibliográficas y diferentes anexos relacionados a la solución.

1. PLANTEAMIENTO DEL PROBLEMA

Las criptomonedas han extendido su uso con gran rapidez apoyándose en la tecnología disponible a través de Internet y utilizando como instrumento de pago a los teléfonos inteligentes, que son altamente dependientes de Internet. Sin embargo, la limitante de muchas criptomonedas al momento de realizar una transacción sigue siendo la rapidez para realizar dichas transacciones y esto genera una sobrecarga bastante particular en el caso de pagos de baja denominación.

Por otro lado, los venezolanos se han instruido ampliamente en el funcionamiento de la economía basada en criptomonedas, dado el problema a nivel nacional que ha generado la hiperinflación, pero se presentan diversos obstáculos que pueden interferir en la adopción completa de este esquema económico, siendo el primero a considerar, la extensión de la infraestructura tecnológica necesaria para la conexión a internet extendida a nivel nacional y la necesidad de poseer teléfonos inteligentes como contenedores de las billeteras electrónicas que son indispensables para que los usuarios interactúen con las criptomonedas. En segundo lugar, pero no menos importante, se debe tomar en cuenta el factor seguridad. Los problemas de seguridad existentes en el país hacen que la mayoría de sus usuarios considere no salir a la calle con un teléfono inteligente, puesto que consideran que son objetivos claros de maleantes que muchas veces toman acciones desmedidas con tal de obtener dichos dispositivos.

Se pretende elaborar una solución que sirva como base de un mecanismo de pagos y permita al usuario común realizar transacciones de baja denominación, utilizando una criptomoneda que tenga cuotas bajas de transacción, que la hagan apta para trabajar en una plataforma que agilice estos procesos como sería un punto de venta basado en Identificación por Radiofrecuencia (*Radio Frequency Identification*, RFID).

1.1. Justificación

En primera instancia, cabe plantearse que para lograr una extensión del uso de criptomonedas es importante crear elementos que puedan ser fácilmente asociados por cualquier usuario miembro de la población. En la actualidad las criptomonedas a nivel financiero son un instrumento válido y pueden llegar a ser fáciles de usar y generar ciertas recompensas pequeñas por su uso, tal como ha implementado la franquicia *Church's Chicken* en Venezuela[1].

Al hablar de Venezuela, país de Latinoamérica en el que se ha extendido el uso de criptomonedas de forma más rápida debido a la movida hiperinflacionaria que se vive en el territorio, surge una necesidad de desarrollar mecanismos de pago que incorporen los esquemas ya definidos para criptomonedas como es el uso de billeteras electrónicas, junto con formas de funcionamiento más familiares como es el uso de una tarjeta en un punto de venta. Esto puede derivar en soluciones

[1] https://dashnews.org/churchs-chicken-venezuela-offers-special-promotion-for-paying-with-dash/

económicas más tangibles basadas en la criptoeconomía, pero que no requieran una curva de aprendizaje relativamente pronunciada para usuarios que no estén tan orientados a la tecnología, es decir, serán de fácil adopción para un grupo importante de usuarios.

La solución a plantear contempla el desarrollo de dos elementos que sirvan de soporte a un sistema de Identificación por Radiofrecuencia, para almacenar una billetera de alguna criptomoneda que permita rapidez en sus transacciones. Dicho sistema podrá interactuar con la tarjeta para obtener información sobre la billetera electrónica, realizar transacciones y reportar dichas transacciones a la *blockchain* de la criptomoneda seleccionada. La información almacenada en la tarjeta se obtendrá bajo un esquema de autenticación adicional al esquema criptográfico que se emplee al usar la tecnología RFID.

Esto implica que un usuario final podrá realizar transacciones en criptomonedas sin requerir que su billetera se encuentre almacenada en un teléfono inteligente y por ende, realizar transacciones con criptomonedas puede ser más seguro para el usuario en varios aspectos, principalmente físicos al emplear un medio sencillo como es una tarjeta, y tecnológicos, dados por la implementación del sistema y los medios de autenticación empleados. Estos aspectos convierten a la tarjeta en un elemento muy similar a las tarjetas tradicionales de débito y crédito que se utilizan actualmente en Venezuela, con la ventaja de poder operar sin contacto directo con el lector y con un nivel de rapidez similar al de los puntos de venta tradicionales.

1.2. Objetivo General

Desarrollar una plataforma conformada por un servidor de autenticación y un motor de transacciones destinada a la realización de transacciones en criptomonedas que provea una interfaz eficiente para el ingreso de datos desde un punto de venta basado en RFID.

1.3. Objetivos Específicos

1. Definir una arquitectura base de hardware y software para el desarrollo de la plataforma.
2. Implementar un modelo relacional de datos para el soporte del servidor de autenticación de la plataforma.
3. Desarrollar la interfaz de interacción entre el servidor de autenticación y el motor de pagos de la solución, implementando un API (*Application Programming Interface*, o Interfaz de Programación de Aplicaciones) de uso interno en ambas aplicaciones.
4. Implementar esquemas de creación de transacciones e interacción con la *blockchain* de una determinada criptomoneda.
5. Desarrollar la interacción entre la aplicación base del servidor de autenticación y el motor de pagos, y el elemento de hardware que fungirá como medio de entrada de datos.
6. Realizar las pruebas pertinentes para asegurar la integración y la calidad de la plataforma.

1.4. Alcance

Con este proyecto se busca verificar la viabilidad de los primeros pasos hacia un sistema de punto de venta para criptomonedas con bajos costos de transacción y que pueda realizarse con tecnología RFID en conjunto a placas de desarrollo tales como Arduino o Raspberry Pi. En este sentido, el desarrollo de la solución está orientado a:

- Desarrollo de dos aplicaciones web interconectadas en un proyecto base general: el servidor de autenticación y el motor de pagos o transacciones.
- Desarrollo de una interfaz web para el servidor de autenticación, de forma que un usuario administrador pueda realizar el registro adecuado de otros usuarios administradores de la plataforma, de puntos de venta y de las tarjetas a ser asociadas al sistema.
- Desarrollo de las respectivas API de Autenticación y Operación para el Servidor de Autenticación y el Motor de Pagos respectivamente.
- Implementación y configuración adecuada de los proveedores de servicio en criptomonedas, tanto en la nube como a nivel local para el procesamiento de las transacciones a realizar.
- Verificación de información básica a nivel de usuario cliente respecto a su cuenta y transacciones realizadas.

En el caso de este Trabajo Especial de Grado, se utilizó la criptomoneda *Dash* como base de todas las implementaciones realizadas.

2. MARCO CONCEPTUAL

En este capítulo se exponen y definen diversos conceptos y la investigación realizada sobre las tecnologías relacionadas que permitirán abordar el problema y el diseño de su solución de una forma estructurada.

2.1. Criptomonedas

En las monedas fiduciarias, organizaciones como los bancos centrales controlan la oferta y la demanda de dinero y añaden parámetros que permiten detectar y luchar contra la falsificación a la moneda física. Estas características de seguridad aumentan la dificultad de realizar maniobras maliciosas por parte de un posible atacante, pero no hacen que el dinero, en formato físico, sea imposible de falsificar.

Desde el 2009, han entrado nuevos actores al mundo financiero. Las criptomonedas, que son una visión *Peer-To-Pee*r (P2P) de lo que se considera como dinero electrónico, forman parte de un esquema que hace un gran uso de la criptografía para establecer sus normas de creación y seguridad, y cuyo objetivo es garantizar el intercambio entre partes sin que exista un organismo centralizado. La criptografía proporciona un mecanismo para codificar de forma segura las reglas de un sistema de criptomoneda dentro del propio sistema. Esto se hace con el fin de evitar manipulaciones y dobles transacciones, así como para codificar en un protocolo matemático las reglas para la creación de nuevas unidades de moneda.

El primero de los movimientos hacia una economía digital basada en criptografía fue dado por el criptógrafo David Chaum desde el año 1982. En ese año, Chaum propuso un sistema de pago no rastreable basado en firmas ciegas[2], lo que permitía a sus usuarios obtener tokens anónimos de un banco, cada token representando una cantidad fija de dinero. Sin embargo, esta primera versión podía recibir ataques de doble gasto, por lo que solo podía ser usado como un sistema de pago online.

Luego, en el año 1990, Chaum propone *Ecash*, una versión mejorada de su sistema de firmas ciegas, propuesto para transacciones *offline*. Para ello, el usuario debía enviar al banco un número aleatorio e información oculta relacionada con el usuario, que no puede ser accedida por algún *merchant* que esté recibiendo dicha información para un pago. La desventaja de este método era que, si dos *merchants* recibían el mismo token como pago, los tokens podían combinarse para recuperar esa información oculta sobre el usuario. De cualquier manera, los métodos propuestos por Chaum involucraban a un tercero de confianza, a los bancos.

En 1997, Adam Back introduce *HashCash*, un token que se añadía a las cabeceras de algunos mensajes de correo electrónico, y tenía el propósito de forzar a los emisores de spam a gastar una suma considerable de recursos computacionales para poder hacer ataques de este tipo. Cada HashCash

[2] Primitiva criptográfica introducida por David Chaum en 1982 en la que el mensaje se combina con un factor "enceguecedor" (*blinding factor*) y luego es firmado por el ente que envía el mensaje, sin vincular de alguna manera el mensaje a algún usuario [2]

es específico de cada receptor, de manera que el receptor puede verificar su validez sin una tercera parte de confianza, convirtiéndolo en uno de los primeros esquemas descentralizados de intercambio. Si bien no es un sistema de pagos digitales, provee un antecedente importante, tanto en generación de tokens de forma criptográfica, como de ser un organismo descentralizado que tuvo el potencial de convertirse en una divisa en el mundo de la web.

En cuanto a los sistemas distribuidos, tanto Nick Szabo como Wei Dai propusieron de forma independiente en el año 1998 esquemas digitales de dinero, por un lado, *bit gold*, propuesto por Szabo, y *b-money* propuesto por Dai. La idea principal tras estas propuestas era que los balances de cada uno de estos esquemas se alojaran en una base de datos distribuida, y la creación de sus divisas se realizaba a través de un problema computacional difícil de resolver, pero fácil de verificar, es decir, una prueba de trabajo. En ambos esquemas, la dificultad del problema era ajustable, aunque ninguno de ellos especificaba algún esquema de consenso concreto para los nodos de la red.

En 1999 Tomas Sander y Amnon Ta-Shma, propusieron un sistema anónimo de dinero electrónico que no requería un servidor central que creara las firmas digitales ciegas. En el sistema de Sander y Ta-Shma, una moneda se representa por el hash de su número de serie. Luego, una lista de monedas válidas se mantiene en el banco y para realizar un retiro, el usuario crea un número aleatorio para la moneda y utiliza una función trampa para computar un número adicional que corresponde a la moneda. Esta función trampa contiene información que sirve para levantar el anonimato del usuario en caso de que se presente un doble gasto. Para su funcionamiento, en vez de utilizar el sistema de firmas ciegas, utiliza una prueba de *Zero-Knowledge*[3], presentándose como antecedente directo de algunas criptomonedas actuales como Zerocoin.

Otro de los antecedentes es el *RPOW* de Hal Finney, en 2004. Siendo las siglas de *Reusable Proof-of-Work* (Prueba de trabajo reusable), RPOW funciona creando un token de tipo hashcash que no se encuentra ligado a ninguna aplicación y por tanto puede ser gastado libremente. Los clientes pueden crear tokens POW realizando una prueba de trabajo, por lo que el valor de los tokens POW se encuentra ligado a los recursos computacionales que se gasten en su creación. Lo distintivo de RPOW, es que cuando un usuario decide usar el token POW, éste se envía a un servidor RPOW y puede ser cambiado por un nuevo token POW. Para evitar el doble gasto, este token POW debe ser reportado al servidor RPOW, por lo que es un sistema enteramente online. Sin embargo, el servidor RPOW poseía diversas fallas criptográficas y fue dado de baja. [2]

2008 fue un año importante para la economía digital, pues el 15 de septiembre de ese año, luego de que *Lehman Brothers*, el cuarto banco de inversiones más grande de Estados Unidos se declarara en bancarrota, inicia la Crisis Financiera Global del 2008. Justo después, en el mes de octubre fue publicado por un autor con el pseudónimo de Satoshi Nakamoto, un *whitepaper* que proponía una solución de dinero electrónico descentralizado. El software de Bitcoin, lanzado a inicios del 2009, fue construido sobre un libro de transacciones público, distribuido y validado por una red de nodos llamado *Blockchain* o cadena de bloques, una tecnología de código abierto que fue una solución

[3] Tipo de prueba en la que se reduce básicamente a cero la cantidad de conocimiento que debe transmitirse de un ente a un verificador para probar la validez de un sistema o un mensaje (https://blockonomi.com/zero-knowledge-proofs/)

innovadora a uno de los problemas más prominentes que presentaba la economía digital: el doble gasto.[1]

2.1.1. Bitcoin

Bitcoin es una red descentralizada, que cuenta con una moneda digital que usa un sistema *P2P* para verificar y procesar transacciones. En lugar de contar con la presencia de terceros confiables para procesar las transacciones, como bancos y procesadores de pago, la tecnología Bitcoin utiliza pruebas de trabajo criptográficas para procesar las transacciones y verificar la legitimidad de los bitcoins, así como de asegurar que la cantidad de trabajo es distribuida a través de la red [3].

El *whitepaper* de Bitcoin fue publicado por Satoshi Nakamoto, un autor cuya identidad real aún es desconocida, el 31 de octubre de 2008, y en él, se explica como una estructura denominada *Blockchain* es capaz de soportar una moneda digital puramente descentralizada, sin necesidad de una autoridad central. El *whitepaper*, además, menciona los problemas que surgen al confiar en instituciones financieras para procesar transacciones y mediar con la reversibilidad de las transacciones fraudulentas, aumentando el riesgo y el costo de estas. El objetivo principal de Bitcoin es asegurar que las transacciones, por ser computacionalmente impráctico revertirlas, protejan a los participantes del fraude, es decir, el dueño de sus claves controla su dinero, por lo que se puede deducir que toda la solución de Nakamoto recae en pruebas criptográficas.

Bitcoin utiliza principalmente criptografía de clave pública para solucionar el problema del doble gasto, pues cada transacción tiene una firma digital y contiene un hash que permite identificar cada transacción.

El primer bitcoin fue minado o creado a inicios del 2009, con la creación del llamado Bloque Génesis, afianzando la estructura básica de la red Bitcoin: La *Blockchain* [1].

2.1.1.1. La Blockchain de Bitcoin

La *blockchain*, o cadena de bloques, es un libro público de cuentas que tiene registro de todas las transacciones realizadas en la red Bitcoin. Se le da el nombre *blockchain* porque todas las transacciones son agrupadas en bloques que son validados por la red de nodos de Bitcoin.

En el *whitepaper* de Bitcoin, se define una moneda electrónica como una cadena de firmas digitales. Cada propietario de una firma transfiere su moneda al siguiente al firmar la transacción anterior y la clave pública del próximo propietario y añadiendo ésta al final de la cadena de propiedad (*chain of ownership*) o de la moneda. La forma que este método aplica para evitar el doble gasto es acordando que la primera transacción registrada en la cadena es la que cuenta. Desde luego, para lograr esto es necesario que todos los nodos estén al tanto de todas las transacciones y el momento en el que se realizan para llegar al consenso de cuál fue la primera transacción. Para ello, se implementa un servidor de marcas de tiempo, que realiza un hash de un bloque de ítems y publica el hash realizado.

Además, es necesario implementar un sistema de prueba de trabajo (*Proof-of-Work*), que implica el escaneo de un valor tal que cuando se le aplica una función hash, el hash inicia con un número de bits

en cero. El trabajo promedio requerido para el cálculo es exponencial según el número de bits en cero, pero puede ser verificado ejecutando un hash simple. Como esta prueba de trabajo depende de cierto tiempo de trabajo y consumo de recursos de CPU, el bloque que satisfaga la prueba de trabajo no puede ser cambiado sin rehacer todo el trabajo.

A través de esas pruebas de trabajo, la cadena puede ir creciendo en tamaño y al final, el mayor peso en el consenso lo tiene entonces la cadena más larga, que posee la mayor cantidad de esfuerzo invertido. En caso de existir nodos maliciosos que pretendan realizar un ataque, si la mayoría de los CPU en los nodos son controlados por nodos honestos, la cadena honesta crecerá más rápidamente y, por consiguiente, otras cadenas creadas para alterar la cadena principal crecerán de forma más lenta y no podrán mantenerse [1].

2.1.1.2. Bloques

En general, la *blockchain* es una cadena de bloques enlazados desde el bloque génesis generado en 2009 hasta el último bloque de transacciones. Cada nodo conectado a la red mantiene una copia completa de la cadena de bloques entera. Es un sistema redundante, pero ha logrado afianzar la resistencia del sistema Bitcoin. En la figura 2.1.1.2.1 se esquematiza la estructura de un bloque en la *blockchain* de Bitcoin.

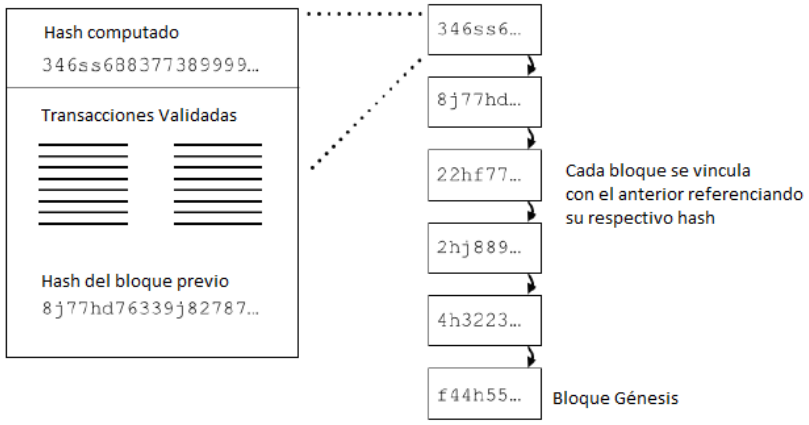

Figura 2.1.1.2.1: Estructura de un bloque
Fuente: [4]

Para añadir nuevos bloques a la cadena, proceso que recibe el nombre de **minado**, los nodos escuchan las transacciones que se realizan en la red. Los nodos pueden compartir y retransmitir transacciones entre sí. Cuando un nodo recibe una nueva transacción, se añade al nuevo bloque. El bloque se mantiene localmente hasta que un problema computacionalmente difícil se resuelve usando el nuevo bloque como base para la solución, es decir, se realiza la prueba de trabajo creando un hash del bloque y el hash del bloque anterior. El nivel de dificultad se calcula usando la tasa de aceptación de los bloques minados. Si la tasa es menor a 10 minutos, se incrementa la dificultad para el próximo bloque. Si toma más de 10 minutos, la dificultad baja. Este nivel de dificultad se actualiza cada 2016 bloques.

En general, el proceso de minado sigue el siguiente esquema:

- Las transacciones nuevas son transmitidas a toda la red. Los nodos retransmiten nuevas transacciones a los otros nodos. Estas nuevas transacciones se marcan inicialmente como no confirmadas (*unconfirmed*).
- Cada nodo recolecta y valida las transacciones, y las inserta en un nuevo bloque. Los nodos escuchan continuamente en espera de nuevas transacciones y actualizan el bloque según sea necesario.
- Cada nodo busca la solución a un problema difícil que usualmente involucra el cómputo de un hash del bloque. La solución también incluye encontrar un valor hash que sea menor al valor objetivo publicado.
- Si se encuentra una solución, se transmite a toda la red. La solución consiste en el bloque de transacciones y su valor hash, para su posterior verificación por parte de los otros nodos en la red.
- Si todas las transacciones en el bloque y su valor hash son válidos, entonces el bloque se añade a la cadena, que se convierte en la cadena más larga. El proceso se repite a partir de esta cadena [1].

2.1.1.3. El problema a solucionar: Minado

La idea principal del minado es obtener un valor hash que sea menor al nivel de dificultad publicado en la red. Para poder resolver este problema, el protocolo de Bitcoin permite que los mineros agreguen un número simple o *nonce* al final de la transacción, para poder computar distintos valores hash hasta obtener uno que sea menor al valor de dificultad. Dicho *nonce* es ignorado por la red y se incrementa por cada intento, lo que permite obtener un amplio rango de valores. Este proceso se ilustra en la figura 2.1.1.3.1.

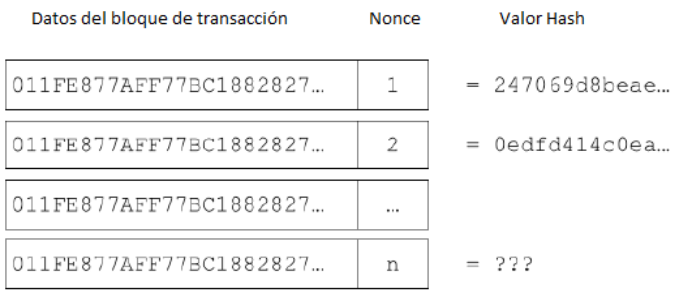

Figura 2.1.1.3.1: Forma de realizar el minado
Fuente: [4]

Esto también agrega un alto nivel de competitividad en el minado, pues para poder solventar el problema en la actualidad, es necesario generar hasta billones de operaciones por segundo.

La cadena de hashes es crítica para mantener la integridad de la cadena. Cada nuevo bloque contiene el hash del bloque anterior por lo que, si una de las transacciones es modificada, el hash cambiará al

igual que el resto de la cadena, por lo que inmediatamente se volverá inválido. Por lo tanto, el sistema logra que sea imposible modificar la cadena conforme ésta crece.

La recompensa por añadir nuevos bloques a la cadena, que se realizan aproximadamente cada 10 minutos dependiendo de la cantidad de mineros compitiendo se traduce en bitcoins a la dirección que resuelva el problema para añadir el último bloque. Al inicio, en el 2009, se liberaron 50 bitcoins por el bloque Génesis como recompensa. Después de cada 210.000 bloques minados, con un tiempo estimado de 4 años, la proporción de recompensas por bloque baja a la mitad, haciendo de Bitcoin un esquema de emisión de moneda deflacionario.

2.1.1.4. Forks

Dada la independencia de los nodos en una red Bitcoin, existe la posibilidad de que existan nodos maliciosos que intenten aceptar una transacción corrupta o rechazar algunas transacciones legítimas. Es esta una de las razones principales por las que existe el consenso en la red, pues determina cuáles bloques son aceptados en la red. Desde el bloque Génesis, la mayoría de los nodos suelen tener un comportamiento honesto, pues es mayor la recompensa al minar bitcoins de esta forma.

Sin embargo, a lo largo de los años ha cambiado la forma en la que los nodos pueden aceptar o rechazar bloques, por lo que es posible que la cadena se divida (*fork*) y cree una *blockchain* paralela. Si un bloque corrupto es detectado por la red, el resultado es un *fork* en la cadena. No obstante, al no ser validados por la red, los bloques se vuelven huérfanos y el *fork* se invalida. Los bloques válidos se añaden a la porción válida de la cadena, y se retoma el principio de considerar la cadena de bloques válidos más larga como la *blockchain* principal. Los bloques huérfanos por lo general no se utilizan.

Soluciones a bugs o actualizaciones también pueden causar *forks* en la *blockchain*. Cuando los nuevos nodos implementan una actualización de sistema, el resultado puede ser un *hard fork*. Es de esta forma como la red Bitcoin acepta o rechaza los cambios en el protocolo y el software de la red, por lo que sí es posible tener múltiples versiones del software evolucionando en la cadena [1].

2.1.1.5. Suministro de bitcoins

Desde el principio, fue establecido que el límite de bitcoins (BTC) existentes sería de 21 millones de bitcoins, y se espera que dicho límite sea alcanzado para el año 2140. En principio, todos los bitcoins que se están generando surgen por minería, por lo que el minado cumple dos funciones: añade nuevos bloques a la cadena y genera nuevos bitcoins. Una vez se haya alcanzado el límite de 21 millones de bitcoins, el incentivo para los mineros será una cuota por transacciones, similar a una comisión. Una vez sea minada la última fracción de bitcoin, los mineros que sigan contribuyendo con poder de cómputo a la red serán recompensados con esta cuota.

