



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA ELECTROMECÁNICA

**APLICACIÓN PARA LA TARIFICACIÓN  
DE SEGUROS DE COCHE  
RECOGIENDO DATOS MEDIANTE EL  
USO DE BLOCKCHAIN**

Autor: Ramiro Rivera Rodríguez

Director: José Manuel Suárez Fernández

**Madrid**

Agosto 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título  
**Aplicación para la tarificación de Seguros de coche recogiendo datos  
mediante el uso de Blockchain**  
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el  
curso académico 2017/2018 es de mi autoría, original e inédito y  
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es  
plagio de otro, ni total ni parcialmente y la información que ha sido tomada  
de otros documentos está debidamente referenciada.

  
Fdo.: Ramiro Rivera Rodríguez

Fecha: 27/08/2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

  
Fdo.: José Manuel Suárez

Fecha: 27 / 8 / 2018

## **AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESIS O MEMORIAS DE BACHILLERATO**

### **1º. Declaración de la autoría y acreditación de la misma.**

El autor D. **Ramiro Rivera Rodríguez** DECLARA ser el titular de los derechos de propiedad intelectual de la obra: **Aplicación para el proceso de tarificación de Seguros de coche mediante el uso de *blockchain***, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

### **2º. Objeto y fines de la cesión.**

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

### **3º. Condiciones de la cesión y acceso**

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

- a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.
- b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
- c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
- d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
- e) Asignar por defecto a estos trabajos una licencia Creative Commons.
- f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

### **4º. Derechos del autor.**

El autor, en tanto que titular de una obra tiene derecho a:

- a) Que la Universidad identifique claramente su nombre como autor de la misma
- b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
- c) Solicitar la retirada de la obra del repositorio por causa justificada.
- d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

### **5º. Deberes del autor.**

El autor se compromete a:

- a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
- b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
- c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
- d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

**6º. Fines y funcionamiento del Repositorio Institucional.**

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 27 de agosto de 2018,

**ACEPTA**



Fdo. Ramiro Rivera Rodríguez

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA ELECTROMECÁNICA

**APLICACIÓN PARA LA TARIFICACIÓN  
DE SEGUROS DE COCHE  
RECOGIENDO DATOS MEDIANTE EL  
USO DE BLOCKCHAIN**

Autor: Ramiro Rivera Rodríguez

Director: José Manuel Suárez Fernández

**Madrid**

Agosto 2018



# APLICACIÓN PARA LA TARIFICACIÓN DE SEGUROS DE COCHE RECOGIENDO DATOS MEDIANTE EL USO DE BLOCKCHAIN

**Autor: Rivera Rodríguez, Ramiro**

Director: Suárez Fernández, José Manuel

Entidad Colaboradora: GMS Management Solutions

## RESUMEN DEL PROYECTO

Este Proyecto nace en un contexto de revolución digital en el que un gran número de empresas de todo tipo de tamaños y sectores buscan revisar y comprender los nuevos avances tecnológicos para reinventarse, mejorar sus servicios o, incluso, simplemente para ponerse a la altura de sus competidores. Muchos equipos de Investigación, Desarrollo e Innovación (I+D+i) hacen grandes esfuerzos por evaluar la viabilidad y escalabilidad de la aplicación en su modelo de negocio de estos avances relacionados con la digitalización.

El sector de los “seguros de automóvil”, sobre el que pone el foco este Proyecto, no escapa a esta corriente y también invierte en la investigación de las nuevas herramientas que van surgiendo, con el objetivo de añadir valor a sus servicios. Por un lado, son conscientes de la importancia de la presencia del teléfono móvil, y por otro, debido a la intrínseca componente legal que acompaña a sus servicios, se están interesando fuertemente por la tecnología *blockchain*.

*Blockchain* es un tipo de tecnología de registro distribuido. Estas tecnologías consisten en sistemas de almacenamiento de datos que cuentan con la peculiaridad de albergar varios participantes y de no depender de un organismo central que medie y controle las acciones ni cuente con autoridad para realizar cambios unilateralmente. De este modo, *blockchain* es, a grandes rasgos, una tecnología de almacenamiento y de transmisión de información, que no depende de un organismo central que la controle, que guarda estos datos de manera encriptada e indeleble –de modo que no se puedan manipular ni eliminar–, y que cuenta con una serie de ventajas en materia de seguridad y transparencia, frente a las tecnologías tradicionales de almacenamiento de datos. Tradicionalmente, la razón de ser de estos organismos centrales tiene que ver con la falta de confianza en el sistema, haciendo necesaria la existencia de una entidad dotada de autoridad para regular la validez de las transacciones que se llevan a cabo.