El requerimiento de las cuotas de transacciones se basa en un conjunto de reglas aceptadas por la red. Una de ellas, usada para prevenir pagos que tengan como objetivo añadir spam a la red, es que si una transacción es de menos de 0,01BTC entonces requiere una cuota de 0,0001BTC. Otra regla se basa en el tamaño de la transacción, pues se puede requerir una cuota de transacción si ésta supera los 10000 bytes. Para estimar el tamaño de una transacción, se utiliza la siguiente fórmula:

Tamaño en bytes = 148 * número de entradas de la transacción + 34 * número de salidas de la transacción + 10

Estas cuotas son deducidas directamente de la transacción, reduciendo la salida total por el valor de dicha cuota. Estas reglas son muy similares entre los mineros, aunque cada minero tiene la opción de escoger cuáles reglas desea implementar. Es incluso posible que un minero decida aceptar una transacción sin una cuota, sin embargo, para aumentar la probabilidad de confirmación de la transacción, es recomendable seguir la estructura de cuotas de transacción.

2.1.1.6. Tiempo de transmisión de transacciones

Si bien las transacciones de Bitcoin son transmitidas inmediatamente a toda la red, en la práctica existe un retardo de 10 minutos para la confirmación de una transacción. Este retardo se origina en el tiempo que toma crear un bloque y añadirlo a la *blockchain*, que corresponde con aproximadamente 10 minutos. Tener la confirmación asegura que la red de mineros ha verificado que los bitcoins son válidos y que no se ha incurrido en doble gasto [4].

Habitualmente, los usuarios esperan por 6 confirmaciones, que equivalen a aproximadamente 1 hora, antes de considerar una transacción "confirmada", pero cada usuario tiene la libertad de decidir cuánto tiempo desea esperar antes de considerar esta confirmación.

Sin embargo, esto hace que utilizar Bitcoin para hacer transacciones de montos pequeños o para divisa de compraventa no sea eficiente, pues estos son esquemas que requieren confirmaciones rápidas. Para agilizar un poco estos procedimientos, hay esquemas que trabajan fuera de la *blockchain*, en una red separada de microtransacciones, para luego actualizar los valores, como es el caso de la red *Lightning Network* de Bitcoin [5].

2.1.1.7. Seguridad y criptografía

Como principio básico de las criptomonedas, Bitcoin debe usar algunos parámetros específicos de criptografía, que le permiten asegurar la tecnología usada en la red. Ya que todo el funcionamiento de Bitcoin y su *blockchain* se basa en hashes, la función hash utilizada por Bitcoin es SHA-256. Esta función fue diseñada por la NSA (*National Security Agency*, Agencia Nacional de Seguridad) de los Estados Unidos y es una función de dominio público, que surge como actualización de la serie de funciones SHA-1, que ha sido analizada extensivamente y es usada en la actualidad por Bitcoin para el firmado digital que asegura cada una de las transacciones y la *blockchain*, así como para la base de los problemas matemáticos que sustentan las pruebas de trabajo requeridas para el minado y mantenimiento de la red.

Así, se deduce que el esquema criptográfico que usa Bitcoin es de clave pública asimétrica, en el que se usa SHA-256 para generar direcciones Bitcoin, firmar transacciones y verificar pagos. Se usa un algoritmo que genera dos claves separadas, enlazadas asimétricamente, es decir, una clave pública y una clave privada. Las claves son asimétricas en el sentido de que la clave pública deriva de la clave privada, pero es computacionalmente imposible obtener una clave privada desde una clave pública. En Bitcoin, la clave pública se utiliza como las direcciones Bitcoin desde las que se reciben y envían

los pagos. La clave privada se mantiene guardada, pues el sistema funciona de tal manera que se pueden verificar transacciones sin compartir la clave privada [3].

2.1.1.8. Billeteras Electronicas (*Wallets)*

Suponiendo que un ente A y un ente B desean realizar una transacción, cada uno de ellos debe contar con al menos una dirección (clave pública) asociada que realizará un pago o a la que se realizarán pagos dependiendo de su rol en la transacción. La transacción se crea, y se transmite a toda la red Bitcoin para su confirmación y si la transacción es confirmada, se transfieren los bitcoins de una dirección a otra y estará disponible para su gasto. Es posible para un usuario tener más de una dirección, en cuyo caso, para administrar dichas direcciones se utiliza un producto de software denominado *wallet* o billetera electrónica (en adelante, se referirá como billetera).

Es posible que una o más direcciones pertenecientes a una misma billetera, sirvan como entrada a una transacción de envío de bitcoins, que son firmadas digitalmente por el ente que realiza el envío utilizando su clave privada. De igual forma, en la actualidad existen múltiples billeteras que se pueden usar para Bitcoin u otras criptomonedas que permiten gestionar las direcciones. En el caso de Bitcoin, la principal es Bitcoin Core [4], que ofrece el almacenamiento de toda la *blockchain* de Bitcoin para la emisión de transacciones directamente a la *blockchain* al momento de su creación.

2.1.1.9. Billeteras Jerárquicas Deterministas (*Hierarchical Deterministic Wallets*)

Las Billeteras Jerárquicas Deterministas, conocidas generalmente como *HDWallets* por su nombre en inglés, son un tipo de billeteras, usualmente generadas por software a partir de una clave privada maestra, y cuyas direcciones restantes pueden ser reconstruidas a partir de dicha clave maestra, al emplear un procedimiento determinista para la generación de nuevas direcciones [6].

En general, una *HDWallet* posee una estructura de árbol, en el que cada nodo posee una clave privada extendida y una clave pública. Cada nodo dentro de este árbol puede tener cualquier cantidad de nodos hijos. Otra particularidad de las *HDWallets* es que pueden servir como billeteras de distintas redes o criptomonedas, dada la estructura bajo la que se construye una *HDWallet*, ilustrada en la figura 2.1.1.9.1:

```
m / purpose' / coin_type' / account' / change / address_index
```

Figura 2.1.1.9.1: Estructura básica de una Billetera Jerárquica Determinista
Fuente: [6]

En este caso, la raíz, que indica el propósito, define el tipo de *HDWallet* creada de acuerdo a la Propuesta de Mejora de Bitcoin (*Bitcoin Improvement Proposal*, BIP). En la actualidad, las *HDWallets* que utilizan la estructura de árbol suelen ser creadas bajo BIP44 [6], y este es el propósito especificado en el *path* de la billetera. Lo siguiente es el tipo de moneda, que corresponde al segundo nivel del árbol. Gracias al comportamiento de la estructura de datos, cada nodo en una *HDWallet*

BIP44 puede tener múltiples hijos y esto permite que se puedan especificar diversos valores para los nodos en el segundo nivel, cada uno relacionado a un tipo de moneda diferente.

En el tercer nivel del árbol se manejan los nodos relacionados a cuentas, y nuevamente, gracias a la estructura del árbol, un nodo de un tipo de moneda puede tener como hijos distintos tipos de cuentas de la misma moneda, de forma similar al funcionamiento de distintos tipos de cuenta de banco equivalentes en moneda fiduciaria.

Finalmente se tiene el nivel de llaveros o *keychains*. En este nivel por lo general se generan dos nodos, uno con un camino 0 destinado a la generación de nuevas claves públicas y el camino 1, destinado a la creación de las direcciones de cambio o *change*, donde suele almacenarse por lo general los resultados de una transacción en forma de *Unspent Transaction Outputs* (UTXOs) o Salidas de Transacción sin gastar.

Cada nodo hoja de la estructura de árbol mencionada corresponde con una nueva dirección en la HDWallet. Al momento de realizar una transacción, la misma se firmará con una clave privada derivada asociada al nodo hoja involucrado en la transacción. Bajo el esquema de Billeteras Jerarquicas Deterministas, no se firman las transacciones con la llave maestra privada, pues ésta sirve únicamente para la reconstrucción de la billetera.

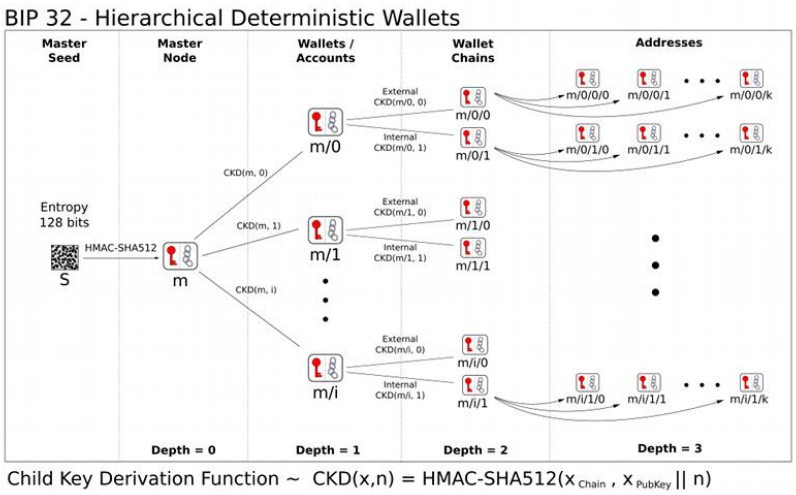

Figura 2.1.1.9.2: Estructura de una *HDWallet* de acuerdo a BIP 32
Fuente: Extraido de: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

2.1.1.10. Unspent Transaction Outputs (UTXOs)

El elemento fundamental para la construcción de una transacción en Bitcoin, o en cualquier criptomoneda que provenga de la *blockchain* de Bitcoin, son las salidas de transacción, o *Transaction Outputs*. La *Blockchain* lleva registro de todas las salidas disponibles para gastar, conocidas como *Unspent Transaction Outputs* (*UTXOs*) [7].

Una *Unspent Transaction Output*, o Salida de transacción sin gastar, indica la salida de una transacción que se le atribuye a una dirección específica de una billetera y que puede servir como entrada a futuras transacciones. En general, la mayoría de los sistemas y billeteras existentes, no solo de Bitcoin sino de criptomonedas alternativas, dependen ampliamente en la especificación de UTXOs para determinar si una billetera o dirección específica posee suficientes fondos para realizar una transacción, pues el balance de una billetera es la suma de todos las UTXOs que una billetera puede utilizar. Por lo general las billeteras utilizan un proveedor de servicio, o una base de datos para mantener un registro de las UTXOs que puede utilizar en nuevas transacciones con las direcciones que maneja.

Habitualmente, las UTXOs se toman de una dirección dependiendo del gasto que se vaya a realizar en la transacción. Es muy común que las UTXOs en una dirección no posean exactamente el balance requerido para una transacción, sino que en ocasiones, poseen fracciones mayores. Las UTXOs son indivisibles, es decir, deben ser gastadas en una única transacción, así que dado el caso en el que el valor de una UTXO supera al de la transacción, se genera una nueva UTXO en esta transacción con el cambio de retorno, y es alojado en una de las direcciones *change* de una determinada billetera. Por supuesto, al momento de generar la transacción hay que tener en cuenta las cuotas de transacción asociadas, por lo que la cuota de transacción es deducida de la UTXO a generar como retorno, de acuerdo a la siguiente ecuación:

UTXO de retorno = (Suma de UTXOs en la transacción) – (Cantidad de la transacción) – (Cuota de Transacción)

El sistema de UTXOs es un detalle importante de la *blockchain* como libro de cuentas, ya que en lugar de llevar registro de todas las transacciones crudas realizadas en la red, se puede llevar registro de los balances disponibles en forma de UTXOs y validar de esta forma cada una de las transacciones al tener un registro de las direcciones en la *blockchain* que poseen UTXOs asociadas. Este hecho las convierte en herramientas indispensables al momento de evitar el doble gasto, puesto que si se planea realizar una transacción con una salida (*Output*) que no se encuentre en el registro de UTXOs especificado por la *blockchain*, tanto el nodo que realiza la transmisión de la transacción como la red rechazarán la transacción [7].

2.1.2. Dash

Dash es una criptomoneda de código abierto, *peer-to-peer*, creada en el año 2014 por Evan Duffield bajo el nombre de XCoin. Después de varios renombramientos, Dash posee un amplio foco en la industria de pagos comerciales, tanto en forma presencial como en forma electrónica, puesto que ofrece una forma de dinero que es portable, económica, divisible y rápida. Al estar basada en Bitcoin posee una *blockchain*, sin embargo, el objetivo de Dash es convertirse en una moneda que permita hacer pagos escalables y accesibles para el usuario. Ofrece transacciones instantáneas bajo el nombre de *InstantSend* y transacciones privadas, llamadas *PrivateSend*, operando como una moneda autogobernable y permitiendo pagar a individuos y negocios que trabajen para añadir valor a la red.

Su esquema principal es el de una Organización Autónoma Descentralizada (*Decentralized Autonomous Organizations*, DAO) [8].

La red de Dash posee las siguientes características:

- Confirmación de transacciones instantánea
- Protección del doble gasto
- Anonimato al igual que la moneda física

2.1.2.1. Los Masternodes

Las características mencionadas anteriormente se basan en el uso de ***Masternodes***, nodos completos en una red P2P que permite que los pares utilicen los nodos para recibir actualizaciones sobre los eventos en la red. Estos nodos utilizan cantidades significativas de tráfico y otros recursos que aumentan el costo, por lo que la cantidad de este tipo de nodos suele ser reducida. No obstante, estos nodos son muy importantes para mantener la salud de la red, pues proveen a los clientes de la habilidad de sincronizar y facilitar la propagación rápida de mensajes a través de la red. En Dash, recibe el nombre de *Dash Masternode Network.*

Los *masternodes* en la red de Dash, deben proveer un nivel de servicio a la red y tener un vínculo de garantía para participar. Dicha garantía está segura mientras el *masternode* esté operativo, lo que permite a los operadores de este nodo proveer su servicio a la red, obtener ganancias por sus servicios y reducir la volatilidad de la moneda. Para ello, el operador de un *masternode* debe demostrar control sobre 1000 DASH.

2.1.2.2. Proof-of-Service

Los *masternodes* pueden proveer cualquier cantidad de servicios a la red, requiriendo que dichos nodos estén en línea, respondiendo y con el estado correcto de la *blockchain*. Todo el trabajo realizado para verificar la red y probar que los nodos se encuentran activos, se hace por la red de m*asternodes* propiamente. Aproximadamente el 1% de la red será verificada en cada bloque, lo que resulta en tener la totalidad de la red verificada cerca de 6 veces en un día. Para ello se utiliza un sistema de *Quorum*.

Los *masternodes* se propagan a través de la red utilizando una serie de extensiones de protocolo que incluyen un mensaje de anuncio y un mensaje de ping. Existen otros mensajes que sirven para ejecutar una solicitud de prueba de servicio, *PrivateSend* e *InstantSend*.

2.1.2.3. PrivateSend

De acuerdo con el *whitepaper* de Dash, *PrivateSend* es una versión extendida y mejorada de *CoinJoin*, con mejoras como descentralización y fuerte anonimato. *PrivateSend* busca que Dash como moneda sea anónimamente intercambiable, al igual que lo es cualquier moneda física fiduciaria, es decir, todas las monedas en sí deben ser iguales, sin poseer un historial de los usuarios que alguna vez han tenido dicha moneda.

PrivateSend utiliza el hecho de que una transacción puede ser formada por múltiples partes y puede enviarse a diferentes partes para unir fondos de forma que no puedan ser divididos después de la transacción. En primer lugar, *PrivateSend* divide las entradas de la transacción en varias denominaciones estándar de 0,01, 0,1, 1 y 10 DASH. A partir de esa división, la billetera envía un requerimiento a los *masternodes*, informado que el usuario se encuentra interesado en "mezclar" una cierta denominación. En este proceso, no se envía ninguna información a los *masternodes*.

Cuando dos usuarios más envían mensajes similares, indicando que desean mezclar la misma denominación, inicia una sesión de mezcla. El *masternode* mezcla las entradas e instruye a las billeteras de los tres usuarios para que paguen la entrada transformada a sí mismos. La billetera realiza dicho pago a una dirección diferente llamada dirección de cambio o *change address.*

Este proceso de mezcla ocurre entre 2 y 8 veces por denominación, el usuario indica el número de veces que desea mezclar. El proceso en sí ocurre en segundo plano, siendo transparente por el usuario y es instantáneo. *PrivateSend* está limitado a 1000 DASH por sesión, en donde cada sesión está limitada a 3 usuarios, por el esquema de requerimientos que utiliza.

2.1.2.4. InstantSend

InstantSend es la solución de Dash al problema de la espera por confirmación de un bloque al momento de realizar una transacción que poseen criptomonedas como Bitcoin. Nuevamente, esto es posible gracias a la existencia de los *masternodes*. Utilizando los *quorums* por parte de los *masternodes*, los usuarios son capaces de enviar y recibir transacciones instantáneas irreversibles. Una vez se ha formado el *quorum* y se decide que una transacción es válida, sus entradas se bloquean para ser gastadas únicamente en una transacción.

Dicho bloqueo de transacciones suele tomar aproximadamente 4 segundos. Si el consenso se alcanza en la red de *masternodes* durante un bloqueo de transacciones, todas las transacciones que tengan conflictos, así como los bloques que tengan conflictos serán rechazados a menos que tengan el identificador exacto de la transacción bloqueada en ese instante.

Esto permite que los vendedores utilicen dispositivos móviles en lugar de los puntos de venta tradicionales para actividades comerciales en la vida real y para acordar transacciones cara a cara como si se tratase de dinero tradicional, todo esto sin necesidad de contar con una autoridad centralizada.

2.1.2.5. Tasa de emisión y sistema de recompensa

Mientras Bitcoin reduce la emisión de su moneda a la mitad cada 4 años, Dash reduce la emisión en aproximadamente 7.14% cada 210.240 bloques, lo que equivale a poco más de un año, produciendo una transición más suave a una economía basada en cuotas de transacción.

En el sentido de la emisión total de monedas, se estima que Dash continuará emitiendo monedas por aproximadamente 192 años antes que un año completo de minado cree menos que 1 DASH. Es decir,

que después del 2209, solo se crearán 14 DASH adicionales y el último DASH será emitido, teóricamente en el año 2477.

En cuanto al sistema de recompensas, Dash retiene un 10% de las recompensas por creación de bloques para el mantenimiento de un presupuesto de Gobernanza Descentralizada y el restante, se divide en 50/50 entre el minero y el *masternode.* Dependiendo del uso del presupuesto disponible, la distribución queda entonces especificada en la tabla 2.1.2.5.1:

45%	Recompensa al minero
45%	Recompensa al *masternode* por su prueba de servicio
10%	Presupuesto para Gobernanza Descentralizada

Tabla 2.1.2.5.1: División de porcentajes de recompensa en la red Dash

2.1.2.6. El algoritmo X11

X11 es un algoritmo de hashing creado por Evan Duffield, que utiliza una secuencia de 11 algoritmos de hashing científicos (blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, echo) para la prueba de trabajo. De esta manera, la distribución del procesamiento es justa y las monedas se distribuyen en la misma forma en la que Bitcoin lo hacía originalmente. Las rondas de hashes se determinan *a priori* en lugar de ser escogidos aleatoriamente, por lo que X11 es uno de los métodos de hashing más seguros y sofisticados en la actualidad, puesto que a menos que los 11 hashes se rompan simultáneamente en un único ataque, la prueba de trabajo de Dash continuará funcionando.

2.1.2.7. Cuotas de transacción

Para diciembre del año 2017, luego del lanzamiento de Dash 0.12.2.0 y la reducción significativa de las cuotas por un factor de 10, mientras el tamaño del bloque aumentaba de 1MB a 2MB, se generó un esquema de cuotas como se observa en la tabla 2.1.2.7.1:

Transacción Estándar	.00001 DASH	Por kB de datos
InstantSend	.0001 DASH	Por entrada de transacción
PrivateSend	.001 DASH	Por 10 rondas de mezclado (promedio)

Tabla 2.1.2.7.1: Cuota de transacciones por tipo de transacciones en la red Dash

Cabe destacar que estas cuotas aplican independientemente del valor del dólar, del Dash o de la transacción en sí.

2.2. Esquemas de micropagos con criptomonedas

Uno de los atractivos más grandes del uso de criptomonedas es poder soportar estructuras económicas cuya base en moneda fiduciaria no sea lo suficientemente estable para un grupo poblacional, y que a su vez, permita a los usuarios pertenecientes a dicho grupo realizar transacciones de pequeño volumen, sin necesidad de emitir cuotas de transacciones demasiado altas que excedan el valor de la transacción, es decir, una evolución en el uso de criptomonedas en el día a día para transacciones comunes. Sin embargo, son pocas las criptomonedas que permiten esta funcionalidad en la actualidad, iniciando desde Bitcoin, la criptomoneda de mayor extensión y uso a nivel mundial.

Este tipo de pagos, que usualmente representan fracciones mínimas de Bitcoin o de otras criptomonedas de volumen similar, se denominan micropagos. Por lo general, suelen usarse este tipo de transacciones en actividades del día a día, compras de objetos pequeños, pagos por servicios simples o sencillos, cuyo valor en moneda fiduciaria llega a representar solamente una fracción del valor de una criptomoneda grande.

2.2.1. Micropagos

Los micropagos son esquemas de pago que enfatizan la habilidad de hacer pagos de pequeñas cantidades de alguna divisa. Las aplicaciones de los micropagos en el ámbito electrónico incluyen el pago por visita de páginas web, o por minuto de transmisiones de música bajo demanda, entre otros.

En principio, los micropagos pueden implementarse simplemente usando cheques electrónicos. El *merchant* y el usuario se encuentran ligados en la transacción por lo que el problema de este esquema viene dado, dependiendo de si se usa un enfoque centralizado o descentralizado, en el tiempo y costo del procesamiento de cualquier forma de pago en este pago [9].

En general, no existe una definición de qué tan pequeño puede ser un micropago, puede ubicarse en unos centavos de dólar o más allá, sin embargo, suele considerarse micropagos, aquellos en los que el monto a transar es superado usualmente por las cuotas de transacción.

2.2.2. Micropagos con criptomonedas

Bitcoin, siendo una de las criptomonedas principales, tiene cierto potencial en la actualidad para realizar micropagos, con las siguientes restricciones:

- La red Bitcoin posee varios algoritmos de *Anti-Flooding* para evitar que se hagan muchas transacciones muy rápido, por lo que hacer una gran cantidad de microtransacciones logrará que muchas de ellas bajen en prioridad.
- Hay un mínimo valor que puede enviarse en una transacción y es determinado por el tamaño de la transacción, en el que se toma en cuenta los bytes necesarios para reclamar el contenido de la transacción y las cuotas que se cargan a la misma.

- Al final, el receptor de los fondos termina con transacciones minúsculas que pueden ser difíciles de gastar tomando en cuenta las cuotas de transacción aplicadas

En la actualidad, existen diversos proyectos que tratan de disminuir la sobrecarga de las cuotas de transacciones y de la confirmación de estas. Como en general, la mayoría de las criptomonedas basan sus transacciones en la inserción de bloques en la *blockchain* hay proyectos que buscan realizar transacciones fuera de la cadena, buscando acelerar el proceso de validación.

2.2.3. Dash InstantSend

InstantSend de Dash funciona colocando banderas en las transacciones, con el fin de causar la selección determinista de un quorum de 10 *masternodes* por cada entrada gastada en una transacción de *InstantSend*. Los *masternodes* examinan la entrada y si la mayoría determina que tiene al menos 6 confirmaciones, aceptan la transacción. La entrada es entonces bloqueada hasta que la transacción sea confirmada en 6 bloques minados, punto en el que la salida puede ser usada como entrada en otra transacción *InstantSend*.

Esto difiere de las entradas usadas en transacciones normales, que pueden ser gastadas después de una sola confirmación independientemente de si el Dash se recibió a través de *InstantSend* o no. Este tipo de transacciones tiene una cuota ligeramente más alta y para ser usada, la billetera que recibe los fondos debe estar al tanto del uso de *InstantSend* para poder continuar con la transacción inmediatamente [8].

InstantSend suele activarse con un *checkbox* en las distintas aplicaciones para billeteras que posee Dash para smartphones o en Dash Core. En la siguiente tabla puede observarse el flujo de datos de *InstantSend* y el envío de mensajes entre el cliente y la red (Tabla 2.2.3.1).

Cliente InstantSend	**Dirección**	**Pares**	**Descripción**
`inv` message (ix)	→		Cliente envía inventario para solicitud de bloqueo de transacción
	←	`getdata` message (ix)	El par responde con un requerimiento de bloqueo de transacción
`ix` message	→		Cliente envía requerimiento de bloqueo por InstantSend
	←	`inv` message (txlvote)	Los masternodes en el quorum responden con sus votos
`getdata` message (txlvote)	→		El cliente solicita los votos
	←	`txlvote` message	El par responde con un voto

Tabla 2.2.3.1: Flujo de datos de una transacción *InstantSend*

Una vez se envía una solicitud de *InstantSend* el campo `instantsend` de varias llamadas remotas durante el procedimiento se coloca como verdadero. Si una transacción de *InstantSend* es válida pero no recibe un bloqueo de transacción, entonces se revierte a una transacción estándar en la red Dash. Por otro lado, para que una entrada sea utilizada en una transacción *InstantSend*, es necesario que tenga un cierto número de confirmaciones, dependiendo de la red en la que se realice (Tabla 2.2.3.2) [10]:

Red	**Confirmaciones requeridas**
Mainnet	6 bloques
Testnet	2 bloques
Regtest	2 bloques
Devnet	2 bloques

Tabla 2.2.3.2: Confirmaciones requeridas para una transacción *InstantSend*

2.3. Identificación por Radio Frecuencia (RFID)

La identificación por radiofrecuencia (*Radio Frequency Identification,* RFID), es una tecnología que se utiliza principalmente para la transmisión y captura de información contenida en diversos elementos llamados etiquetas o "*tags*". Esta captura se realiza utilizando señales de radiofrecuencia en un espectro determinado, que puede variar en un rango discreto que va desde los 135 KHz a los 2,5GHz [11].

Los primeros pasos hacia el desarrollo de la tecnología de Identificación por Radio Frecuencia datan del año 1940, con el sistema de identificación "*Friend or Foe*" utilizado por los militares estadounidenses durante la Segunda Guerra Mundial, en el que se podían identificar a distancia los aviones. Desde entonces, y hasta la década de los 80 cuando comenzó su implementación, la tecnología fue desarrollada constantemente.

Su estandarización comenzó en la década de los 90, con la aparición de diversas aplicaciones comerciales como identificación de automóviles, peajes, gestión de autopistas, y la creación en el año 1999 del centro Auto-ID del Instituto Tecnológico de Massachusetts, que en 2003 se convertiría en EPCGlobal, uno de los referentes relacionados a esta tecnología.

Desde entonces, se ha mejorado considerablemente la tecnología RFID y su costo ha bajado lo suficiente para ser incorporado a múltiples soluciones en casi cualquier ámbito. En combinación con distintos elementos de comunicación inalámbrica, la identificación por radiofrecuencia deriva en la aparición del esquema NFC (*Near Field Communication*), que logra concentrar en un solo elemento un receptor y un transmisor de radiofrecuencia que puede ser incorporado a dispositivos móviles y que puede lograr la comunicación entre dispositivos de este tipo a una distancia máxima de 4 cm [12].

De esta forma, es posible observar cómo la tecnología RFID ha influenciado diversas actividades en el mundo comercial y se ha convertido en elemento fundamental en el uso diario de la sociedad, por su alta capacidad de almacenamiento y comodidad de uso. Para ahondar un poco más en ello, es necesario establecer una definición formal de la tecnología y las diversas implementaciones físicas que la componen y estructuran su funcionamiento.

En la actualidad es una práctica estandarizada la utilización de un mecanismo de etiquetado o identificación en diversos elementos que permita obtener información básica, pero importante, sobre un objeto determinado. Los códigos de barras fueron y siguen siendo un elemento clave para este tipo de prácticas, sin embargo, éstos requieren que el objeto en sí se encuentre a milímetros de distancia y a línea de vista del lector. Solventando este pequeño obstáculo, aparece la tecnología de tarjetas inteligentes con contactos, muy extendida a nivel de las tarjetas bancarias, pero requiere del contacto permanente entre el lector y la tarjeta mientras se realiza la transacción [11].

La tecnología RFID expande los límites generados por el uso de códigos de barras y tarjetas de contactos, puesto que soporta un conjunto más amplio de identificadores a almacenar, y de igual forma, puede guardar metadatos sobre el objeto e incluso información que puede ser recolectada a través de sensores alrededor del mismo, en tiempo real. No obstante, la ventaja más grande que posee la tecnología de identificación por radiofrecuencia es la minimización o la eliminación del contacto entre un lector y un objeto a identificar o etiquetar, a relativo bajo costo, lo cual ha acelerado su crecimiento en los últimos años.

La tecnología RFID requiere para su funcionamiento los elementos que serán posteriormente explicados y que son mostrados en la Figura 2.3.1:

- Etiqueta o Transpondedor
- Lector o interrogador
- Elementos programadores
- Sistema anfitrión

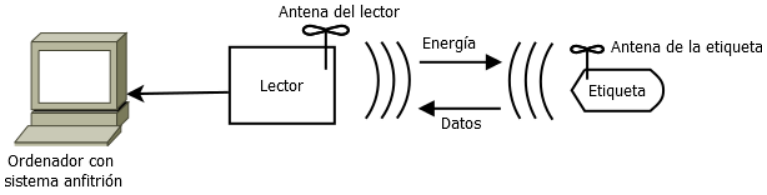

Figura 2.3.1: Estructura de un sistema RFID pasivo
Fuente: Elaboración propia – 2019

Por lo general, los sistemas RFID suelen tener estructuras distintas que dependen de varios factores dados por forma de alimentación, el esquema de programación, el protocolo de comunicación, el espectro de radiofrecuencia bajo el que funcionan y según su principio de propagación.

Sin embargo, el funcionamiento básico de los sistemas es similar en todos los casos:

- Se equipa a todos los objetos a identificar, controlar o seguir, con una etiqueta RFID.
- La antena del lector o interrogador emite un campo de radiofrecuencia que activa las etiquetas.
- Cuando una etiqueta ingresa en dicho campo utiliza la energía y la referencia temporal recibidas para realizar la transmisión de los datos almacenados en su memoria. En el caso de etiquetas activas la energía necesaria para la transmisión proviene de la batería de la propia etiqueta.
- El lector recibe los datos y los envía al ordenador de control para su procesamiento [12].

2.3.1. Transpondedores

El transpondedor es el dispositivo que se encuentra embebido en las etiquetas o *tags* utilizados en un sistema RFID. Se compone principalmente de un microchip en el que se encuentra la circuitería asociada a su lógica de control, seguridad y procesamiento, así como una memoria para el almacenamiento de datos y una circuitería analógica para la alimentación y transferencia de datos. Así mismo, posee una antena para la transmisión de datos que puede ser de dos tipos: de bobina o un dipolo [12].

2.3.1.1. Formatos de construcción

La forma común de ensamblar un elemento transpondedor de RFID, es colocar la circuitería y la antena sobre un material soporte que, en principio, no impida la transmisión de señales y pueda ser insertado en diversos medios. Los medios pueden variar dependiendo de su uso.

- **Cubiertas tipo "Moneda":** Este tipo de encapsulamiento utiliza discos con forma de monedas con un diámetro que varía entre los milímetros hasta 10 centímetros. Como material se pueden utilizar resinas epoxy o poliestireno, para un manejo más eficiente de las temperaturas.
- **Cubiertas de vidrio:** Estas han sido diseñadas específicamente para ser inyectadas bajo la piel de un animal, con propósitos de identificación. Los tubos de vidrio, de 12 a 32 mm de longitud (ver Figura 2.3.1.1.1), contienen un microchip montado sobre un soporte y un capacitor para facilitar la alimentación de corriente. La bobina del transpondedor incorpora un cable de 0,03 mm de grosor en un núcleo de ferrita, y los componentes internos están embebidos en un adhesivo suave para mayor estabilidad mecánica.

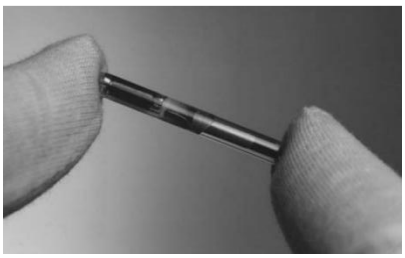

Figura 2.3.1.1.1: Transpondedor en capsula de vidrio
Fuente: [12]

- **Cubiertas de plástico:** El desarrollo de este tipo de cubiertas fue pensado para aplicaciones de alta demanda mecánica, y puede ser fácilmente integrable en otros productos, como, por ejemplo, llaves de automóviles. Los transpondedores que usan estas cubiertas suelen tener una bobina más larga, pueden usar chips más grandes y tienen mayor tolerancia a vibraciones mecánicas, lo que las hace ideales para el mercado automovilístico.
- **Tarjetas inteligentes:** Estas tarjetas se presentan en un formato que tienen un tamaño idéntico al de las tarjetas de crédito (85,72 × 54,03 × 0,76 mm), y su ventaja principal radica en poseer una bobina de área incluso más extensa que los transpondedores con cubierta de plástico, lo que hace que el rango de las tarjetas sea igual de extenso. La producción de estas tarjetas se lleva a cabo laminando un transpondedor entre 4 láminas de PVC[4]. Las láminas individuales se hornean a alta presión y temperaturas sobre los 100° para producir una unión permanente entre las láminas (Figura 2.3.1.1.2).

Figura 2.3.1.1.2: Formato de tarjetas/botones inteligentes
Fuente: [11]

- **Etiquetas inteligentes:** Es un formato en el que la bobina del transpondedor se aplica a una lámina de plástico de 0.1 mm de grosor, ya sea vía impresión o grabado. Esta lámina usualmente tiene una pequeña cubierta adhesiva en la parte trasera y suelen ser bastante económicos por el material de fabricación, al igual que convenientes por su delgadez y flexibilidad. Suelen usarse en productos de consumo cotidiano o en elementos que requieran una forma de etiquetado compacta, de manera muy similar a un etiquetado por código de barras.

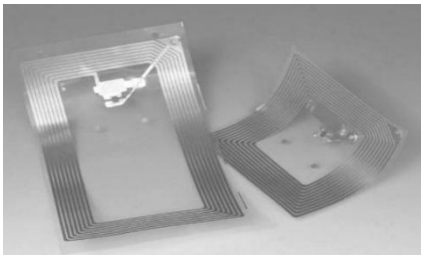

Figura 2.3.1.1.3: Estructura de etiquetas inteligentes
Fuente: [12]

Dado el material, se debe especificar los parámetros asociados a la transmisión de los datos entre los distintos elementos.

[4] Policloruro de Vinilo, es una resina termoplástica obtenida a través de la polimerización de un 57% de Cloro, y 43% carbón. (http://www.pvc.org/en/p/what-is-pvc)

2.3.1.2. Frecuencia, Rango y Acoplamiento

El rango operativo de frecuencia, el rango de distancias y el sistema de acoplamiento físico son, probablemente, los elementos diferenciadores más importantes en un sistema de identificación por radiofrecuencia. Los sistemas RFID suelen operar en un rango amplio de frecuencias que va desde los 135 kHz de onda larga a los 5,8 GHz en el rango de microondas. En cuanto al acoplamiento físico, se utilizan campos eléctricos, magnéticos y electromagnéticos para la transmisión de energía y datos en el sistema, en tanto que, para el rango físico de distancias, pueden variar entre unos pocos milímetros hasta poco más de 15 metros.

Los sistemas en los que el rango varía de milímetros a 1 cm se conocen como ***Sistemas de Acoplamiento Corto***, y suelen tener el transpondedor insertado en el lector, o posicionado en una superficie muy cerca del lector dedicada a este fin. Para su acoplamiento, utilizan tanto campos eléctricos como magnéticos y pueden ser operados, teóricamente a cualquier frecuencia entre DC[5] y 30 MHz, puesto que la operación del transpondedor no depende de la radiación de los campos. Esto facilita la provisión de grandes cantidades de energía, permitiendo la operación de un microprocesador con un esquema de consumo de energía no-óptimo. Se usan principalmente en aplicaciones que están sujetas a requerimientos estrictos de seguridad, pero no requieren rangos extensos, como es el caso de sistemas de cerradura electrónicos o sistemas de tarjetas inteligentes para funciones de pago.

Por otra parte, los ***Sistemas de Acoplamiento Remoto***, cuyo rango de distancia llega hasta 1 metro, se basan en un sistema de acoplamiento inductivo magnético entre el lector y el transpondedor (Sistemas inductivos de radio), o sistemas con acoplamiento por proximidad, como es el caso de las tarjetas inteligentes o algunos sistemas de identificación de animales que trabajan bajo el estándar ISO/IEC[6] 14443. Estas tarjetas suelen emplear frecuencias entre los 135 kHz y 13,56 MHz para la transmisión. Su estructura puede visualizarse en la figura 2.3.1.2.1, a continuación:

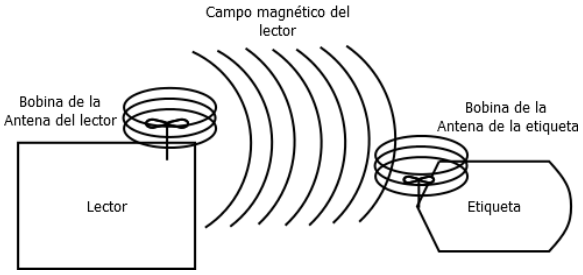

Figura 2.3.1.2.1: Esquema de acoplamiento inductivo
Fuente: Elaboración propia - 2019

[5] DC: Frecuencia de 0 Hertz.

[6] *International Organization for Standardization*, organización encargada de desarrollar estándares de productos, materiales, procesos y servicios a nivel internacional. (https://www.iso.org/standards.html) e International Electrotechnical Commission, organización encargada de publicar los estándares relacionados a dispositivos eléctricos o electrónicos. (https://www.iec.ch/about/)

Para la transmisión de datos, la energía administrada por el lector a través del campo electromagnético que éste despliega es la única fuente de alimentación necesaria por parte del transpondedor.

2.3.1.3. Fuentes de alimentación

Es importante notar que la fuente de alimentación de un transpondedor juega un rol importante en la clasificación del sistema general y en la selección de la aplicación en la que será utilizado dicho sistema.

En principio, los ***transpondedores pasivos*** no poseen ninguna fuente de energía propia. Estos elementos obtienen toda la energía requerida para operar desde el campo magnético o electromagnético formado por el lector, a través de la antena del transpondedor. Para realizar la transmisión de los datos del transpondedor al lector, el campo de este último puede ser modulado, o el transpondedor puede implementar un almacenamiento corto de energía proveniente del lector, lo que significa que la energía emitida por el lector es utilizada para transmisión de datos en ambos sentidos. Si el transpondedor se encuentra fuera del rango del lector, no tiene ninguna fuente de energía y no será capaz de enviar señales.

Los ***transpondedores activos*** tienen su propia fuente de energía en forma de una batería o incluso una celda solar. Esta fuente de energía provee voltaje al chip, por lo que el campo magnético o electromagnético generado por el lector no es utilizado para darle energía al transpondedor, sino para la transmisión de datos, por lo que dicho campo puede ser significativamente más débil que uno requerido por un transpondedor pasivo. La ventaja de este enfoque es que se puede incrementar el rango de comunicación ya que la señal del lector se usaría solo para la transmisión de datos. La comparación de la estructura de ambos enfoques puede observarse en la figura 2.3.1.3.1:

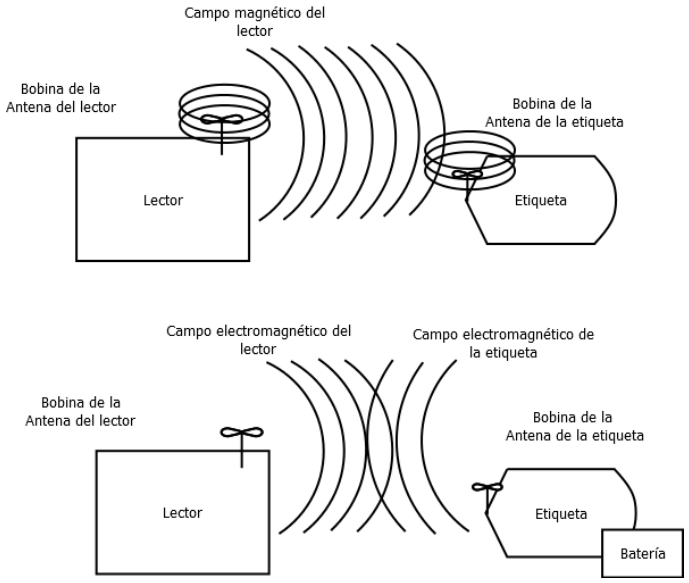

Figura 2.3.1.3.1: Comparación entre Transpondedores activos y pasivos
Fuente: Elaboración propia - 2019

2.3.1.4. Procesamiento de información y almacenamiento de datos

Dependiendo de la calidad del sistema, es posible obtener al menos 2 categorías de sistemas RFID.

Comenzando desde los de más bajo nivel, existen los transpondedores de solo lectura. Este tipo de elementos tienen un único número de serial compuesto de varios bytes y, además, tiene un conjunto de datos codificado permanentemente en su memoria. Al ser colocado en el campo de radiofrecuencia de un lector, el transpondedor iniciará la transmisión de su propio número de serial. En el caso de los sistemas de solo lectura, es necesario asegurarse que solo un transpondedor se encuentra en la zona de interrogación del lector, de lo contrario, se presentaría colisión de datos entre las transmisiones de varios transpondedores. Sin embargo, gracias a su funcionamiento simple, estos sistemas proveen bajo consumo de energía y bajo costo de fabricación.

Los sistemas de solo lectura se usan en aplicaciones que requieren poco uso de datos y operan a todas las frecuencias disponibles para sistemas RFID, con rangos de lectura bastante altos gracias al bajo consumo de energía del microchip que compone al transpondedor.

Al aumentar la capacidad de memoria y añadir la funcionalidad de escritura, los sistemas RFID de lectura/escritura poseen tamaños de memoria que varían desde unos cuantos bytes hasta 100 kB. En el caso de los transpondedores pasivos, se utiliza una memoria EEPROM[7], en tanto que los transpondedores activos utilizan una memoria SRAM[8]. En ambos casos, los transpondedores son capaces de procesar comandos simples desde el lector para una lectura selectiva y escritura de los datos a la memoria en una máquina de estados permanentemente codificada. En general, también soportan procedimientos anticolisiones, de forma que varios transpondedores de este tipo pueden encontrarse en la zona de radiofrecuencia de un único lector sin interferir entre sí, y dicho lector será capaz de discernir entre cada uno de los transpondedores.

Los sistemas de más alto nivel poseen incorporados un microprocesador y un sistema operativo para un manejo más complejo de los protocolos y algoritmos de encriptación y autenticación. Estos sistemas, que operan casi exclusivamente en la frecuencia de 13,56 MHz, suelen usarse en aplicaciones que impliquen altos estándares de encriptación de datos, como billeteras electrónicas y sistemas de billetería (*ticketing*). La descripción de la transmisión de datos entre el transpondedor y el lector se especifica a profundidad en el estándar ISO/IEC 14443 [12].

2.3.2. Lectores

Los lectores en un sistema RFID funcionan como interfaz de entrada de datos a una aplicación. Ya que todo el sistema funciona en una arquitectura Maestro-Esclavo, la aplicación tiene el rol maestro, en tanto que el lector tiene el rol esclavo, esperando por instrucciones de lectura y/o escritura por

[7] *Electrically Erasable Programmable Read-Only Memory*, memorias de solo lectura que pueden ser reprogramadas aplicando un voltaje superior al que requieren para su funcionamiento usual. (https://whatis.techtarget.com/search/query?q=EEPROM)

[8] *Static Random Access Memory*, memorias de acceso aleatorio que retienen su información mientras sean alimentadas eléctricamente. (https://whatis.techtarget.com/search/query?q=SRAM)

parte de la aplicación. A partir de estas instrucciones, se inicia una serie de comunicaciones entre el lector y el transpondedor para activar la transferencia de datos y estructurar la secuencia de funcionamiento. A continuación, la figura 2.3.2.1 ilustra un esquema básico de la arquitectura maestro-esclavo en los distintos niveles de un sistema RFID.

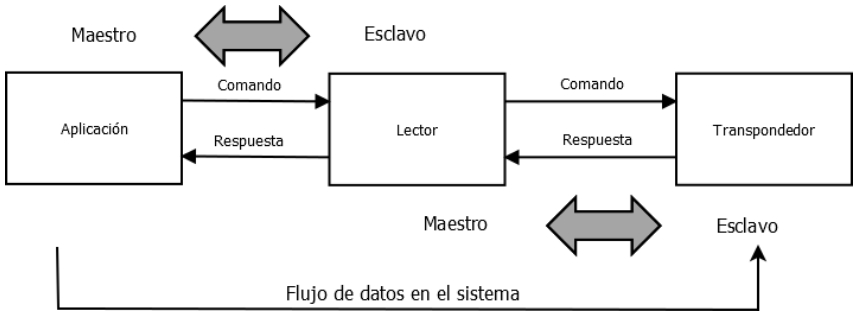

Figura 2.3.2.1: Esquema Maestro-Esclavo en un sistema RFID
Fuente: Elaboración propia – 2019

La estructura básica de un lector, en todos los sistemas, se reduce a una interfaz de radiofrecuencia (transmisor y receptor), una unidad de control y una antena [11]. Asimismo, también incluye una interfaz que se comunica con un controlador o sistema anfitrión, a través de un enlace local o remoto, que permite enviar los datos del transpondedor al sistema anfitrión [12].

2.3.2.1. Interfaz de Radiofrecuencia

La interfaz de radiofrecuencia de un lector es capaz de cumplir las siguientes funciones:

- Generar poder de transmisión de alta frecuencia para activar el transpondedor y proveerle energía.
- Modulación de la señal de transmisión para enviar datos al transpondedor.
- Recepción y demodulación de las señales de radiofrecuencia transmitidas por el transpondedor.

Para ello, la interfaz (Figura 2.3.2.1.1) contiene dos rutas establecidas para las señales, de forma que éstas correspondan con los dos sentidos del flujo de datos, desde y hacia el transpondedor, que procesan las señales enviadas desde el transpondedor y enviadas hacia él [11].

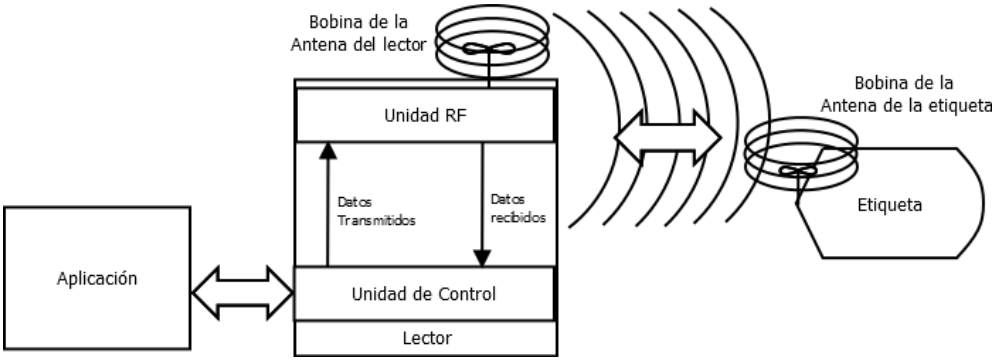

Figura 2.3.2.1.1: Interfaz completa de un sistema RFID por acoplamiento inductivo
Fuente: Elaboración propia - 2019

2.3.2.2. Unidad de Control

La unidad de control está conformada por un microprocesador que, en ocasiones, es acompañado por un circuito integrado ASIC[9] que depende de los requerimientos específicos de la aplicación y ayuda a realizar algunos cálculos adicionales [12].

Con estos componentes, la unidad de control cumple las siguientes funciones:

- Comunicación con el software de la aplicación y la ejecución de comandos desde dicho software.
- Control de la comunicación con el transpondedor según el principio Maestro-Esclavo.
- Gestión del acceso al medio y activación de las etiquetas.
- Codificación y decodificación de la señal.
- Ejecución de un algoritmo anticolisión.
- Encriptación y desencriptación de los datos a ser transmitidos entre el lector y el transpondedor.
- Ejecución de autenticación entre el lector y el transpondedor.