La disminución del uso del coche particular por parte de los ciudadanos (debido a la irrupción de modelos de negocio como *BlaBlaCar* y de los servicios de *car sharing*) tiene un impacto directo en el mercado de los seguros de automóvil. Las nuevas pautas de uso ofrecen al consumidor la posibilidad de abandonar la práctica tradicional de tener un contrato que dure todo el año con una única compañía de seguros, independientemente de las horas de uso que se le da al coche, y pueden animarlas a embarcarse en otras posibilidades que concuerden con un uso más reducido del automóvil particular.

La respuesta que este Proyecto da a esta situación consiste en una plataforma que hace de lugar de encuentro entre empresas aseguradoras y usuarios que desean estar

asegurados por estas compañías, pero únicamente para aquellos momentos en los que estén conduciendo su coche. Un sistema de contratos de seguros cortos, de un trayecto únicamente, desligaría al usuario de una única compañía y le permitiría beneficiarse de la competencia entre las empresas de la red. En un trayecto cualquiera, puede ser que la compañía A le dé el precio más barato de entre todos los que ofrecen las compañías presentes en la red, mientras que, en un trayecto diferente, sea, en cambio, la compañía B la que ofrezca al usuario el menor precio de todos.

Así, el principal objetivo del Proyecto es desarrollar una red entre iguales que sirva de punto de encuentro entre diferentes usuarios para intercambiar servicios. Según se ha concebido, no hay razón para que esta aplicación deba contar con una entidad centralizada que controle la plataforma, por lo que se ha decidido desarrollar una aplicación descentralizada, que otorgue el gobierno de la plataforma a la red, evitando así tanto la presencia de intermediarios que no aporten valor como eventuales situaciones de coacción o corrupción. El protocolo *Ethereum* permite desarrollar una aplicación de este tipo.

*Ethereum* es una red pública que utiliza el protocolo *blockchain* para funcionar. Para poder prescindir de una entidad central que valide el estado del sistema, el protocolo *blockchain* combina un sistema de transición de estados basado en la firma digital, con un algoritmo de consenso que garantiza que todos los participantes estén de acuerdo con las transacciones que se realizan. Se trata de una funcionalidad que asegura la validez del orden de las transacciones (es decir, la cronología de los estados) utilizando la llamada “Prueba de Trabajo”.

Una vez haya varias transacciones validadas por la red, estas serán añadidas a un bloque que se colocará detrás del último bloque añadido, apuntándole y formando una nueva cadena, un poco más larga, que se enviará inmediatamente a todos los nodos de la red. La información está, de este modo, distribuida en la red, por lo que es prácticamente imposible de atacar las actualizaciones que ha aceptado la red cuando los datos están siendo continuamente replicados en multitud de lugares físicos, al mismo tiempo. Del mismo modo, la información no se actualiza, sustituyendo lo inmediatamente anterior, sino que se añade, formando una cadena en el que todos los estados globales de la red y las transacciones que se han enviado son rastreables.

Los miembros de la red son los llamados nodos, rol que puede desempeñar quien desee participar de la red y posea una mínima capacidad computacional. Destacan, de entre estos nodos, los “nodos mineros”: máquinas que trabajan constantemente para validar la veracidad y cronología de las transacciones, buscando dar solución al problema matemático que plantea la Prueba de Trabajo, resultado de la encriptación de las transacciones. El algoritmo de consenso, para añadir un bloque, comprueba que el bloque anterior existe y es válido, que realmente es cronológicamente anterior al que se pretende añadir, y que algún “nodo minero” ha dado solución a la Prueba de Trabajo (esfuerzo por el que es recompensado, lo que supone un aliciente) y que esta es correcta. La razón de ser de la Prueba de Trabajo consiste en que, al conllevar la incorporación de cada bloque



de transacciones a la cadena un coste computacional, impide que un atacante pueda rehacer la cadena a su antojo. Así, se garantiza la confianza descentralizada en la red.

Tras haber comprendido su funcionamiento y sus ventajas, desplegaremos una red privada de tipo *Ethereum* en un nodo propio, creando el primer bloque de una cadena a la que se irán añadiendo los siguientes conforme se envíen transacciones. *Ethereum* es el protocolo que permite desarrollar aplicaciones descentralizadas, pues combina la lógica de la cadena de bloques con las infinitas posibilidades que ofrecen los llamados *Smart Contracts*, o contratos inteligentes. Se trata de códigos que, una vez se escriben y se envían a una dirección de la red descentralizada, no se pueden alterar, y que automatizan procesos que suelen requerir la validación de una tercera parte.