De todas estas funciones, la autenticación, cifrado y codificación de la señal son realizadas por el módulo ASIC, que se comunica con el microprocesador principal utilizando un bus dedicado (orientado a registros). Por otro lado, el intercambio de datos entre la aplicación y la unidad de control normalmente es realizado por una interfaz RS232 o RS485[10]. Para evitar errores de transmisión se utilizan diversos procedimientos de comprobación como LRC[11] y CRC[12] [11].

Como fue acotado en la sección referente a transpondedores, el número de transpondedores que un lector puede identificar depende de la frecuencia de trabajo, la calidad de los componentes y el protocolo de trabajo utilizado, en especial a partir de las bandas de Alta Frecuencia (HF), en donde el número de etiquetas puede variar entre 50 y 200 etiquetas por segundo [12].

2.3.2.3. Detección y lectura

Una vez una etiqueta es detectada y seleccionada, el lector procede a realizar operaciones de interrogación sobre la misma, o incluso operaciones de escritura. Para este tipo de operaciones,

[9] *Application Specific Integrated Circuit*, microchip diseñado para una aplicación o protocolo específico. (https://whatis.techtarget.com/definition/ASIC-application-specific-integrated-circuit)

[10] RS232, RS485: Estándares de transmisión serial de datos entre dos dispositivos.

[11] *Longitudinal Redundancy Check*, método que verifica la validez de datos transmitidos utilizando bits de paridad (https://www.techopedia.com/definition/1800/longitudinal-redundancy-check-lrc)

[12] *Cyclic Redundancy Check*, método que verifica la validez de datos transmitidos utilizando división polinomial binaria (https://www.techopedia.com/definition/1793/cyclic-redundancy-check-crc)

existen algoritmos que determinan el orden en el que estas operaciones serán realizadas y el número de etiquetas sobre el que actúan. Existen algoritmos como el Protocolo Orden-Respuesta, en el que el lector ordena a un transpondedor que cese su transmisión cuando reconoce que ya ha recibido la información. Otro método, el Sondeo Selectivo, hace que el lector busque específicamente las etiquetas, que tienen un identificador determinado y la interroga por turnos.

Los lectores pueden variar de complejidad dependiendo del tipo de transpondedor que tengan que alimentar y de las funciones que deban desarrollar. Por ende, se pueden dividir en fijos y móviles.

- Los lectores fijos se posicionan en lugares estratégicos como puertas de acceso, lugares de paso o puntos críticos dentro de una cadena de ensamblaje, sirven principalmente para la tarea de monitoreo.
- Los lectores móviles, por otro lado, suelen ser dispositivos de mano. A veces incorporan una pantalla y un teclado para introducir datos y una antena integrada. Por lo general, su radio de cobertura es menor.

En resumen, los parámetros que caracterizan un lector RFID son:

- Frecuencia de operación: Puede funcionar a baja frecuencia, alta frecuencia, ultra alta frecuencia y frecuencia de microondas.
- Protocolo de funcionamiento: Pueden funcionar tanto con los estándares ISO/IEC como con protocolos propietarios, sin embargo, no admiten todos los protocolos existentes.
- Tipo de regulación que siguen: Existen distintas regulaciones de frecuencia en Estados Unidos y Europa.
- Interfaz con el sistema anfitrión: Se pueden conectar al sistema usando TCP/IP[13], WLAN[14], Ethernet[15], o conexión en serie RS232 o RS485.
- Capacidad para actualizar el software del lector online: Puede hacerse vía Internet o vía interfaz directa con el software anfitrión.
- Capacidad para gestionar múltiples antenas: Normalmente se usan 4 antenas por lector.
- Capacidad para interactuar con otros productos de middleware.
- Entrada y salida: Con una interfaz I/O pueden conectar otros dispositivos como sensores o circuitos de control adicionales [12].

2.3.3. ISO/IEC 14443 - Proximity Integrated Circuit Cards (PICC)

El estándar ISO/IEC 14443, titulado "*Identification cards - Proximity integrated circuit(s) cards*" o "Tarjetas de identificación - Tarjetas de Circuitos Integrados por Proximidad", describe el método y

[13]Suite de protocolos de Internet que modelan el funcionamiento y la interconexión de dispositivos basados en Internet. (https://searchnetworking.techtarget.com/definition/TCP-IP)

[14] Red de área local inalámbrica

[15]Tecnología y protocolo para conectar a nivel físico redes de área local (LAN). (https://searchnetworking.techtarget.com/definition/Ethernet)

los parámetros de operación de las tarjetas inteligentes que usan acoplamiento por proximidad. Esta definición cubre las tarjetas inteligentes cuya cobertura es de un rango aproximado de 7 a 15 centímetros, y usualmente cuentan únicamente con un microprocesador y contactos adicionales.

2.3.3.1. Características Físicas

La parte 1 del estándar define las propiedades mecánicas de las tarjetas inteligentes. Las dimensiones corresponden con los valores especificados en el estándar ISO/IEC 7810, es decir, las medidas de 85,72 x 54,02 x 0,76 m, más o menos un margen de tolerancia [12].

2.3.3.2. Principios Operativos

En el caso de las Tarjetas de Circuitos Integrados por Proximidad, en adelante PICC por sus siglas en inglés, la interfaz se crea utilizando acoplamiento inductivo. Este tipo de acoplamiento funciona encapsulando un microchip, que almacena los datos, y que contiene una bobina que sirve de antena.

Un campo magnético alternante es producido por una corriente sinusoidal que fluye a través de un bucle formado por la antena del lector. Cuando la tarjeta entra en el rango, se induce una corriente alterna en el bucle que forma la antena de la tarjeta. El circuito integrado de las PICC contiene un rectificador y un regulador de poder para convertir la corriente alterna a corriente directa, y alimentar el circuito integrado.

Por otro lado, el lector hace una modulación de amplitud[16] al campo de radiofrecuencia para enviar información a la tarjeta. El circuito integrado contiene un demodulador que convierte la señal con amplitud modulada (AM) a señales digitales. El circuito integrado también contiene un circuito de extracción de reloj que produce un reloj digital de 13,56 MHz para su uso interno. Los datos del lector son registrados, decodificados y procesados por el circuito integrado. Además, el circuito integrado se comunica con el lector modulando la carga en la antena de la tarjeta, que a su vez modula la carga en la antena del lector. Las PICC bajo el estándar ISO/IEC 14443 usan una señal subportadora[17] de 847,5 kHz para la modulación de carga, que permite al lector filtrar la frecuencia de la subportadora de su antena y decodificar los datos [13]

2.3.4. Trabajos Relacionados

Durante los últimos años se han desarrollado diversos trabajos en pro de crear soluciones que agilicen la realización de micropagos a nivel comercial. De igual forma, con el nacimiento de las criptomonedas, es de esperar que en la búsqueda de aceptación a través del uso en actividades del día

[16] Técnica de modulación para transmitir datos en una señal portadora (https://whatis.techtarget.com/definition/amplitude-modulation-AM)

[17] Señal portadora sobre una señal portadora que son demoduladas por separado (https://searchnetworking.techtarget.com/definition/subcarrier)

a día, se destinen esfuerzos en crear iniciativas que acerquen el uso de criptomonedas a los usuarios comunes. Analizando estos trabajos relacionados se puede observar que puede existir una relación entre agilizar formas de pago utilizando la tecnología de identificación por radiofrecuencia, y la búsqueda de rapidez en las transacciones que involucran criptomonedas hoy en día.

2.3.5. Sistemas de pago abiertos

Los sistemas de pago abiertos fueron implementados utilizando el estándar ISO/IEC 14443 y fueron extendidos por estándares globales surgidos por las especificaciones EMV (*Europay, Mastercard y Visa*). Este tipo de sistemas, que ya han sido puestos en práctica por Mastercard y Visa desde el año 2005, requieren que la tarjeta inteligente no se encuentre a más de 200 metros del lector. En cuanto a los puntos de venta (*Point-Of-Sale*, POS), se implementa un terminal POS que lee los datos de la tarjeta, verifica los datos y los envía para su procesamiento transaccional a alguna entidad bancaria relacionada, así que, en principio, no funciona de manera muy diferente a un punto de venta tradicional.

Sin embargo, este sistema de pago abierto busca aún mayor rapidez en los sistemas de pago, por lo que los datos transmitidos se reducen a los mínimos requeridos para lograr esos tiempos de transacción más cortos.

El terminal POS es un lector RFID equipado con elementos electrónicos de seguridad y conectado en línea a una red de comunicaciones a la entidad bancaria del ente que recibe el pago. Durante las transacciones de pago, el terminal POS envía, después de chequear los datos obtenidos desde la tarjeta inteligente, un conjunto de datos al banco, que contiene la fecha de compra, los datos de la tarjeta y la fecha de vencimiento. Luego, se envía una solicitud al banco del cliente para decidir si la tarjeta y la transacción son aceptadas o no. El ente que recibe el pago también envía una solicitud al sistema de pago (Mastercard o Visa) para establecer una conexión con el banco del cliente, quién decidirá si la transacción es válida y autoriza el proceso. Luego de dicha autorización, el proceso de pago desde el punto de vista del cliente está realizado.

La diferencia principal radica entonces en el manejo de la transacción entre el terminal POS y la tarjeta en sí. Por la gran cantidad de datos que pueden transmitirse entre el transpondedor y el lector en un sistema RFID, es necesario realizar verificaciones y asegurar criptográficamente estos datos, usando métodos como RSA para determinar la autenticidad de la tarjeta. Estos métodos también se definen adecuadamente en las especificaciones de la EMV.

Este esquema ya ha sido aplicado por diversas organizaciones de tarjetas de crédito como American Express (llamado *ExpressPay* en Estados Unidos), MasterCard (llamado *PayPass*) y Visa (llamado *Visa Wave* en Asia), con una respuesta positiva por parte de los usuarios y que fue extendido de igual forma a diversos servicios específicos que requieran pagos inmediatos, como en el caso de ExxonMobil para el pago de servicios en estaciones de gasolina, bajo el nombre de *SpeedPass*, por lo que vemos que el foco principal de estos sistemas es la realización de pagos casi instantáneos.[12]

2.3.6. BioCrypt

BioCrypt Technologies Inc. es una compañía con la iniciativa de llevar las funcionalidades de *blockchain* al alcance de los usuarios utilizando tanto la tecnología RFID como NFC, a través de un producto llamado BioCrypt posee un token llamado BIO, cuyas transacciones son validadas a través de un contrato inteligente para verificación de identidad y autorización de transacciones. Se utiliza una blockchain para tener registro público de las transacciones realizadas con BioCrypt, que físicamente se generan utilizando dispositivos llamados BioBands, que son bandas de muñeca que poseen tecnología NFC para comunicarse con otros dispositivos compatibles a distancias de 4 cm. BioCrypt también aspira el uso de BioChip, una etiqueta RFID que puede ser implantado para usarlo con la finalidad de tener un acceso más rápido a los contratos realizados bajo BioCrypt y por supuesto, a los tokens disponibles en su respectiva billetera [14].

Figura 2.3.6.1: Esquema RFID de BioCrypt y su BioBand
Fuente: [14]

2.3.7. Dash Text

Dash Text es una iniciativa venezolana creada con el fin de manejar una billetera de Dash y realizar transacciones con la misma a través de mensajes de texto. Dash Text funciona por el momento con proveedores de servicio de telefonía móvil en Venezuela y permite crear una billetera en su plataforma sin costo alguno, enviando la palabra “CREAR” a cierto número designado.

Figura 2.3.7.1: Creación de una billetera electrónica utilizando Dash Text
Fuente: [15]

Con ello, permite el envío y, por ende, las transacciones de Dash utilizando simplemente el número celular de los entes envueltos en la transacción. De acuerdo con su página web, Dash Text cuenta en la actualidad con aproximadamente 3400 usuarios sobre el territorio venezolano, ampliando los horizontes de funcionamiento de las criptomonedas en aquellas áreas del territorio que no cuentan con conexión a Internet o conexiones de datos móviles, funcionando enteramente sobre la plataforma de telefonía móvil [15].

Figura 2.3.7.2: Transacciones realizadas vía Dash Text
Fuente: [15]

3. MARCO APLICATIVO

En este capítulo se definen las metodologías usadas a nivel de investigación, diseño y programación de la aplicación descrita en este Trabajo Especial de Grado, así como las herramientas de hardware y software utilizadas para su desarrollo.

3.1. Metodología General de Trabajo

A continuación, se describe en detalle la metodología o entorno de trabajo SCRUM, las fases que lo comprenden y cómo estas se adaptan a un proceso de desarrollo de software al proveer un entorno de trabajo ágil, flexible, iterativo e incremental.

3.1.1. SCRUM

De acuerdo con T. Satpathy [16], SCRUM es un entorno de trabajo adaptativo, iterativo, rápido y flexible diseñado para otorgar valor significativo a un proyecto, por lo que asegura transparencia en la comunicación entre equipos y crea un entorno de responsabilidad colectiva y progreso continuo.

La fortaleza de SCRUM radica en que, a pesar de poseer cierta flexibilidad a la hora de establecer los procesos necesarios para su ejecución, sus posiciones bien definidas dentro de un equipo de trabajo permiten una generación de requerimientos ágil y el inicio rápido del desarrollo para un mínimo producto viable en cada iteración.

3.1.1.1. Roles principales de SCRUM.

- ***Product Owner***: El *Product Owner* o Dueño del Producto, es la persona encargada de lograr el máximo valor del negocio para el producto o servicio a desarrollar en el proyecto, por ello, se requiere que articule correctamente los requerimientos del producto, representando así la voz del cliente dentro del equipo de trabajo.
- ***SCRUM Maste***r: Es un facilitador que se asegura que el *SCRUM team* o equipo de desarrollo pueda laborar en un entorno que los dirija a la conclusión exitosa del proyecto, siguiendo los procesos definidos en el entorno SCRUM.
- ***SCRUM Team***: El equipo SCRUM es el grupo de personas responsables de entender los requerimientos especificados por el *Product Owner* y crear o desarrollar los entregables del proyecto que serán mostrados al cliente al final de cada *Sprint* o incremento del proyecto.

3.1.1.2. Procesos de Scrum

Los procesos de SCRUM abordan las actividades específicas y el flujo a seguir al trabajar en un proyecto. En general, suele haber 19 procesos fundamentales que aplican a casi cualquier proyecto que se desarrolle bajo el entorno SCRUM, y se agrupan en 5 fases principales, como se muestra en la tabla 3.1.1.2.1 a continuación:

FASE	PROCESO SCRUM
INICIACIÓN	1. Crear la visión de proyecto 2. Identificar al *Scrum Master* y propietarios del producto 3. Formar el *Scrum Team* 4. Desarrollar las *Epics* a cumplir 5. Crear el *Product Backlog* Priorizado 6. Llevar a cabo la planificación de la(s) liberaciones de producto
PLANIFICIACIÓN Y ESTIMACIÓN	7. Crear las Historias de Usuario 8. Estimar las Historias de Usuario 9. Asignar las Historias de Usuario 10. Identificar las tareas 11. Estimar las tareas 12. Crear el *Sprint Backlog*
IMPLEMENTACIÓN	13. Crear los Entregables 14. Llevar las reuniones diarias 15. Refinar el *Product Backlog* Priorizado
REVISIÓN Y RETROSPECTIVA	16. Demostrar y validar el *Sprint* 17. Hacer retrospectiva del *Sprint*

LIBERACIÓN	18. Entregar Producto 19. Hacer Retrospectiva del proyecto

Tabla 3.1.1.2.1: Fases de Scrum

3.1.1.3. Time Boxes de SCRUM

- ***Sprint***: Un *Sprint* es una iteración cuya duración varía entre 1 y 6 semanas, en el que el *Scrum Master* y el *Scrum Team* trabajan en conjunto para convertir los requerimientos del *Product Backlog* en funcionalidades concretas del producto. Durante los *Sprints* se desarrollan incrementos del producto, es decir, todas las funcionalidades especificadas en el Sprint Backlog para el avance en la funcionalidad del producto final.

- **Reuniones Diarias**: Son reuniones cortas cuya duración varía entre 15 y 30 minutos, en el que los miembros del equipo se reúnen para reportar el progreso del proyecto respondiendo las siguientes preguntas:

  - ¿Qué he hecho desde la última reunión?
  - ¿Qué planeo hacer antes de la próxima reunión?
  - ¿Qué obstáculos o impedimentos tengo para completar mis tareas?

- **Reunión de Planificación de *Sprint***: Esta reunión suele llevarse a cabo antes de cada *Sprint*, para cumplir con los procesos SCRUM destinados a la asignación de Historias de Usuario, identificación de tareas, estimación de tareas y creación del *Sprint Backlog*. Suele durar 8 horas si el *Sprint* está planificado para extenderse durante un mes.

- **Reunión de Revisión de *Sprint***: Esta reunión cumple con el fin de validar los resultados del *Sprint*, y en ella se presentan los entregables planificados para dicho *Sprint* al *Product Owner*. El *Product Owner* realiza la revisión del incremento de producto, verifica si cumple con los criterios de aceptación deseados y acepta o rechaza las historias de usuario completadas.

- **Reunión de Retrospectiva de *Sprint*:** Durante esta reunión, el *Scrum Team* revisa y reflexiona sobre las actividades del *Sprint* previo: Procesos seguidos, herramientas empleadas, mecanismos de colaboración y comunicación, y otros aspectos relevantes al proyecto. El equipo discute cuales actividades resultaron exitosas en el *Sprint* anterior y cuáles no, para realizar las mejoras correspondientes en los *Sprints* siguientes.

3.1.1.4. Artefactos de SCRUM

En SCRUM se utilizan diversos artefactos que permiten documentar el progreso del desarrollo de un producto y sirven para comunicar efectivamente a todo el equipo de trabajo las tareas a realizar durante el proceso de creación del producto y durante el progreso de cada *Sprint* definido.

- ***Product Backlog:*** Es una lista priorizada de los requerimientos del negocio y describe el proyecto en forma de épicas, que son historias de usuario de alto nivel. Se basa en tres factores principales: valor (el producto debe ofrecer el mayor valor), riesgo (mientras más incertidumbre exista, más riesgoso es el proyecto) y dependencias (siempre existe dependencias entre las historias de usuario). Esta lista es desarrollada y refinada contínuamente por el Product Owner, quien se encarga de verificar que las prioridades en las historias de usuario se mantengan o se modifiquen de acuerdo al cumplimiento de requerimientos durante el proceso de desarrollo.
- ***Sprint Backlog:*** Es la lista de tareas que llevará a cabo el SCRUM Team en el Sprint a iniciar. Se suele representar en un tablero de tareas que proporciona un recordatorio visual constante del estado de las historias de usuario a contemplar en el sprint. De igual manera, se incorpora cualquier riesgo asociado a las tareas que componen el Sprint Backlog y cualquier tarea que involucre la mitigación de dichos riesgos debe formar parte de este Backlog. Una vez el SCRUM Team finaliza y se compromete a las tareas del Sprint Backlog, no se deben agregar nuevas historias de usuario. Si durante un Sprint surgen nuevos requerimientos, estos serán agregados al Product Backlog e incluidos en un futuro Sprint.
- ***Incrementos*:** Los incrementos son los entregables que cumplen con los criterios de aceptación de las historias de usuario, definidos por el cliente y el Product Owner. Por definición, estos incrementos son potencialmente enviables, es decir, es posible enviar cada incremento, que debe incorporar también las funcionalidades de los incrementos anteriores, al cliente, que a su vez decidirá el momento en el que los entregables pasan a producción [16].

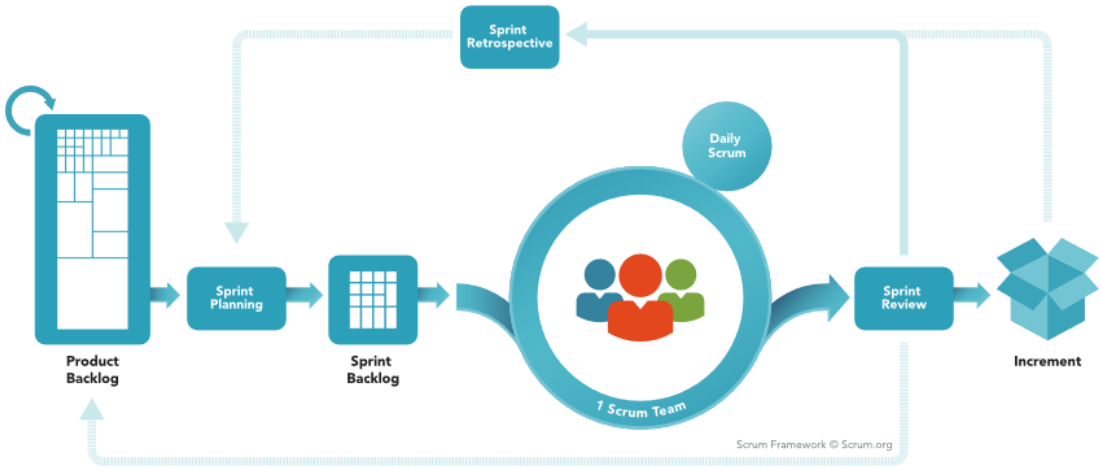

Figura 3.1.1.4.1: Estructura completa de un Sprint de Scrum
Fuente: Scrum.org

3.2. Modificaciones al entorno de trabajo SCRUM

Para el desarrollo del presente Mecanismo de Pagos en Criptomonedas con Tecnología RFID, se realizó una adaptación del entorno de trabajo SCRUM para un equipo de trabajo sumamente pequeño. Las modificaciones se listan a continuación:

- **Roles**: El rol de Cliente y *Product Owner* está ejercido por el tutor principal del presente trabajo, en tanto que el *SCRUM Team* estará compuesto por un solo desarrollador, quien elabora este Trabajo Especial de Grado.
- **Artefactos**: Los artefactos a generar son los mismos artefactos base requeridos por SCRUM, es decir, un *Product Backlog*, en el que se describen las funcionalidades necesarias para el producto final, una serie de *Sprint Backlogs* por Sprint, donde se describen una serie de tareas para completar cada una de las funcionalidades de acuerdo a su distribución por Sprint y los entregables generados al final de cada Sprint, que contarán con su debida documentación y código.
- ***Time Boxes***: La duración de cada Sprint se estima a 2 semanas aproximadamente para el desarrollo de cada una de las funcionalidades descritas en el Product Backlog y desglosadas en cada Sprint Backlog. Finalmente, la planificación queda estimada para el *Product Backlog* como se describe en la tabla 3.2.1 a continuación.

Sprint	**Tarea**	**Duración**
Sprint 1	Levantamiento de requerimientos	1 semana
	Definir el modelo de datos a utilizar	
Sprint 2	Desarrollo del Servidor de Autenticación en base a un modelo propio de datos (API Interna – Aplicación Web)	2 semanas
	Creación de la Base de Datos, Interacción del Servidor de Autenticación con la Base de Datos (API)	
Sprint 3	Desarrollo de las interfaces de interacción entre el Servidor de Autenticación y el Motor de Pagos (Token Authentication)	2 semanas
	Desarrollo de la aplicación del Motor de Pagos	
Sprint 4	Desarrollo del API Interna del motor de Pagos y su interacción con la librería para la creación de	2 semanas

	transacciones, registro de transacciones en *blockchain* y Documentación del API	
Sprint 5	Integración de la plataforma completa	1 semana
	Pruebas de integración - APIs	
	Pruebas funcionales – Transacciones (Dash Mainnet)	

Tabla 3.2.1: Planificación Scrum estimada para el proyecto

3.3. Metodología de Versionado

Para el control de versiones de código, se utilizó el esquema *A Successful Git Branching Model* [17], que consiste en la estructuración de un repositorio versionado con Git en ciertas ramas fundamentales. Este modelo permite crear un ambiente de desarrollo en la que el repositorio principal o *master*, solo contenga porciones del producto final que estén listas para su funcionamiento en producción, en tanto que el desarrollo de características se hace sobre otras ramas.

En primera instancia, suelen existir dos ramas fundamentales para este esquema: *master*, que ya fue mencionada previamente y contiene las versiones de código estable que pueden pasar a producción, y *develop* o *development*, donde se encuentra el código en desarrollo y que ha de ser probado antes de su incorporación a la rama maestra. De acuerdo con Driessen, esta rama suele ser considerada la rama de integración. También es posible crear ramas adicionales como las ramas de *feature* o característica, que permiten el desarrollo de características individuales antes de ser incorporadas a la rama *develop*.

Una vez las funcionalidades en *develop* alcanzan el nivel de estabilidad necesario, se integran a la rama *master*. De ser necesario, se pueden crear ramas para realizar correcciones de errores (*hotfix*), que deben ser integradas al finalizar las correcciones tanto a *master* como a *develop.*

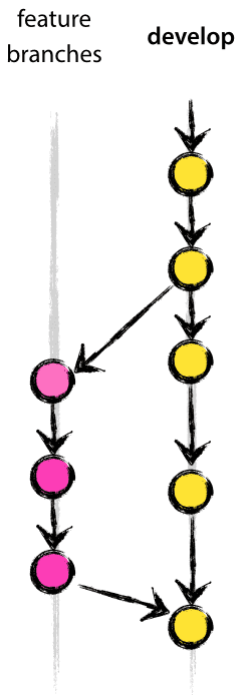

Figura 3.3.1: Ejemplo gráfico del funcionamiento de las ramas feature y develop
Fuente: https://nvie.com/posts/a-successful-git-branching-model/

Para este Trabajo Especial de Grado, se utilizaron 3 tipos de ramas fundamentales, *master* para el código estable, *develop* para el desarrollo constante y corrección de errores, y ramas *feature*, cada una con el nombre de la característica a desarrollar, utilizando la siguiente notación: feature-<SP_número de sprint_nombre del feature>, y que serán derivadas de la rama *develop*.

3.4. Herramientas de trabajo

En esta sección se describen de forma conceptual las diversas herramientas, tanto de hardware como de software que fueron utilizadas para el desarrollo de este trabajo.

3.4.1. Hardware para el Servidor de Autenticación y Motor de Pagos

Para la implementación del Servidor de Pagos, que se encuentra conformado por el Servidor de Autenticación y el Motor de Pagos, se utilizó para el desarrollo un equipo con las siguientes características:

- Procesador Intel® Core i5 7200U Dual Core - 2.5GHz
- 8GB Memoria RAM

- NVIDIA GeForce MX150 – 2GB VRAM

Para las pruebas, realizadas en el Laboratorio de Comunicaciones y Redes (LACORE), se utilizó un equipo con las siguientes características:

- Procesador Intel® Core i7 3770 Quad Core - 3.4GHz
- 8GB Memoria RAM

3.4.2. Python Django

Python es un lenguaje de programación con estructuras de datos de alto nivel, eficientes, y un enfoque simple a la programación orientada a objetos. Python posee tipado dinámico de datos y se caracteriza por ser un lenguaje de programación interpretado, por lo que es altamente utilizado para scripting y desarrollo de aplicaciones, tanto web como de escritorio en múltiples plataformas.

Python cuenta con su propio intérprete, que puede ser accedido a través de una interfaz de línea de comandos en el sistema operativo en el que se encuentre instalado. Además, cuenta con una extensa colección de paquetes y bibliotecas compatibles con diversas aplicaciones y plataformas, así como con otros paquetes y módulos desarrollados por terceros, por lo que puede servir para la personalización de aplicaciones ya establecidas [18].

Por otra parte, Django es un framework de desarrollo web de alto nivel que se caracteriza por promover el desarrollo rápido de aplicaciones web basadas en Python, con diferentes paquetes y bibliotecas preinstaladas que permiten la construcción de una aplicación web de forma rápida, completa y segura, bajo una variante del patrón Modelo-Vista-Controlador denominado Modelo-Plantilla-Vista (*Model-Template-View*) en donde los *templates* o plantillas se comportan exactamente igual que las vistas en MVC, y las *views* cumplirían el rol de los controladores.

Django maneja por defecto diversos *back-ends* que facilitan la incorporación de funcionalidades como manejo de sesiones, autenticación, prevención contra ataques de *Cross-Site Scripting* (XSS) y *Cross-Site Request Forgery* (CSRF) integrados con el lenguaje Python que a su vez permiten el empleo de bibliotecas externas construidas en este lenguaje para complementar las funcionalidades de una aplicación web [19].

Django suele emplear un servidor web, en principio de manera local, al que el navegador realiza solicitudes y espera por respuestas que pasan desde el núcleo de la aplicación web, a través de los diversos middlewares que comprende el framework, para finalmente interactuar con el servidor web a través de una interfaz WSGI (*Web Server Gateway Interface*), en la que se pueden ejecutar aplicaciones en Python y se encarga de realizar la interpretación de los requerimientos provenientes del servidor web hacia el *back-end* de la aplicación web. Por su alta compatibilidad con Python, es posible ejecutar paquetes o scripts de Python dentro de Django, en especial dentro de las *views*, como

fuese el intérprete nativo, dado el rol de controladores que poseen las *views* de Django y que básicamente son el soporte programático de toda la aplicación web (Figura 3.4.2.1).

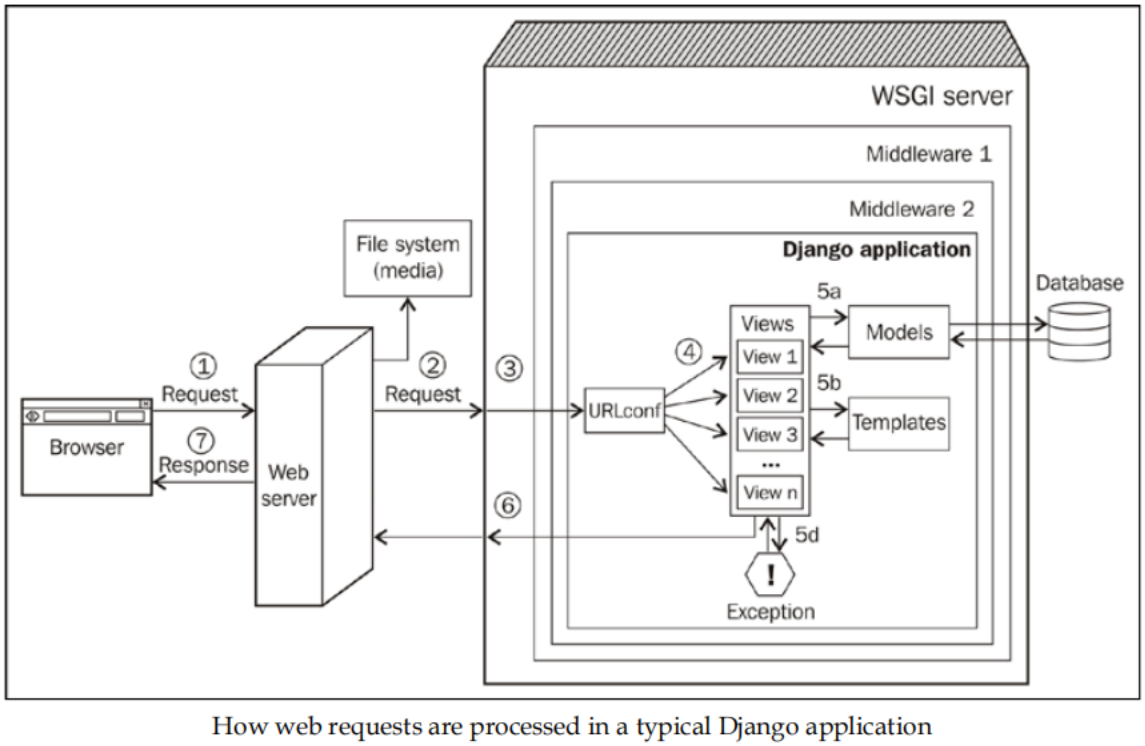

Figura 3.4.2.1: Estructura de una aplicación Django y cómo se procesan las solicitudes web en Django
Fuente: [19]

Como framework, Django también provee su propio ORM (*Object-Relational Mapping*) que establece un sistema de comunicación eficiente con la base de datos, agnóstico al sistema manejador de base de datos que se utilice en la aplicación y empleando un lenguaje orientado a objetos, completamente compatible con las funcionalidades que Python provee al framework. Estos objetos, llamados modelos, pueden ser accedidos por las *views*, para el manejo de las funcionalidades generales de la aplicación y las respuestas que la misma retornará al servidor web a través de rutas (URLs) y páginas HTML establecidas en el navegador (plantillas o *templates*).

De igual forma, el sistema de enrutamiento de Django permite implementar el sistema como un *back-end* dedicado a una arquitectura orientada a servicios, específicamente servicios REST que permiten la ejecución de acciones desde y hacia el servidor web utilizando únicamente verbos o peticiones HTTP, y cuyo intercambio de información puede ser base para un *front-end* web, móvil o incluso de cualquier dispositivo que tenga la capacidad o compatibilidad necesaria para realizar peticiones HTTP, ya sea de forma nativa o incorporando librerías adicionales, como puede ser el caso de una placa Arduino o Raspberry Pi.

3.4.3. Django Admin

Django Admin es una interfaz de administración de usuarios personalizable que Django provee por defecto, a manera de un sistema de manejo de contenido o usuarios para una aplicación web. A partir de esta página es posible administrar cualquier objeto cuyo modelo de datos se encuentre registrado en el respectivo archivo *admin.py* relacionado a una aplicación Django. Al acceder al Django Admin, es posible visualizar todos los modelos registrados de todas las aplicaciones del proyecto general, y de igual forma es sencillo visualizar información, hacer operaciones básicas de Creación, Lectura, Actualización y Eliminación (*Create, Retrieve, Update, Delete,* CRUD) en los objetos que se encuentren en base de datos. Por lo general es usado en gran medida por los desarrolladores de un proyecto para la creación de los objetos necesarios, pero se puede personalizar lo suficiente a nivel de interfaz y permisología para ser accedido por un grupo de usuarios finales que tengan permisos de administración y así gestionar tanto el contenido desplegado en un proyecto o aplicación web completa desarrollada en Django, como los usuarios y perfiles que hagan uso de ésta.

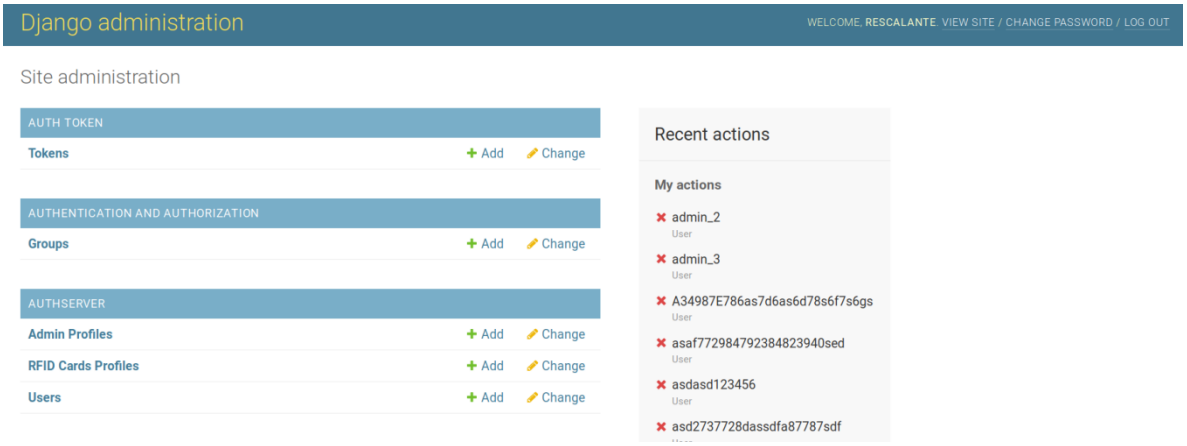

Figura 3.4.3.1: Apariencia inicial de la interfaz Django Admin
Fuente: Elaboración propia - 2019

3.4.4. Django Rest Framework

Para la creación de API (*Application Programming Interface*) bajo una estructura REST, existe una biblioteca o paquete Python altamente compatible con Django bajo el nombre Django Rest Framework. Este paquete toma el esquema de programación característico de Django para la creación de clases y funciones que son específicamente diseñadas para las operaciones clásicas utilizadas en el esquema REST de servicios web, es decir, realizar operaciones utilizando diversos verbos HTTP para comunicarse y transmitir datos a un servidor web, local o en la nube, o realizar operaciones CRUD sobre el esquema de datos de una aplicación web [20].

Al igual que Django, Django Rest Framework incluye por defecto una cantidad importante de *plugins*, *middlewares*, validadores y esquemas de autenticación que conforman una base sólida para crear una aplicación web que funcione perfectamente como un proveedor de servicios web para otro tipo de aplicaciones (web, móviles, o de escritorio) que tengan compatibilidad con peticiones HTTP.

Django Rest Framework permite la implementación de serializadores, elementos que permiten transformar la información a utilizar por los servicios en un formato JSON adecuado para su despliegue y utilización en diversas plataformas, ya que Python y Django suelen manejar la información por defecto en objetos denominados *Querysets* que usualmente no son compatibles con diversas plataformas que utilicen esquemas HTTP.

Al igual que Django, Django Rest Framework cuenta con una documentación extensa y al ser altamente basado en clases, permite personalización en sus funciones básicas a través de la herencia de clases y la sobreescritura de los métodos en las clases, todas ventajas inherentes a la aplicación del lenguaje Python y su paradigma orientado a objetos.

3.4.5. Bitcoinlib

Bitcoinlib es una biblioteca Python creada por Lennart Jongeneel [21], que permite realizar operaciones en diversas criptomonedas como Bitcoin, Litecoin o Dash, utilizando una serie de primitivas relacionadas a creación de billeteras, claves, direcciones y manejo de transacciones de una determinada criptomoneda. Todas las funcionalidades provistas por Bitcoinlib pueden ser accedidas desde el intérprete de Python o utilizando una interfaz de línea de comando. A nivel de persistencia de datos, utiliza SQLAlchemy como ORM y una base de datos SQLite3 para los modelos relacionados con Billeteras, Claves, Transacciones, y proveedores de servicio.

Bitcoinlib implementa como base las Billeteras Jerárquicas Deterministas (*HDWallets*), permitiendo la creación de billeteras desde cero, importación desde una clave privada o incluso desde una *passphrase* que, tal como fue referenciado en el apartado de HDWallets en la sección 2.1.1.9 del presente documento, permiten la reconstrucción de la billetera a partir de dicho *passphrase* semilla. De igual modo, la forma en la que se manejan las transacciones bajo Bitcoinlib es utilizando *Unspent Transaction Outputs* (UTXOs), y la biblioteca define funciones y clases que permiten la consulta y actualización de las UTXOs en base de datos, ya sea desde un proveedor de servicios en línea (*Block Explorers*) o incluso desde un nodo completo de una determinada criptomoneda, si este último se encuentra correctamente instalado y configurado en un equipo.

Si bien la mayoría de las criptomonedas soportadas por la biblioteca suelen ser forks de Bitcoin y por ende, trabajan bajo estándares similares, de donde proviene la facilidad de implementar *HDWallets* como base, la versatilidad de esta biblioteca radica en su estructura basada en clases, por lo que si se desea crear nuevas funcionalidades para diferentes criptomonedas que no se encuentren soportadas, se pueden añadir clases especializadas con las especificaciones dadas por cada criptomoneda, ya sean un fork de Bitcoin o no. La estructura principal de módulos disponibles en Bitcoinlib se puede apreciar en la figura 3.4.5.1

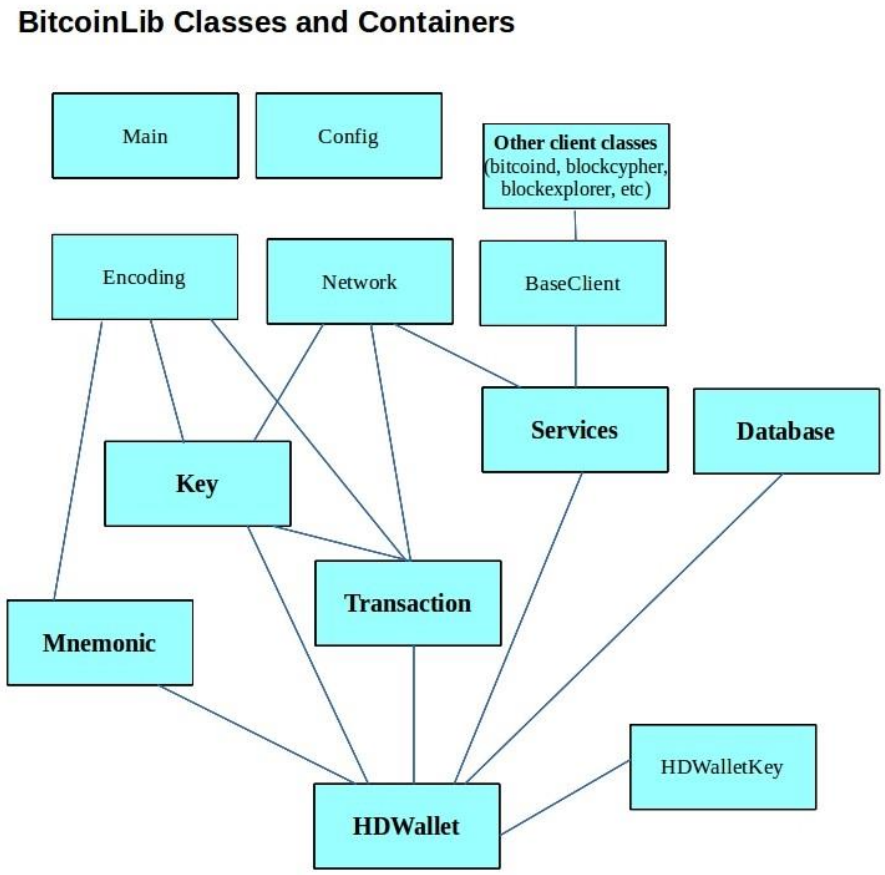

Figura 3.4.5.1: Clases principales de Bitcoinlib
Fuente: [21]

4. DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN

En este capítulo se presenta la arquitectura de la solución propuesta y los detalles de implementación pertinentes a cada módulo a desarrollar siguiendo el esquema de desarrollo especificado en el capítulo anterior.

4.1. Diseño de la solución

Para la aplicación que se engloba como Servidor de Pagos, se proponen dos módulos importantes. El primer módulo corresponde a un servidor de autenticación que permitirá a los usuarios poseedores de una tarjeta inteligente realizar transacciones en el sistema, a través de operaciones de autenticación que serán respaldadas por una base de datos con información de usuarios, y un segundo módulo, que funge como motor de pagos o transacciones, realizando verificaciones de saldo disponible en las billeteras asociadas a las tarjetas y la construcción y envío de transacciones a la red.

La figura 4.1.1 ilustra con un diagrama de bloques la estructura general del mecanismo a implementar y su funcionamiento:

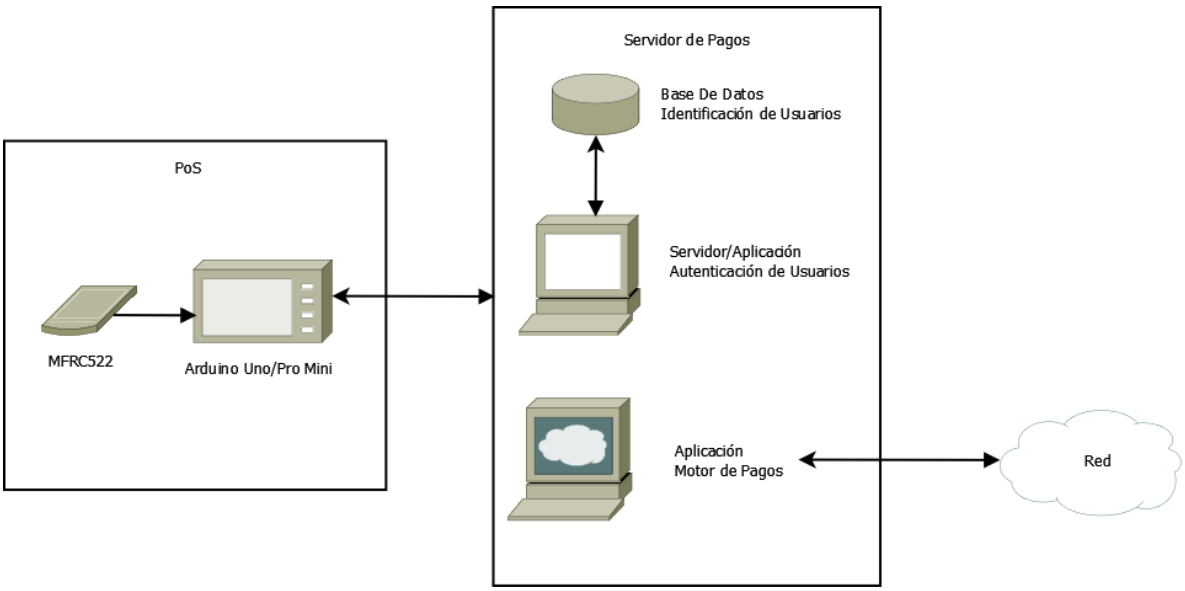

Figura 4.1.1: Diagrama de Bloques de la solución
Fuente: Elaboración propia - 2019

4.1.1. Diseño de Casos de Uso

Para establecer los casos de uso de la propuesta anterior, se definen en primera instancia dos actores del sistema, administrador y cliente. Los usuarios de tipo administrador se encargan registrar usuarios regulares en el servidor de autenticación, puntos de venta y gestionar el proceso de transacción e ingreso de datos al utilizar el sistema, en tanto que los usuarios de tipo cliente simplemente interactúan con el sistema de punto de venta para el ingreso de datos y con el módulo de administración para la visualización de sus datos.

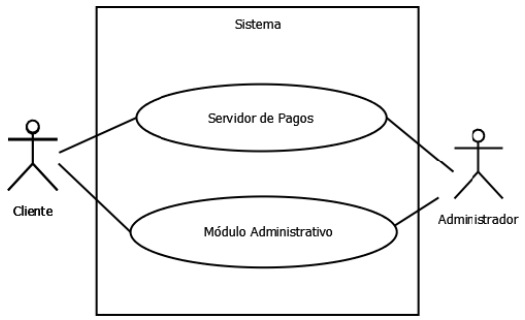

Figura 4.1.1.1: Casos de uso de nivel 0
Fuente: Elaboración Propia – 2019

En el caso del módulo de administración, Django provee una interfaz de administración completa y configurable para la gestión de usuarios a través de Django Admin, sin embargo, es posible crear una interfaz sencilla para el registro de ambos tipos de usuario, Luego, las funciones principales pueden verse ilustradas en la figura 4.1.1.2

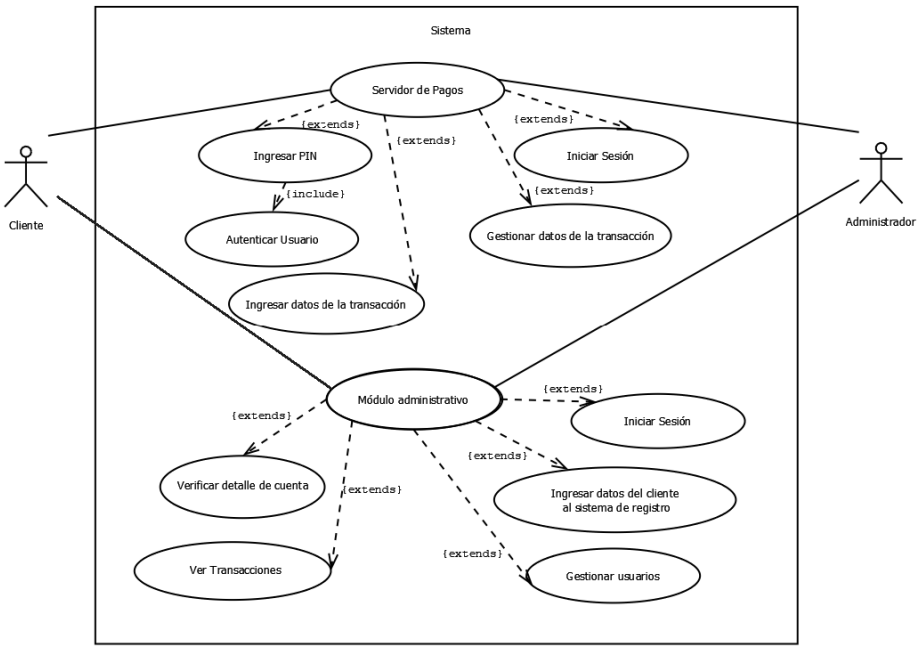

Figura 4.1.1.2: Casos de uso de nivel 1
Fuente: Elaboración Propia - 2019

4.2. Descripción de la implementación

Para la implementación de la arquitectura descrita en el diseño de la solución, es necesario tener en cuenta la implementación de dos APIs de uso interno alojadas en cada una de las aplicaciones, el servidor de autenticación y el motor de pagos, que servirán como comunicadores de peticiones HTTP desde el punto de venta hacía el servidor de autenticación y el motor de pagos, entre el servidor de autenticación y el motor de pagos para realizar consultas y desde el motor de pagos hacia la red para la realización y envío de transacciones a la red.