Un contrato inteligente recibe información de entrada, la procesa según las reglas establecidas en el contrato, y realiza las acciones que éste marque. Los contratos son un tipo de las conocidas como cuentas: objetos de *Ethereum* que se identifican por la dirección que ocupan en la red. Los contratos son cuentas que tienen código propio y que están, a su vez, controladas por código. Pueden disponer de fondos (la moneda en *Ethereum* se denomina *ether*) que serán controlados únicamente por su propio código interno. De esta manera, el “estado global” en *Ethereum* no es otra cosa que el registro de cuentas en cada momento, su balance de fondos y su código. El protocolo de cadena de bloques, por un lado, garantiza que este estado sea legítimo, y, por otro, permite analizar todas las transacciones históricas de la red.

La arquitectura de una aplicación descentralizada consiste, por un lado, en código desplegado en la red por medio de los contratos inteligentes, y, por otro, una interfaz que permita el acceso a estos por parte de los usuarios. Como existen muchas maneras de interactuar con las cuentas de *Ethereum* y con los métodos del código de los contratos, se ha decidido desarrollar una aplicación móvil intuitiva para acceder a la plataforma.

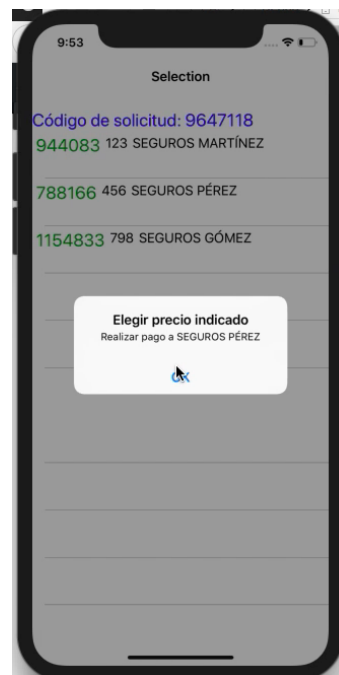
Así, una vez se ha configurado y desplegado la red privada con una nueva cadena de bloques sobre la que se va a ir añadiendo la información y los cambios que se producen en la plataforma, se han desarrollado cuatro contratos inteligentes que contienen toda la lógica requerida por la aplicación descentralizada. El primero acredita el registro en la plataforma por parte de un nuevo cliente usuario y gestiona sus trayectos (eventualmente estará asociado al *wallet* de la persona física y este a su identidad digital), el segundo hace lo propio con las aseguradoras y el tercero se despliega en el momento de la solicitud de un nuevo trayecto por parte de un cliente y contiene las condiciones para la transferencia de fondos en una u otra dirección. El último hace las veces de registro de la aplicación particular para facilitar la identificación de datos.

La aplicación se ha desarrollado en el entorno de desarrollo de *Xamarin*, ya que la llamada librería *Nethereum* acerca la biblioteca *Web3* a un proyecto en *.NET*. Esta ofrece una interfaz que facilita la comunicación con el nodo, posibilitando el despliegue de contratos inteligentes, el acceso a las llamadas a sus funciones y al envío de transacciones...

La aplicación móvil se ha desarrollado de modo que, en la primera pantalla, se pueda escoger entre proceder al registro en la plataforma, si no se ha realizado en una sesión previa, o bien al inicio de sesión en la aplicación haciendo uso del DNI y la contraseña indicados en su momento. Si se accede al registro, se deberá introducir todos los datos que correspondan. Se introduce el nombre, los apellidos, el DNI, la edad, el número de años con carné de conducir, la dirección, el país y la contraseña que se desee para acceder a la aplicación al iniciar sesión en futuras ocasiones. Si se desea solicitar un trayecto, se muestra un mapa con dos marcadores para arrastrar que señalan los puntos de partida y llegada, un botón que permite acceder a un desplegable para elegir el tipo de cobertura y, por último, un botón de confirmación del trayecto para avanzar a la siguiente pantalla. En ella se muestra una lista con todas las compañías de la plataforma y el precio que ofrecen, para que el usuario pueda así seleccionar uno y activar el contrato de seguros. Cada compañía tiene sus propias ponderaciones para cada parámetro que interviene en la ecuación de cálculo de precio (distancia, tipo de cobertura, edad, número de años con carné) y eventualmente podría cambiarlos para contratos futuros, con el objetivo de mejorar sus márgenes.



*Ilustración 2: Solicitud de un trayecto*



*Ilustración 1: Elección de precio y compañía*

El resultado es una aplicación independiente de una entidad centralizada, con especial seguridad en lo que se refiere al almacenamiento de la información, con transparencia y trazabilidad en las operaciones y que ofrece la automatización de un proceso que normalmente requiere de revisión por personas físicas. Se han conseguido, así, los objetivos de libre competencia, funcionamiento continuo de la aplicación, mayor eficiencia al eliminar intermediarios y garantía de la integridad de la información.