4.2.1. Levantamiento de Requerimientos

De acuerdo entonces con los casos de uso ilustrados y la descripción de la implementación a realizar, se cubren los siguientes requerimientos funcionales:

- La aplicación debe proveer una entrada adicional de datos de usuario, a través de un sistema adicional de gestión de usuarios, esto permite flexibilidad a la hora de realizar pruebas.
- El sistema de gestión de usuarios debe permitir visualizar los usuarios actuales del sistema y su información, tal como balance actual y dirección de depósito para añadir fondos a una determinada tarjeta.
- La aplicación debe proveer entrada de datos de usuario a través de APIs para su comunicación empleando llamadas o peticiones HTTP desde el dispositivo de hardware.
- El motor de pagos debe implementar el mismo esquema de autenticación utilizado en el servidor de autenticación, de forma que solo usuarios con tarjeta pertenecientes al sistema y peticiones realizadas desde puntos de venta registrados puedan realizar acciones sobre los *endpoints* del mismo.
- A través del mismo sistema de *endpoints* del motor de pagos, debe ser posible realizar la solicitud de un balance de una billetera (en fracciones de criptomoneda).
- Debe ser posible realizar una transacción una vez se ha recibido confirmación de que el balance disponible en la billetera es suficiente para realizar la transacción (en fracciones de criptomoneda).

Asimismo, la aplicación debe cumplir con ciertos requerimientos no funcionales

- El sistema de gestión de usuarios debe ser sencillo y fácil de entender, con una interfaz simple pero agradable a la vista.
- Los *endpoints* de la aplicación deben proporcionar mensajes informativos respecto a su estado y la acción que están realizando.
- La arquitectura de la aplicación y por ende de sus API internas debe ser modular y extensible.
- Disponibilidad de la plataforma.
- Seguridad en el almacenamiento de datos personales como contraseñas o PIN.

4.2.2. Diseño de modelos

Para el cumplimiento inicial de estos requerimientos, se diseñaron dos modelos base. Ambos modelos están basados en un modelo de usuario, configurable en Django a través del archivo de configuraciones del proyecto, *settings.py*.

Al ser basado en Python, Django provee las mismas funcionalidades para realizar programación orientada a objetos dentro del mismo framework, por lo que es posible crear nuevas clases que hereden de otras clases que Django provee para su funcionamiento, en casi cada aspecto de la aplicación, tanto a nivel de modelos como a nivel de *views*. En el caso de los modelos, es posible heredar de dos clases que Django posee por defecto, AbstractBaseUser, que es la definición a más bajo nivel de un usuario dentro de Django y AbstractUser, que ya posee una definición un tanto más básica pero completa, con campos que pueden ser aprovechados por cualquier modelo de usuario que se pueda instanciar en Django. Para este trabajo, se realizó una implementación de un modelo User, que hereda de AbstractUser todos sus campos.

De este modelo User, que se establece en el archivo de configuración del proyecto como el modelo base para la autenticación (AUTH_USER_MODEL), derivan dos modelos de usuario básicos. El primero es el modelo CardUser, que hereda los campos de nombre de usuario y contraseña del modelo User, para ser usados como serial de tarjeta y PIN, y además añade un par de campos adicionales, un UID, o identificador único para la tarjeta, y un campo Wallet, para almacenar el nombre de la billetera que será creada a través de Bitcoinlib y utilizada posteriormente en el motor de pagos.

Por otro lado, se tiene un modelo AdminProfile, que en esta fase inicial hereda de User para poder ser autenticado por el sistema tanto a nivel de API, como a nivel del Sistema de Gestión de usuarios. En el caso del AdminProfile, se especifica si el usuario de este tipo es Staff para que tenga ciertos roles dentro del Django Admin.

Los modelos quedan a nivel de base de datos reflejados con las siguientes tablas:

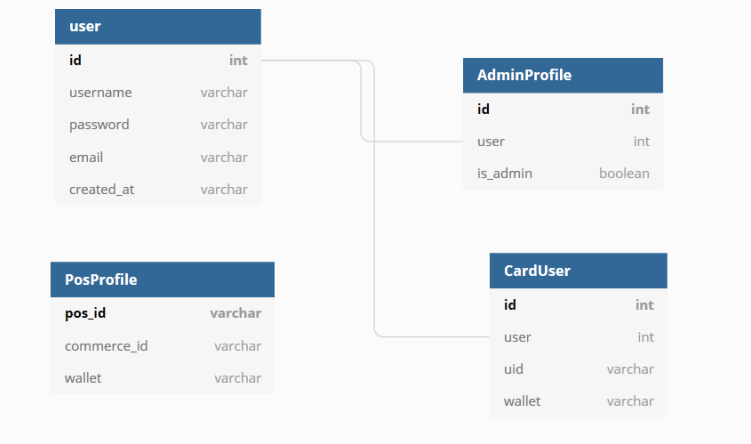

Figura 4.2.2.1: Modelo de la base de datos de la aplicación *authserver*
Fuente: Elaboración propia - 2019

Por el lado del motor de pagos se registra un único modelo Transaction con el fin de registrar transacciones exitosas, de forma que es posible desplegar un histórico de transacciones por usuario en el sistema. El modelo se refleja en la siguiente tabla:

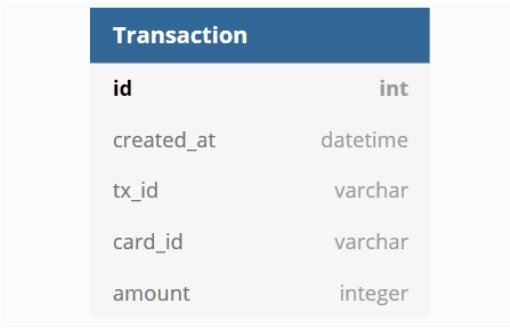

Figura 4.2.2.2: Modelo de datos de la aplicación *paymentengine*
Fuente: Elaboración propia - 2019

4.2.3. Desarrollo del Servidor de Autenticación

Como objetivo del Sprint 2 y parte del Sprint 3 del proyecto se planeó el desarrollo de un servidor de autenticación que realice la verificación de usuarios regulares en el sistema. Para ello, se creó una aplicación en el proyecto Django llamada '*authserver*' en la cual se crearon los modelos previamente mencionados, pues son base para los esquemas de autenticación y la primera API interna, destinada para el servidor de autenticación y desarrollada con Django Rest Framework.

Como parte del servidor de autenticación, y su definición de URLs de acceso, se establece una pequeña interfaz web que permite el inicio de sesión para usuarios, y el registro de ambos tipos de usuarios. Es posible interrelacionar ambas funcionalidades implementando el esquema de autenticación TokenAuthentication que provee Django Rest Framework. Este esquema permite registrar un token único por cada usuario que lo identifica en el sistema, y dicho token puede ser utilizado para la autenticación de otras funciones dentro del proyecto general u otras aplicaciones que sean dependientes de los modelos definidos en la aplicación *authserver*.

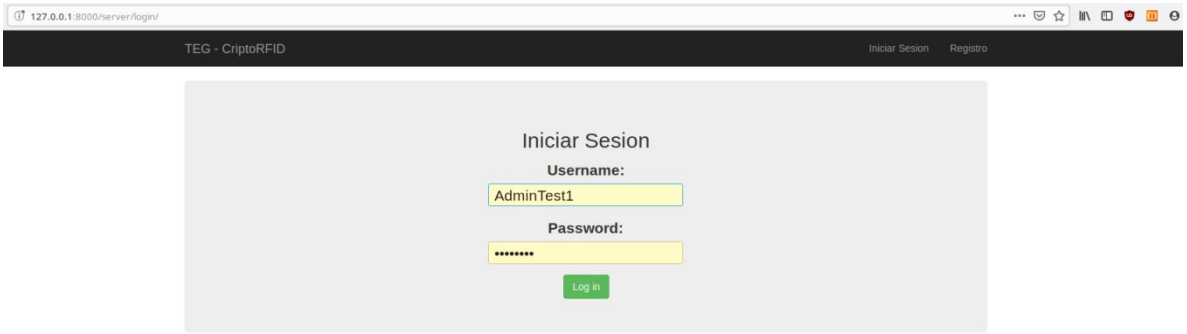

Figura 4.2.3.1: Interfaz de inicio de sesión
Fuente: Elaboración propia - 2019

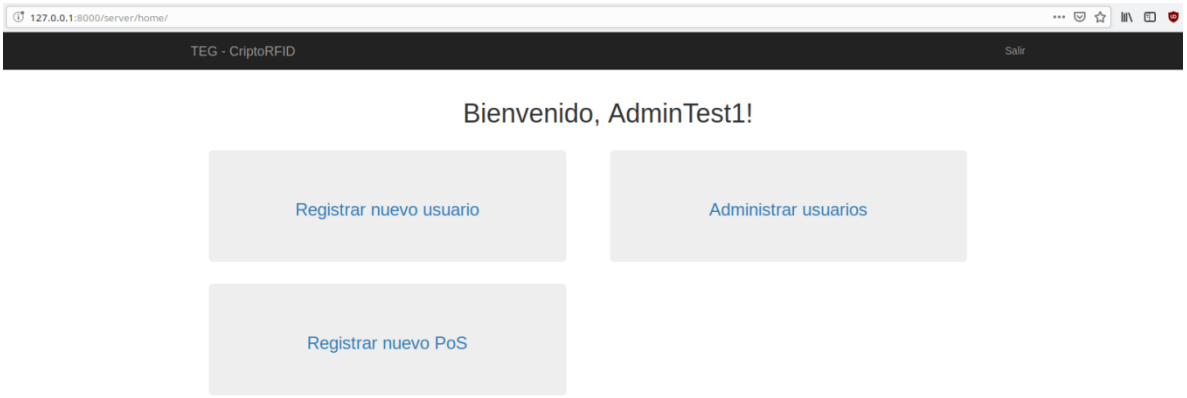

Figura 4.2.3.2: Interfaz de administración
Fuente: Elaboración propia -2019

Para la implementación de la autenticación por tokens, es necesario realizar una configuración inicial. En primer lugar, se debe registrar en el archivo de configuraciones general del proyecto, un apartado relativo a las clases de autenticación de Rest Framework, como se aprecia en la siguiente figura:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ],
```

Figura 4.2.3.3: Configuración de backends de autenticación para Django Rest Framework
Fuente: Elaboración propia - 2019

BasicAuthentication y TokenAuthentication se definen como las dos clases principales que se utilizarán para la gestión de autenticación dentro del proyecto. Una vez se realiza esta configuración, es necesario aplicar el comando `python manage.py migrate` desde la terminal para realizar una migración que genere una nueva tabla en la base de datos, destinada a guardar los tokens de cada usuario.

Al realizar esta última migración, es posible registrar una función inicial, encargada de gestionar los tokens creados desde Django Rest Framework y solicitar las credenciales correctas a través del API interna. Esta función, o *view* como se suele conocer en el patrón MTV manejado por Django, se denomina *obtain-auth-token* y es posible importarla directamente en el archivo de enrutamiento de la aplicación, es decir, el archivo *urls.py* de la aplicación *authserver:*

```python
from .views import *
from rest_framework.authtoken.views import obtain_auth_token

app_name = 'authserver'

urlpatterns = [
    path('v1/auth', obtain_auth_token, name='token_auth'),
```

Figura 4.2.3.4: Configuración de la ruta obtain-auth-token
Fuente: Elaboración propia – 2019

Al configurar esta URL, es posible obtener el token asociado a un grupo de credenciales otorgadas a esta ruta utilizando el método POST. Este primer paso de identificación es fundamental, ya que se establece un esquema de autenticación extensible a todo el proyecto y sus API internas.

Para el caso particular de este proyecto, se realizó una extensión del esquema de autenticación, añadiendo una verificación que recibe el identificador de un punto de venta, previamente registrado en el sistema y que dará paso a la verificación de identificador de tarjeta y PIN únicamente si la verificación del punto de venta es exitosa.

Es necesario acotar que para el empleo de este tipo de autenticación, en particular con Django Rest Framework, se requiere tener bien establecido el modelo de usuario de Django, ya que Django Rest Framework basa enteramente el esquema de autenticación en dicho modelo. Por ende, es necesario que los usuarios sean creados de dos formas en el sistema: empleando una vista de API (*APIView*) que tome las credenciales y datos del usuario base y los envíe a la aplicación usando el método POST, o empleando un formulario de creación de usuarios que utilice el modelo de datos directamente desde Django. Ambas funciones fueron creadas para esta aplicación. En primer lugar, la APIView que realiza la creación de usuarios, bajo la URL '*v1/signup*' toma los datos de usuario provenientes del request realizado para armar una estructura base:

```python
# API VIEWS
class CreateUser(APIView):
    """
    View to create a single user via post
    * Requires token authentication.
    """
    authentication_classes = (authentication.TokenAuthentication,)
    renderer_classes = [HTMLFormRenderer]

    def post(self, request):
        user = {
            'username': request.data.get('username'),
            'password': request.data.get('password'),
            'is_staff': request.data.get('is_staff')
        }
        serializer = UserSerializer(data=user)
        if serializer.is_valid():
            user_saved = serializer.save()
            return Response({"success": "User'{}' created successfully".format(user_saved.username)}, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figura 4.2.3.5: Vista basada en clases para creación de usuario con Django Rest Framework
Fuente: Elaboración propia – 2019

Hay que hacer notar el uso de un serializador, que permite realizar el guardado de un nuevo usuario con el formato correcto. Cada aplicación puede tener un serializador, que se encarga de procesar directamente los datos o de simplemente darles el formato adecuado para ser mostrados en la respuesta que el API otorgará al lado consumidor del servicio. En este caso particular, el serializador realiza el guardado del usuario, ya sea que se cree por primera vez o sea actualizado en algún momento utilizando una función nueva del API:

```python
from rest_framework import serializers

#Django User Serializer
class UserSerializer(serializers.Serializer):
    """
    Simple Serializer based on the Django User model
    * Can be modified to serialize more data or use another model
    """
    username = serializers.CharField(max_length=256)
    password = serializers.CharField(max_length=20)
    is_staff = serializers.BooleanField(default=False)

    def create(self, validated_data):
        """
        Creates an instance of a User
        """
        return get_user_model().objects.create_user(**validated_data)

    def update(self, instance, validated_data):
        instance.username = validated_data.get('username', instance.username)
        instance.password = validated_data.get('password', make_password(instance.password))
        instance.is_staff = validated_data.get('is_staff', instance.is_staff)

        instance.save()
        return instance
```

Figura 4.2.3.6: Serializador para el modelo de usuario de Django Rest Framework
Fuente: Elaboración propia – 2019

También se define una función de creación de usuarios utilizando un formulario de Django orientado al modelo de usuario. Django provee un tipo de formularios asociados a modelos llamados ModelForms. A partir de ellos es posible crear *views* asociadas a formularios que sean destinados a crear objetos en base de datos de forma muy sencilla:

```python
from .models import User, CardUser

class CustomUserCreationForm1(forms.ModelForm):

    class Meta:
        model = get_user_model()
        fields = ['username', 'password', 'is_staff']
        widgets = {
            "password": forms.PasswordInput(),
            "is_staff": forms.CheckboxInput()
        }
```

Figura 4.2.3.7: Formulario de creación de usuarios usando ModelForm de Django
Fuente: Elaboración propia – 2019

La importancia de realizar el registro de usuarios con ambas funciones radica en el proceso que utilizan tanto Django como Django Rest Framework para almacenar el usuario en base de datos. Django provee diversos *back-ends* para el *hashing* de una contraseña, utilizando funciones de

derivación de contraseñas como PBKDF2 y añadiéndoles una sal basada en SHA256. Es con una contraseña encriptada de esta manera que Django Rest Framework realiza la generación de tokens en base de datos, de forma que si una contraseña o PIN es almacenado en base de datos como texto plano, no obtendrá un token adecuado para su utilización en el sistema.

4.2.4. Rutas

En general, las rutas utilizadas para el servidor de autenticación y el acceso a sus funcionalidades, ya sea a nivel de API o en el sistema de gestión, se definen en el archivo *urls.py* de la aplicación *authserver* como se ilustra en la figura 4.2.4.1:

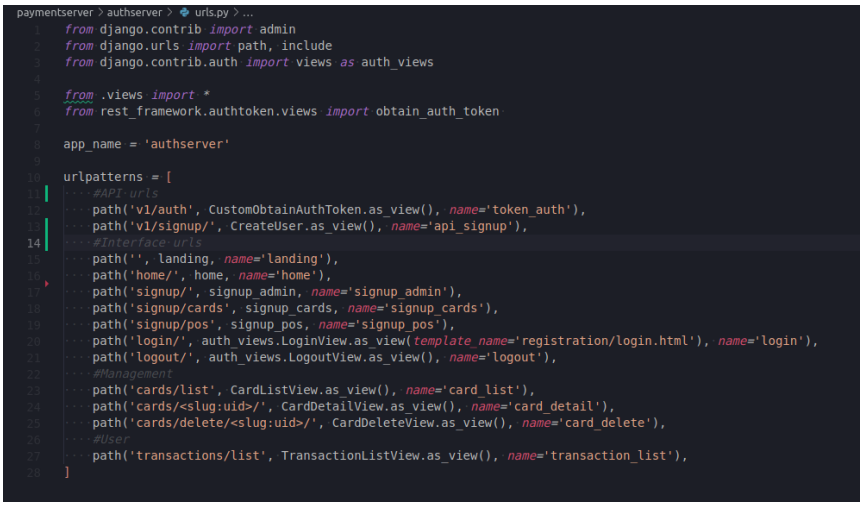

Figura 4.2.4.1: Especificación de URLs para la aplicación *authserver*
Fuente: Elaboración propia – 2019

En la figura se puede observar la distribución de URLs, iniciando desde las URLs principal de autenticación y registro, y un conjunto de URLs para la interfaz de usuario, que sirven de entrada a las acciones del sistema, dos URLs asociadas a dos funciones separadas: '*signup*' para registrar usuarios de tipo Admin al sistema y '*signup/cards*' para el registro de usuarios-tarjeta al sistema. Luego, en la parte inferior, se especifican tres URLs correspondientes a la interfaz de listado de tarjetas, detalle de una tarjeta y eliminación de una tarjeta del sistema:

Figura 4.2.4.2: Listado de usuarios actuales de tarjetas en sistema
Fuente: Elaboración propia – 2019

Figura 4.2.4.3: Detalle de un usuario y su información básica de cuenta
Fuente: Elaboración propia – 2019

En el detalle de usuarios se aprecia claramente información relevante, más no sensitiva, sobre la tarjeta asociada, tal como el balance y una dirección a la que se le puede hacer transferencia directa a la tarjeta para añadirle fondos y que la misma sea utilizada en el sistema.

Al momento de activar la opción de "Eliminar" en el listado, se despliega una página de confirmación para asegurar la eliminación del usuario del sistema.

Figura 4.2.4.4: Confirmación de eliminación de un usuario
Fuente: Elaboración propia – 2019

Estas acciones se logran con vistas específicas que posee Django para facilitar el despliegue de estas interfaces y cuyas operaciones sobre la base de datos suelen ser más eficientes. En el caso de la operación de eliminar un usuario, es una buena práctica realizar una desactivación del usuario antes que borrarlo de golpe, más porque se realiza el manejo de una billetera asociada. Por este motivo, se realizó una sobreescritura de la *view* de Borrado que Django provee para que en vez de eliminar al usuario directamente de la base de datos primero lo desactive. De esta forma, sólo un superusuario puede realizar la reactivación del usuario desactivado a través de la interfaz Django Admin.

Asimismo, estas operaciones de lectura y desactivación son compatibles con el esquema de autenticación utilizado por Django Rest Framework.

Del lado del usuario regular, se posee la misma interfaz de inicio de sesión, en la que se ingresa actualmente utilizando el serial de tarjeta y el PIN asociado a la tarjeta. Una vez autenticado en el sistema el usuario es capaz de visualizar un par de acciones, la primera redirecciona a su detalle de cuenta, en una interfaz similar al detalle de cuenta desde el administrador, y la segunda redirecciona a un historial de transacciones en el cual se muestra las transacciones exitosas que se han realizado en el sistema, con fecha y hora, identificador de transacción, que a su vez permite verificar el estado de confirmación de la transacción en la red si se utiliza un explorador de bloques en línea, y el monto transado.

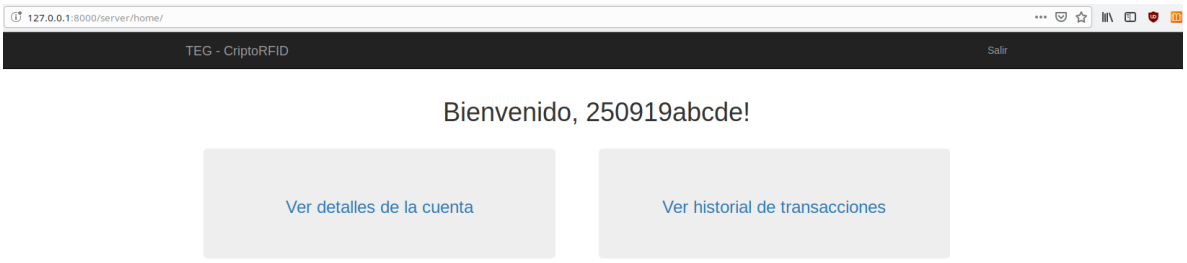

Figura 4.2.4.5: Interfaz de un usuario regular
Fuente: Elaboración propia – 2019

Figura 4.2.4.6: Listado de Transacciones
Fuente: Elaboración propia – 2019

4.2.5. Desarrollo del Motor de Pagos e Interacción con la *Blockchain*

Continuando el desarrollo iniciado en el Sprint 3 y siguiendo con el Sprint 4, se planteó el desarrollo del motor de pagos, generando una aplicación similar a *authserver*, denominada '*paymentengine*'. La diferencia fundamental es que esta aplicación contiene solo operaciones internas de API para el procesamiento simple de la información proveniente de las tarjetas y el manejo de la billetera asignada al momento de crear el CardUser desde la aplicación *authserver*.

En esta aplicación se hace un uso extensivo de la librería bitcoinlib, en específico de sus dos clases wallets y transactions.

Bitcoinlib.wallets se encarga de todo el manejo de billeteras, desde su creación en su propia base de datos (implementada por bitcoinlib como una base de datos SQLite3), manejo de claves, generación de direcciones y gestión de direcciones bajo el esquema de *HDWallets*. Por otro lado, bitcoinlib.transactions permite la creación de transacciones con diversos métodos que actúan bajo los esquemas de creación de transacciones usuales en criptomonedas, permitiendo la creación de transacciones a partir de entradas específicas, creación de transacciones crudas o implementando métodos adicionales que permitan simplificar estas operaciones según los requerimientos.

En este caso, se empleó la función '*send_to'*, que trabaja especificando únicamente una dirección de envío y un monto, expresado en fracciones de criptomonedas. Esta operación realiza el chequeo de existencia de billeteras y UTXOs por igual al generar la transacción y utiliza el servicio por defecto que se encuentre activo y previamente configurado en el archivo de configuración de bitcoinlib *providers.json*, ya sea un explorador online o el full nodo de una criptomoneda, para crear una transacción cruda a partir de esta información.

Al utilizar la criptomoneda Dash como base para crear transacciones, la fracción a utilizar para el envío de las mismas es el Duff, es decir, 0,0000001 DASH. Estas transacciones utilizan como entrada las UTXOs disponibles en la billetera con la que se llama la función 'send_to' al momento de enviar la transacción.

4.2.6. Rutas

El esquema de rutas en esta aplicación resulta mucho más sencillo, especificando únicamente dos rutas, de las cuales una cumple el objetivo simple de consulta de datos:

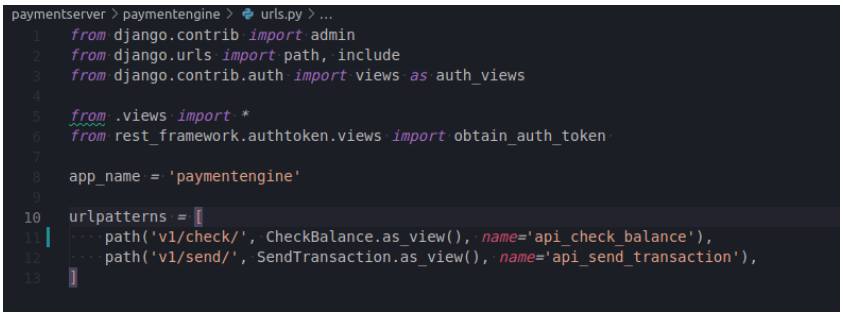

Figura 4.2.5.1: Endpoints de la aplicación *paymentengine*
Fuente: Elaboración propia – 2019

Con el endpoint '*engine/v1/check*/' se puede hacer el chequeo del balance actual de una billetera. Esta consulta se realiza actualizando las UTXOs disponibles de la billetera ya sea a través de un proveedor de servicio online o consultando directamente al nodo completo de la criptomoneda instalado en el mismo equipo en el que se aloja el servidor de pagos. En particular, funciona para la consulta del

saldo del punto de venta y su billetera asociada al comercio en el que se encuentre, como se ilustra en la figura 4.2.5.2:

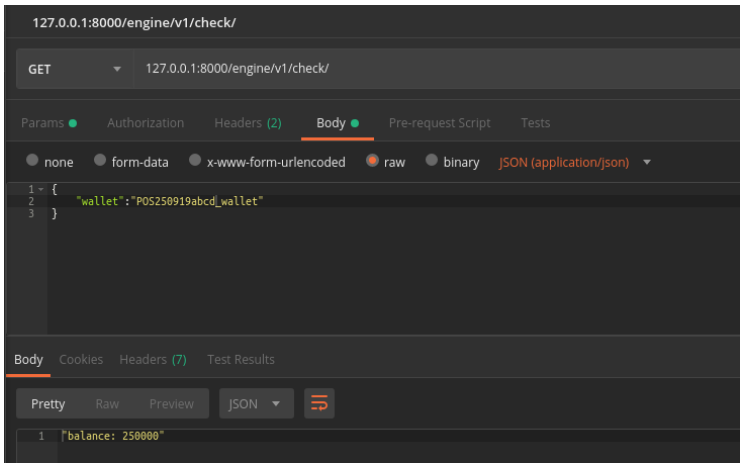

Figura 4.2.5.2: Uso del endpoint 'engine/v1/check'
Fuente: Elaboración propia – 2019

Por otro lado '*engine/v1/send/*' permite el envío de una transacción especificando tres parámetros fundamentales, que deben ser incluidos en una llamada POST desde el dispositivo que funcionará como punto de venta: identificador o serial de tarjeta, identificador o serial del punto de venta y el monto. Cabe acotar que los dos métodos especificados requieren el token de autenticación generado por el servidor de autenticación, en una cabecera Authorization que se adjunta a la petición acompañada del token generado, o de lo contrario generarán un error HTTP 401 (*Unauthorized* – No Autorizado) lo que asegura que estos datos han sido verificados al realizar la llamada. Si la llamada es exitosa, el servidor envía una respuesta con el ID de la transacción realizada:

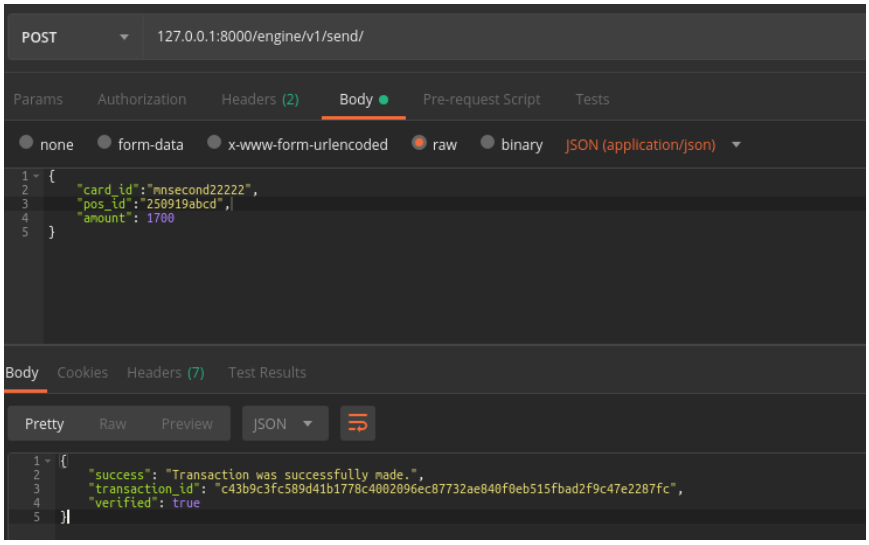

Figura 4.2.5.3: Estructura de una petición al endpoint 'engine/v1/send/' y sus parámetros
Fuente: Elaboración propia – 2019

Para realizar una transacción, se actualizan las UTXOs de la billetera asignada al card_id especificado y se realiza una consulta del balance actual de la billetera. En caso que el balance no sea suficiente, se informa en la respuesta de la llamada que el balance no es suficiente, indicando un status 406 para las peticiones no exitosas.

Bitcoinlib requiere la configuración de proveedores de servicio para poder realizar la emisión de transacciones a la blockchain y actualización de UTXOs. Existe la opción de configurar diversos exploradores de bloques online como proveedores de servicio, sin embargo, para la criptomoneda Dash, la mayoría de exploradores requieren el uso de una API KEY, solicitada bajo un esquema de pago, para prestar el servicio. Por ende, para garantizar la disponibilidad del sistema, el mismo depende de la instalación de un nodo completo de Dash Core para realizar la emisión de las transacciones y la consulta de las UTXOs de cada billetera directamente contra la copia actualizada de la *blockchain* de Dash almacenada en el sistema.

Finalmente, concluyendo el sprint 5, la documentación de las API, tanto de *Authserver* como *Paymentengine* fue realizada gracias a la biblioteca drf-yasg, recomendada por Django Rest Framework, que permite utilizar los recursos de Django Rest Framework como coreapi y openapi para la generación de una URL que lista todos los endpoints y su información asociada, que es descrita en los docstrings[18] de cada una de la APIView desarrolladas. Para el proyecto Paymentserver, la documentación se encuentra alojada en la dirección 'docs/'. De igual manera, en el Anexo 1.1 del presente documento, se encuentra la documentación detallada de los endpoints del proyecto.

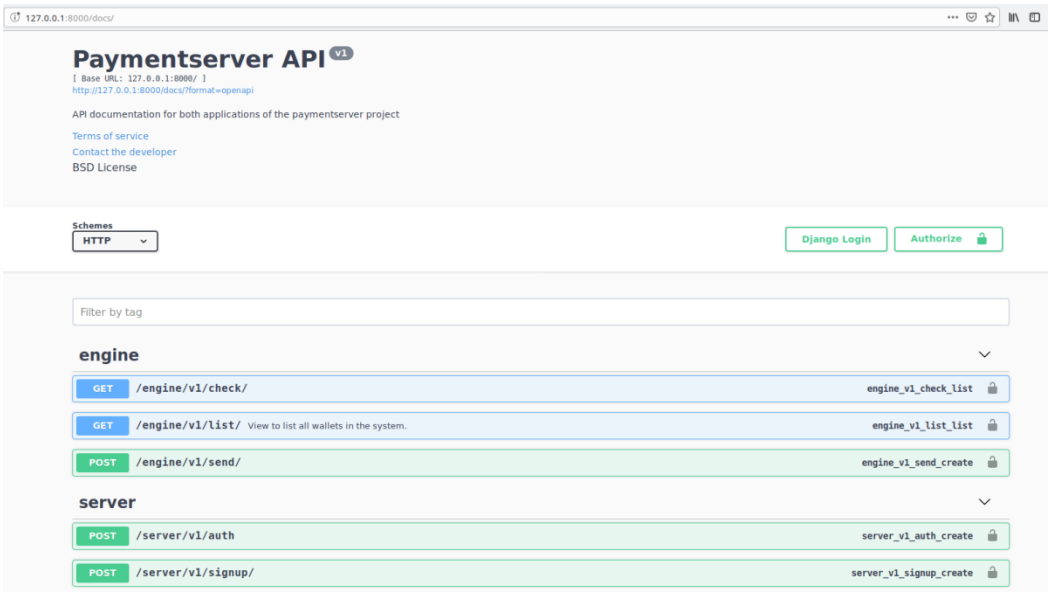

Figura 4.2.5.4: Listado de Endpoints de las APIs del proyecto Paymentserver, separadas por aplicación
Fuente: Elaboración propia – 2019

[18] String que aparece como primer atributo descriptivo en una definición de función en Python https://www.python.org/dev/peps/pep-0257/

4.3. Pruebas y Resultados

El quinto y último sprint de la planificación fue dedicado a la realización de diversas pruebas funcionales realizadas utilizando el software *Postman*, que permite realizar llamadas a los endpoints de una API una vez el servidor se encuentra operativo.

En primera instancia, se probó la capacidad de autenticación del endpoint '*server/v1/auth*'. Al crear los usuarios con un formulario de creación incorrectamente validado o sin utilizar el endpoint de creación de usuarios implementado con API Rest, se evidenció que las contraseñas eran guardadas en el sistema sin el debido *hashing* y por ende, no generaban los tokens de autenticación necesarios para la autorización del API. Esto se evidenció con un número de 4 usuarios registrados, 2 usuarios con tarjeta y 2 usuarios administradores. A partir de aquí se implementó para la interfaz administrativa un formulario correctamente validado y que generaba el *hashing* correcto de las contraseñas y por ende la generación de tokens de autenticación para los usuarios.

Otras pruebas realizadas fueron las de registrar usuarios con seriales o nombres de usuarios repetidos. En el caso de CardUser y AdminProfile, al ser creados como derivados del modelo de autenticación de Django, ya poseen una validación de nombre único, pero en el caso de los Puntos de venta, fue necesario añadir esta restricción en el modelo de usuario, para finalmente lograr la respuesta esperada.

Figura 4.3.1: Mensajes de error al validar existencia de usuarios en Interfaz de Administración
Fuente: Elaboración propia – 2019

La necesidad de esta restricción y, por ende, su respectiva validación, surgió al momento de crear un Punto de Venta repetido, pues si bien el punto de venta no se creaba, el error se levantaba a través de los métodos de creación de billeteras de bitcoinlib, no por una validación desde la creación de un objeto PoSProfile.

De igual forma al agregar el parámetro del identificador del punto de venta en la llamada para obtener el token de autenticación, se realizaron un par de pruebas para verificar que los mensajes eran correctos. En caso de requests en los que el parámetro 'pos_id' se encuentra ausente, se despliega correctamente el siguiente mensaje de error:

Figura 4.3.2: Mensaje de campo requerido del endpoint 'server/v1/auth/'
Fuente: Elaboración propia – 2019

En tanto que en el caso en el que el punto de venta no se encuentre registrado en el sistema, aunque el usuario si se encuentre registrado y posea un token de autenticación, se muestra correctamente un mensaje que impide continuar el flujo, sin el token de autenticación:

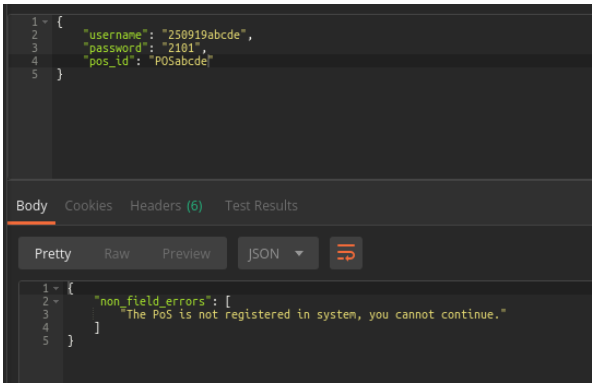

Figura 4.3.3: Mensaje de punto de venta no registrado del endpoint 'server/v1/auth/'
Fuente: Elaboración propia – 2019

Además, se realizaron pruebas de solicitudes erróneas al API con parámetros faltantes o incorrectos, obteniendo respuestas con errores HTTP 400 (Bad Request), en dichos casos, y respuestas HTTP 200 con la información procesada correctamente al momento de enviar los datos de forma correcta, y asimismo, pruebas de despliegue de la información en las interfaces y las diversas operaciones de Creación, Lectura y Eliminación, por lo que el flujo de respuestas de la API y de la interfaz de la aplicación *Authserver* cumplió satisfactoriamente las pruebas realizadas.

En el caso de la aplicación *Paymentengine*, las pruebas fueron un tanto más detalladas. En principio se generaron pruebas para la consulta de balances de una billetera. A partir de estas pruebas se pudo deducir que era necesaria la actualización de UTXOs de una billetera antes de realizar la consulta, pues en las primeras 5 consultas los balances retornaban desactualizados después de agregar balance a las billeteras de forma externa (a partir de una billetera con balance activo en Dash).

Asimismo, se realizaron pruebas en los tiempos de respuesta de la petición a la función de consulta de balances. A partir de los resultados se concluyó que una llamada redundante a la función *scan*()

de bitcoinlib.wallets triplicaba el tiempo de respuesta del endpoint de chequeo de balances, como se observa en la tabla 4.3.1:

	Tiempo mínimo	**Tiempo máximo**
Con llamada a wallet.scan()	12.5s	40s
Sin llamada a wallet.scan()	2.5s	5.6s – 6s

Tabla 4.3.1: c

De igual forma, ya que la llamada al método *scan*() se repetía en el endpoint de envío de transacciones para un chequeo previo de UTXOs y escaneo antes de realizar la transacción, se obtuvieron los siguientes resultados:

	Tiempo mínimo	**Tiempo máximo**
Con llamada a wallet.scan()	15.5s	+60s
Sin llamada a wallet.scan()	3.5s	5.6s – 6s

Tabla 4.3.2: Tiempos de llamadas para envío de transacciones

Gracias a las pruebas se verificó que la actualización de balance se hacía correctamente con el uso del método *utxos_update*(), que no realiza la generación de nuevas direcciones en cada llamada a diferencia del método *scan*() utilizado antes de completar las pruebas.

En cuanto al funcionamiento de los endpoints se generaron diversas llamadas para la prueba de entrega de mensajes, lo que llevó a mejorar el lenguaje de la respuesta y establecer el código 406 para el envío de mensajes personalizados respecto al sistema, y no mensajes inherentes al funcionamiento de las bibliotecas Django Rest Framework y Bitcoinlib respectivamente.

Respecto al monto mínimo de transacciones se establecen dos resultados importantes de las pruebas. Al intentar realizar una transacción con una fracción menor a 1000 Duffs, bitcoinlib emite un mensaje de error indicando que el monto mínimo de transacción es 1000 Duffs, debido a las cuotas de transacción por defecto que la biblioteca establece al construir la transacción. Otra conclusión importante es que en el 30% de las transacciones se obtuvieron errores de cuota de transacción (min relay fee not met – Error 26) al utilizar 1000 Duffs como cantidad de transacción, debido a la cuota de transacción necesaria para construir una transacción con una fracción tan pequeña. En el 40% de las transacciones realizadas con esta cantidad se evidenció que la transmisión a la *blockchain* no es realizada, debido a la baja cuota para realizar la transacción y la misma se convierte en una transacción con baja confiabilidad. El 30% de transacciones restantes realizadas con una cuota de 1000 Duffs fueron transmitidas satisfactoriamente. Se ilustran estos resultados en la figura 4.3.4:

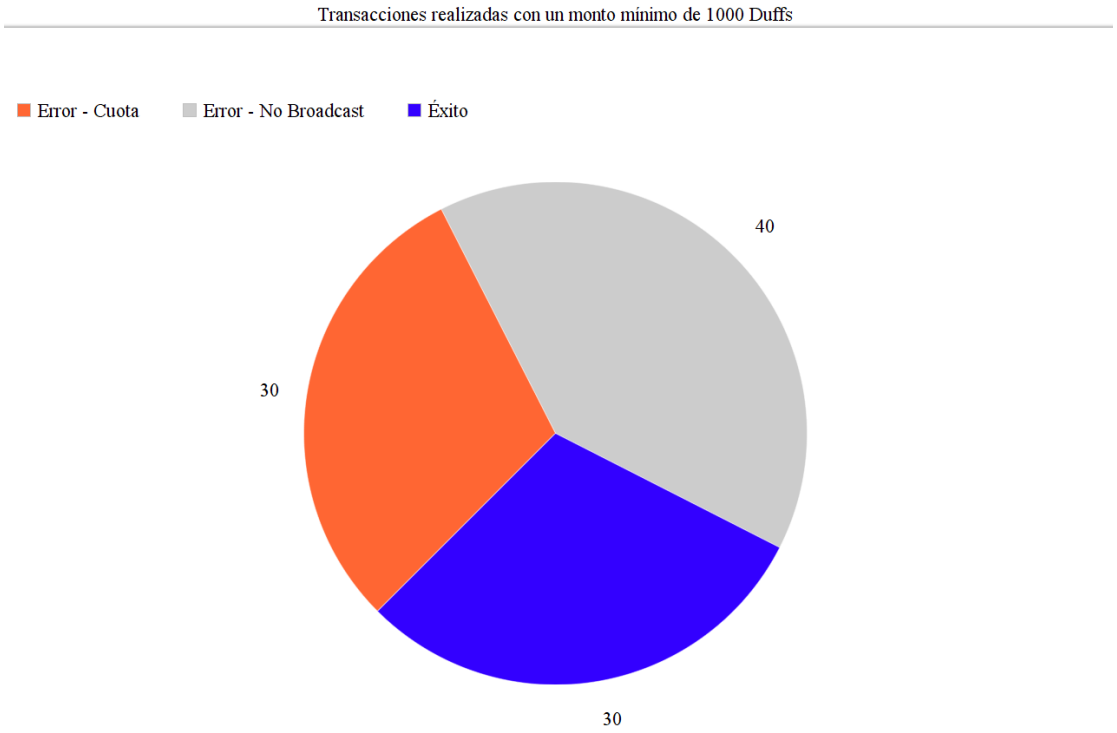

Figura 4.3.4: Gráfica de envío de transacciones con un monto de 1000 Duffs
Fuente: Elaboración propia – 2019

A partir de las pruebas anteriores se confirmó una fracción mínima para las transacciones de 100000 Duffs, realizando una ligera conversión entre el DASH y el dólar como moneda fiduciaria para establecer una fracción de DASH que se asemeje a un micropago equivalente en moneda fiduciaria, con una cuota estimada de transacción de 1000 Duffs por transacción.

Finalmente se realizaron pruebas en relación a los proveedores de servicio. Estas pruebas fueron relativamente delicadas, dado que bitcoinlib cuenta con la opción de configurar proveedores en línea, como diversos exploradores de bloques y un cliente conectado al full nodo para transmitir las transacciones a la red.

En el caso de la red Dash Testnet, no existen exploradores en línea actuales cuya API sea pública o sin protección contra ataques de denegación de servicio que pueda ser accedida por bitcoinlib. De igual forma, se hicieron pruebas de transmisión desde un nodo completo de Dash Testnet, sin éxito. Al realizar las pruebas con Dash Mainnet, se evidenció un 100% de solicitudes de consulta de balance exitosas utilizando el API del explorador en línea *chainz.cryptoid.info*. Este proveedor sirve principalmente para consulta, y provee una API KEY gratuita que puede ser configurada en *providers.json* de bitcoinlib para consultar. Sin embargo, el API no provee operaciones de transmisión de transacciones.

Para la transmisión de transacciones son pocos los exploradores de bloques que proveen una API KEY sin pago para la criptomoneda Dash. Una de los exploradores que otorga acceso limitado a una API KEY gratuita con transmisión de transacciones es *blockcypher.com*. No obstante, cabe destacar que las consultas y operaciones al API de Blockcypher de forma gratuita está limitada a 200 solicitudes por hora, donde se hace una solicitud por cada dirección de la billetera al momento de solicitar balances y actualizar UTXOs, por lo que el número de solicitudes se agota en pocas transacciones. De las transacciones generadas solo configurando Blockcypher como proveedor de servicio, solo un 20% eran transmitidas exitosamente a la blockchain antes de agotar la cuota de solicitudes.

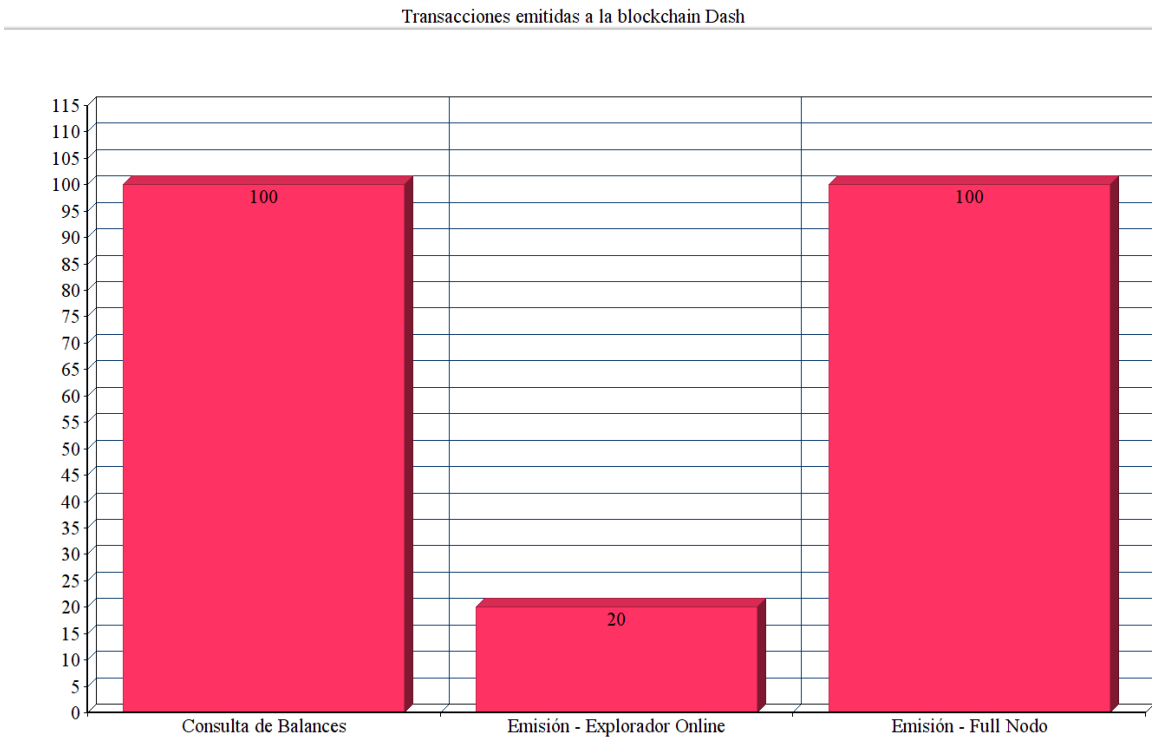

Figura 4.3.5: Gráfica de emisión de transacciones a la Blockchain Dash
Fuente: Elaboración propia – 2019

La última prueba en este último apartado se realizó instalando el nodo completo de la criptomoneda Dash, que requiere actualización constante y por ende, conexión continua a internet. Al conectar el *daemon* del nodo completo (dashd) como proveedor de servicio para la criptomoneda Dash, las transmisiones de transacciones se realizaban correctamente una vez tomados en cuenta otros parámetros como monto de la transacción para la construcción de las mismas, a pesar de los errores recibidos en los intentos de conexión a los exploradores de bloques en línea, por lo que se concluyó la necesidad de poseer el nodo completo de la criptomoneda Dash instalado en el equipo contenedor de la aplicación, para la transmisión de las transacciones y aumentar la disponibilidad del sistema.

4.3.1. Resultados asociados a los requerimientos funcionales

- La aplicación provee dos formas de entrada de datos al sistema, una a través de la interfaz de administración con un formulario web, y otra a través de un *endpoint* de registro especializado para el ingreso de tarjetas.
- El sistema de gestión de usuarios muestra dos interfaces para visualizar la información del usuario, una desde el lado administrador donde se observa el detalle de una tarjeta y su billetera asociada, y otra a través de una interfaz para los usuarios, en el que pueden observar tanto el detalle de su billetera, como las transacciones que haya realizado y sus IDs correspondientes. Es decir, la interfaz cubre satisfactoriamente las operaciones CRUD.
- La aplicación provee entrada de datos validada en sus APIs, tanto en la aplicación *Authserver* como en la aplicación *Paymentengine*.
- El motor de pagos (*Paymentengine*) utiliza los tokens de autenticación generados desde *Authserver*, por lo que utiliza el mismo esquema de autenticación que el resto de la plataforma
- El endpoint '*engine/v1/check*' permite realizar el correcto chequeo de balances de una billetera especificada por nombre.
- El endpoint '*engine/v1/send*' permite realizar transacciones en fracciones de Dash (Duffs) una vez verificado internamente el balance de la billetera de la tarjeta indicada.

4.3.2. Resultados asociados a los requerimientos no funcionales

- Se realizó una interfaz muy sencilla basada en Bootstrap 3 con botones básicos asociados a cada funcionalidad del API.
- Los endpoints en general proveen mensajes de error y éxito descriptivos.
- La arquitectura de la aplicación permite su modularidad, de forma que se puede diseñar un nuevo motor de pagos de forma independiente al esquema de autenticación y viceversa.
- Todas las contraseñas en el sistema se encuentran validadas y guardadas de forma que no pueden ser accedidas fácilmente por un tercero.

5. CONTRIBUCIONES, LIMITACIONES Y TRABAJOS FUTUROS

5.1. Contribuciones

La solución planteada cumplió con el objetivo principal de permitir, a través de un esquema de autenticación basado en un servidor con una base de datos de usuarios, realizar transacciones que puedan transmitirse a una cadena de bloques de una determinada moneda.

La solución fue creada de forma modular utilizando una arquitectura de servicios bajo dos API Rest, lo que permite ubicar rápidamente los servicios que se prestan y en caso que necesiten ser mejorados o expandidos, estas mejoras puedan hacerse de forma sencilla, pues esta arquitectura de servicios no requiere una especificación de casos de uso muy extensa o intrincada.

5.2. Limitaciones

En cuanto a las limitaciones presentes se debe tomar en cuenta el tipo de red a utilizar al momento de hacer las pruebas, pues debido a la falta de disponibilidad e infraestructura en la red de pruebas Dash Testnet, fue necesario utilizar fondos reales de la red principal, Dash Mainnet para la realización de pruebas, por lo que las mismas fueron limitadas.

De igual manera, existe una limitación en el caso de los exploradores online, pues si bien estos generan cierta ventaja en la escalabilidad de la solución al transmitir las transacciones a la *blockchain* de la criptomoneda Dash sin que sea estrictamente necesario la presencia del nodo completo en el equipo que aloja el servidor de pagos, en esta etapa de desarrollo inicial de la solución resulta costoso contratar un plan de uso para las API KEY que los exploradores ofrecen, que inician desde aproximadamente $75 por mes, pero manteniendo aún una limitación importante en el número de solicitudes que se puedan hacer al API, por lo que la disponibilidad de la plataforma depende del plan contratado.

Esta última limitación abre espacio para un análisis de escalabilidad vs. costo al momento de masificar la solución. El uso de un API externa resuelve muchos problemas de escalabilidad, y puede ser factible un pago para el uso de este tipo de servicios si la solución es masificada, sin embargo, la instalación de un nodo completo, en el caso específico de este Trabajo Especial de Grado que corresponde al de la criptomoneda Dash, si bien no es enteramente escalable, mantiene un tamaño en disco manejable por un equipo actual, que no requiera tantos recursos con la ventaja de ser completamente gratuito.

El trabajo, por temas de alcance, se encuentra limitado a una sola criptomoneda, sin embargo, dada la naturaleza de las billeteras creadas gracias a la biblioteca bitcoinlib, permite añadir una capa de

flexibilidad al momento de crear las billeteras, por lo que se puede expandir el alcance a nivel de criptomonedas a utilizar

En cuanto a la rapidez de la solución, si bien la creación y emisión de la transacción se generan en tiempos que varían entre los 3 y 8 segundos, el factor de confirmación, que sigue siendo el más importante en el aspecto de robustez de la criptomoneda, sigue dependiendo altamente de la velocidad de la red a utilizar. Los tiempos de confirmación varían entre 4 y 6 minutos, tiempos típicos de una transacción InstantSend de la criptomoneda Dash.

5.3. Trabajos Futuros

Como elemento primordial a considerar en los trabajos futuros, debe tomarse en cuenta el diseño, levantamiento de requerimientos y de infraestructura a nivel de redes para el despliegue del Servidor de Pagos en un servidor en la nube. Esto requiere de la implementación de un balanceador de cargas para poder procesar una creciente cantidad de solicitudes y tener en cuenta la escalabilidad de la solución a futuro, con una cantidad importante de dispositivos de punto de venta.

A nivel de interfaces, para extender y mejorar las funcionalidades básicas desarrolladas en este Trabajo, se plantea expandir las interfaces administrativas a nivel de usuario, agregando esquemas adicionales de control de la tarjeta y de control de cuentas que permita al usuario administrar y cambiar contraseñas de ingreso a la interfaz, manejo de balances detallados entre sus direcciones (UTXOs) y poseer distinción entre dichas direcciones De igual forma a nivel administrativo, se puede planear una sección similar para el comercio, de forma que el mismo pueda realizar el manejo de su billetera asociada y de sus fondos con mayor libertad.

CONCLUSIONES

A lo largo del presente Trabajo Especial de Grado, se realizó la construcción de una solución basada en tecnologías web para un esquema económico con amplias raíces en la internet como lo es la criptoeconomía, en pro de expandir poco a poco el alcance de las criptomonedas, masificando su uso y adaptación por parte de la población actual. A partir de esta solución, se pudo determinar diversos factores importantes que influyen en la implementación de soluciones de este tipo, como disponibilidad, escalabilidad y costos de implementación, que se evidenciaron al momento de utilizar proveedores de servicio en línea para el envío de transacciones y la implementación de un nodo completo de la criptomoneda Dash, permitiendo generar un análisis que vale la pena realizar dependiendo del estado en producción de la solución. Asimismo, también fue posible determinar el monto mínimo de una transacción en esta criptomoneda que sea compatible con la definición de un micropago y que a su vez, dicho pago pueda ser aceptado por la red Dash y confirmado rápidamente.

En general, los objetivos planteados para este proyecto fueron cumplidos, desarrollando aplicaciones de software que cumplieron los requerimientos funcionales y no funcionales planteados como requerimientos iniciales de este proyecto. La investigación de diversas tecnologías, que definen las bases para el desarrollo de una aplicación web basada en servicios tal como lo son Python y Django, y sus bibliotecas asociadas, así como de algunos trabajos previos y las tecnologías actuales en relación al mundo de las criptomonedas, en la que destaca ampliamente la biblioteca Bitcoinlib, permitió visualizar una forma sencilla de implementar la solución, empleando dos componentes iniciales de software, e incluso visualizar algunas mejoras a nivel de infraestructura para expandir y escalar el proyecto y su capacidad de respuesta.

Estos objetivos fueron logrados aplicando una metodología ágil como SCRUM, ampliamente utilizada en múltiples procesos de desarrollo en la actualidad, y que fue adaptada para lograr el desarrollo en el menor tiempo posible. Esto deja en evidencia la importancia de las metodologías ágiles en los desarrollos actuales y como permiten tener un mínimo producto viable, funcional, modular y con altas probabilidades de expansión y mejora.

Las pruebas realizadas permitieron delimitar y perfilar el camino del desarrollo realizado, dejando en evidencia características importantes de los métodos utilizados y permitiendo crear restricciones necesarias en la creación de los modelos de datos básicos para esta solución. Esto deja en claro la importancia de tener un conjunto de pruebas básicas que permitan formar un camino cada vez más claro en el proceso de desarrollo.

Este proyecto se perfila como un proyecto base que permite generar diversos módulos, con apoyo tanto en el área de desarrollo web, de infraestructura de redes y desarrollo de hardware y software, que puede ser extendido ampliamente con el desarrollo de nuevos proyectos asociados en cada una de las áreas mencionadas, contribuyendo para su uso continuo en una sociedad donde las criptomonedas aceleran su crecimiento y se plantean como mucho más que un esquema económico alternativo.

REFERENCIAS

[1] Nakamoto, S. *Bitcoin*: *A Peer-to-Peer Electronic Cash System*, 2008. [En línea]. Disponible: https://*bitcoin*.org/*bitcoin*.pdf

[2] Franco, P., *Understanding Bitcoin*, 2015. Wiley

[3] Lee, K.C., *Handbook of Digital Currency, Bitcoin, Innovation, Financial Instruments, and Big Data*. 2015. Amsterdam: Elsevier

[4] Caetano, R., *Learning Bitcoin: embrace the new world of fiance by leveraging the power of crypto-currencies using Bitcoin and the Blockchain*. 2015. Birmingham: Packt Pub.

[5] Poon, J. and Dryja, T., *The Bitcoin Lightning Network: Scalable off-chain instant payments.* 2016. [En línea]. Disponible: *https://lightning.network/lightning-network-paper.pdf*

[6] Bitcoin Improvement Proposal 44. 2014.[En línea]. Disponible: *https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki*

[7] Antonopoulos, A., *Mastering Bitcoin: Programming the Open Blockchain*, 2017. O'Reilly.

[8] Dash Core Group, Inc. *Dash Documentation — Dash latest documentation*. [En línea] Docs.dash.org. Disponible: *https://docs.dash.org/en/stable/*

[9] Micali, S. and Rivest, R., *Micropayments Revisited*. 2019. Springer

[10] Dash-docs.github.io. 2019. *Developer Guide - Dash*. [En línea]. Disponible: *https://dash-docs.github.io/en/developer-guide#instantsend*

[11] Portillo García, J., Bermejo Nieto, A. y Bernardos Barbolla, A. *Tecnología de identificación por radiofrecuencia (RFID)*. 2008. Madrid: Fundación Madri+d para el Conocimiento.

[12] Finkenzeller, K y Müller Dörte, *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. 2012. Chichester: Wiley

[13] ATMEL, Application Note, *Requirements of ISO/IEC 14443 Type B Proximity Contactless Identification Cards*.

[14] *Changing The World: RFID and NFC technologies for the future - Whitepaper*, Biocrypt.tech, 2019. [En Línea]. Disponible: *https://biocrypt.tech/BioCrypt_Whitepaper.pdf*. [Accedido: 03 Feb. 2019]

[15] *DASH Text*, 2019. [En Línea]. Disponible: *https://dashtext.io/en/home-2/*. [Accedido: 03 Feb. 2019]

[16] Satpathy, T. (Ed.). *A Guide to the Scrum Body of Knowledge : SBOK Guide*. 2013 [En Línea]. Disponible: *http://gutenberg.cc/*

[17] Driessen, V. *A Successful Git Branching Model.* 2010. [En Línea]. Disponible: *https://nvie.com/posts/a-successful-git-branching-model/*

[18] The Python Tutorial. [En Línea] Disponible en: *https://docs.python.org/3/tutorial/index.html*

[19] Ravindran, A. *Django Design Patterns and Best Practices*. 2015. Packt Publishing.

[20] Django Rest Framework. [En Línea] Disponible en: *https://www.django-rest-framework.org/*

[21] Bitcoinlib 0.4.5 Documentation. [En Línea] Disponible en: *https://bitcoinlib.readthedocs.io/en/latest/*

ANEXOS

A-1. Especificación de las API Rest desarrolladas

Ruta	Accion	Tipo de Petición	Estructura de la peticion

'server/v1/auth'	Autenticación de usuario en el api para obtener un token de autenticación	POST	{
			"username":<serial de la tarjeta>,
			"password":<PIN>,
			"pos_id":<serial del PoS>
			}

'server/v1/signup'	Creación de usuarios a través del api	POST	Header:
			Authorization: Token <token de autenticación>
			
			Request:
			{
			"username":<serial de la tarjeta>,
			"password":<PIN>
			}

'engine/v1/list'	Lista todas las wallets disponibles en el Sistema	GET	Header: Authorization: Token <token de autenticación> Request: En blanco

'engine/v1/check/'	Chequea el balance de un nombre de wallet indicado	GET	Header:
			Authorization: Token <token de autenticación>
			
			Request:
			{
			"wallet":<nombre del wallet>
			}

'engine/v1/send/'	Envía una transacción por la cantidad indicada al punto de venta indicado tomando las UTXOs de la tarjeta indicada	POST	Header:
			Authorization: Token <token de autenticación>
			
			Request:
			{
			"card_id":<serial de la tarjeta>,
			"pos_id":<serial del PoS>,
			"amount":<cantidad de la transacción>
			}

Tabla A-1.1: Especificación de Endpoints del proyecto Paymentserver

A-2. Instalación y Configuración - Python Django

Para realizar la instalación de Python Django es necesario contar con un equipo que tenga Python instalado. En el caso de los sistemas operativos Linux, Python ya se encuentra instalado por defecto, tanto en su versión 2 como en su versión 3. Al tener Python instalado es importante realizar la instalación de un entorno virtual, pues éste provee una capa de separación entre la instalación de Python principal en el sistema operativo y las instalaciones necesarias para Django.

Para este Trabajo Especial de Grado se utilizó Virtual Env Wrapper para agilizar la creación de entornos virtuales. Para instalar Virtual Env Wrapper es necesario seguir los siguientes pasos, para Python 3:

1. Instalar virtualenv y virtualenvwrapper usando PIP.

   Pip es el instalador de paquetes de Python y puede ser utilizado incluso dentro de los entornos virtuales. Para instalar virtualenv y virtualenvwrapper se utiliza el siguiente comando:

   ```
   sudo pip install virtualenv virtualenvwrapper
   ```

   En el caso de Linux. Una vez instalado, se ejecuta el siguiente comando para añadir una línea al script del Shell, para poder configurar la versión de Python a utilizar dentro de cada uno de los entornos virtuales creados con virtualenvwrapper:

   ```
   echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >> ~/.bashrc
   ```

   Luego, se configura el directorio en el que se crearán todos los entornos virtuales y el directorio en el que se encuentra el Shell script para activar los entornos virtuales:

   ```
   echo "export WORKON_HOME=~/Env" >> ~/.bashrc
   echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
   ```

   Finalmente, se agrega esta línea de comando

   ```
   source ~/.bashrc
   ```

   Y ya se encuentra el directorio Env en el Home del sistema. Para crear un nuevo entorno virtual, desde cualquier directorio es posible ejecutar el comando `mkvirtualenv <nombre_del_entorno>` y para comenzar a trabajar dentro del mismo, se ejecuta el comando `workon <nombre_del_entorno>`

2. Una vez dentro del entorno, la instalación de Django se ejecuta con el comando

   ```
   pip install django
   ```

   Y una vez instalado, se procede con la configuración. El primer paso es crear el proyecto con

```
django-admin.py startproject firstsite
```

Luego, hay que ubicarse en el primer nivel del proyecto haciendo `cd /<nombre del proyecto>` y desde allí ejecutar las migraciones iniciales para el proyecto. Muchas de ellas son migraciones para los modelos internos de Django y el funcionamiento de diversos backends y middlewares instalados por defecto. El comando es `python manage.py migrate`.

Django provee por defecto una base de datos SQLite3, por ser portable y ligera, sin embargo, es posible configurar un manejador de base de datos diferente dependiendo de las necesidades del proyecto. Al ser creada la base de datos, es posible crear un superusuario con el comando `python manage.py createsuperuser`. Una vez creado el superusuario, es posible iniciar el servidor de Django con el comando `python manage.py runserver`, y finalmente poder ver la interfaz mostrada en la figura A-2.1 en el navegador, ingresando a 127.0.0.1:8000.

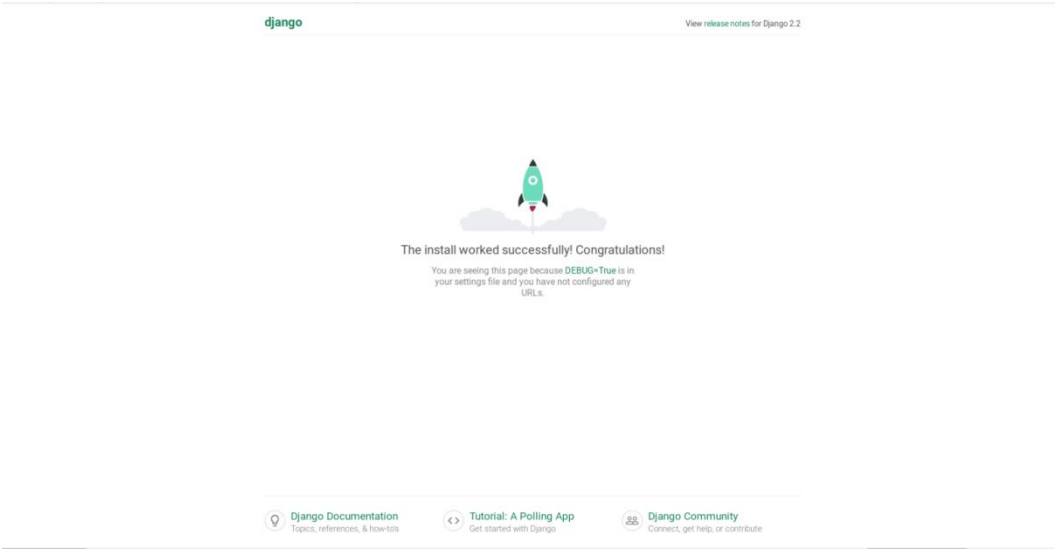

Figura A-2.1: Instalación exitosa de Python Django
Fuente: Elaboración propia – 2019

En el directorio general del proyecto existe un archivo llamado *settings.py* que contiene todas las configuraciones del proyecto y desde el cual se pueden gestionar las bibliotecas adicionales instaladas, manejadores de bases de datos, archivos estáticos, internacionalización y localización, middlewares y backends para el correcto funcionamiento de la aplicación Django.

A-3. Instalación y Configuración - Django Rest Framework

La instalación de Django Rest Framework se hace de forma similar a la instalación de Django. Una vez dentro del entorno virtual en el que se encuentra instalado Django, se ejecuta el siguiente comando:

```
pip install djangorestframework
```

Luego, en el archivo *settings.py* del proyecto Django, es necesario añadir dos parámetros importantes. El primero es la aplicación en las installed apps de Django, de la siguiente forma:

```
INSTALLED_APPS = [
...
'rest_framework',
]
```

El segundo parámetro es añadir el diccionario REST_FRAMEWORK, que contendrá todas las configuraciones necesarias para el funcionamiento de Django Rest Framework. Finalmente, en el archivo *urls.py* ubicado en el directorio de configuración del proyecto, se agrega la siguiente URL para tener acceso a las URLs de Django Rest Framework y navegar las APIs creadas:

```
urlpatterns = [
    ...
    path('api-auth/', include('rest_framework.urls'))
]
```

A partir de estos parámetros iniciales, es posible iniciar la importación de diversas clases y funciones importantes a utilizar durante la creación de una API Rest utilizando Django Rest Framework. Existe un tutorial bastante detallado para el uso de sus clases en la página principal de documentación, referenciada en [20].

A-4. Instalación y Configuracion - Bitcoinlib

Para la instalación de bitcoinlib, se utiliza nuevamente el comando pip desde el entorno virtual en el que se encuentre la aplicación Django, pues bitcoinlib se encuentra disponible en el Python Package Index (PyPI):

```
pip install bitcoinlib
```

Bitcoinlib requiere ciertas dependencias de algunas bibliotecas criptográficas y de desarrollo de Python, tales como:

- Python-dev
- Python3-dev
- ecdsa
- pyaes
- scrypt
- sqlalchemy
- requests
- enum34 (para instalaciones antiguas de Python)

También es necesario especificar el directorio en el que se ubicará la base de datos a utilizar por bitcoinlib. En este trabajo especial de grado, se realizó está configuración en cada una de las vistas a utilizar:

```
test_databasefile = 'bitcoinlib.test.sqlite'
test_database = settings.BCL_DATABASE_DIR + test_databasefile
```

Figura A-4.1: Configuración de la base de datos de bitcoinlib
Fuente: Elaboración propia – 2019

Los archivos de configuración de bitcoinlib suelen ubicarse en el directorio Home (al usar Linux) en un directorio oculto llamado .bitcoinlib. En este directorio se encuentra el directorio config, que contiene dos archivos de configuración. Para realizar la adición de nuevos proveedores de servicio, es necesario acceder al archivo *providers.json*, tal como se aprecia en la siguiente figura:

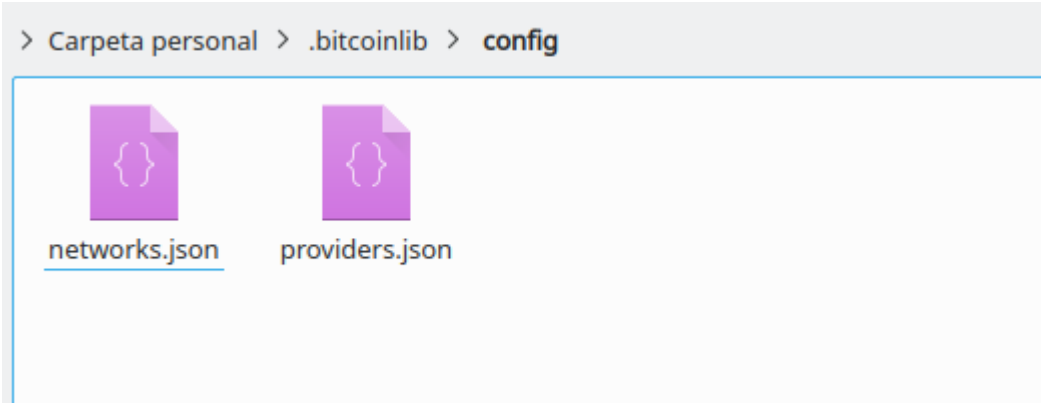

Figura A-4.2: Ubicación de los archivos de configuración de bitcoinlib
Fuente: Elaboración propia – 2019

Ya existen algunos proveedores de servicio disponibles para criptomonedas como Bitcoin, Litecoin y Dash, que son configurables de diferentes maneras. En el caso de los exploradores online, se puede añadir un API KEY para un proveedor de la siguiente forma:

```
},
  "blockcypher.dash": {
  "provider": "blockcypher",
  "network": "dash",
  "client_class": "BlockCypher",
  "provider_coin_id": "",
  "url": "https://api.blockcypher.com/v1/dash/main/",
  "api_key": "c26436d16fbf4a2a8e09103d0f37c9be",
  "priority": 10,
  "denominator": 1,
  "network_overrides": {"prefix_address_p2sh": "05"}
},
```

Figura A-4.3: Configuración de un proveedor de servicio online
Fuente: Elaboración propia – 2019

Por otro lado, si se requiere añadir la configuración para un nodo completo de alguna criptomoneda, puede hacerse de la siguiente manera:

```
"dashd": {
  "provider": "dashd",
  "network": "dash",
  "client_class": "DashdClient",
  "provider_coin_id": "",
  "url": "http://user:password@server_url:9998",
  "api_key": "",
  "priority": 11,
  "denominator": 100000000,
  "network_overrides": null
},
```

Figura A-4.4: Configuración de un nodo completo de la criptomoneda Dash como proveedor de servicio en bitcoinlib

Fuente: Elaboración propia – 2019

Con estas configuraciones realizadas, bitcoinlib será capaz de generar transacciones y consultar balances dentro de un intérprete Python o una aplicación Django.