# MOBILE APPLICATION FOR CAR INSURANCE TARIFFICATION COLLECTING DATA USING BLOCKCHAIN

**Author: Rivera Rodríguez, Ramiro**

Supervisor: Suárez Fernández, José Manuel

Collaborating Entity: GMS Management Solutions

## ABSTRACT

This Project was born in a context of digital revolution in which a large number of companies of all sizes and sectors seek to review and understand the new technological advances to reinvent themselves, improve their services or even simply to catch up with their competitors. Many Research, Development and Innovation (R & D + i) teams make great efforts to evaluate the viability and scalability of the application in their business model of these advances related to digitization.

The "car insurance" sector, on which this Project focuses, does not escape this trend and also invests in the investigation of the new tools that are emerging, with the aim of adding value to its services. On the one hand, they are aware of the importance of the presence of the mobile phone, and on the other, due to the intrinsic legal component that accompanies their services, they are becoming strongly interested in blockchain technology.

*Blockchain* is a type of distributed ledger technology. These technologies consist of data storage systems that have the peculiarity of hosting several participants and not depending on third parties that mediate and control the actions or have the authority to make changes unilaterally. Thus, *blockchain* is, broadly speaking, a technology of storage and transmission of information, which does not depend on a central entity that controls it, in which data is stored in an encrypted and indelible way -so that it cannot be manipulated or eliminated-, and that has a number of advantages in terms of security and transparency, over traditional data storage technologies. Historically, the *raison d'être* of these central bodies has to do with the lack of trust in the system, making it necessary for the network to incorporate an entity endowed with authority to govern the validity of the transactions that are carried out.

The drop in the use of private cars by citizens (due to the emergence of business models such as *BlaBlaCar* and *car-sharing* services) has a direct impact on the car insurance market. The new usage guidelines offer the consumer the possibility to abandon the traditional practice of having a contract that lasts all year with a single insurance company, regardless of the hours of use that is given to the car, and may encourage them to look for other alternatives that better fit with a occasional use of the private automobile.

The answer this Project gives to this situation consists of a platform that functions as a meeting place between insurance companies and users who wish to be insured by these companies, but only for those moments in which they are driving their car. A system

of short, one-way insurance contracts would detach the user from a single company and allow him or her to benefit from competition between the companies in the network. On a given route, company A may announce the lowest price among all the companies present in the network, while, on a different route, it is company B that offers the user the lowest price among all those offered by the insurers participating in the platform.

Thus, the main objective of the Project is to develop a network among equals that serves as a meeting point between different users to exchange services. As it has been designed, there is no reason for this application to have a centralized entity that controls the platform, so it has been decided to develop a decentralized application, which grants control of the platform to the network, thus avoiding both the presence of intermediaries that contribute no value and possible situations of coercion or corruption. The *Ethereum* protocol allows to develop an application of this type.

*Ethereum* is a public network that uses the *blockchain* protocol to function. In order to dispense with a central entity that validates the state of the system, the *blockchain* protocol combines a state transition system based on the digital signature, with a consensus algorithm that guarantees that all participants agree with the transactions that are carried out. It is a functionality that ensures the validity of the order of transactions (that is, the chronology of the states) using the so-called "Proof of Work".

Once there are several transactions validated by the network, these will be added to a block that will be placed behind the last added block, pointing it and forming a new, slightly longer chain, which will be sent immediately to all the nodes in the network. The information is, thus, distributed on the network, so it is practically impossible to attack the updates that the network has accepted when the data is being continuously replicated in a multitude of physical places, at the same time. In the same way, the information is not updated, replacing the immediately previous one, rather it is added, forming a chain in which all the global states of the network and the transactions that have been sent are traceable.

The members of the network are the so-called nodes, a role that can be played by those who wish to participate in the network and possess a minimum computational capacity. Of note among these nodes are the "mining nodes": machines that constantly compete against each other to validate the veracity and chronology of transactions, seeking to solve the mathematical problem posed by the Proof of Work, that is a result of the encryption protecting transactions. The consensus algorithm, in order to add a new block, verifies that the previous block exists and is valid, that it is really chronologically prior to the one it is intended to add, and that some mining node has given solution to the Proof of Work (effort for which they are rewarded, which is an incentive) and that this is correct. The *raison d'être* of the Proof of Work consists in that, when incorporating a computational cost of each block of transactions into the chain, it prevents an attacker from remaking the chain at his whim. Thus, decentralized trust in the network is guaranteed.

Having understood its operation and its advantages, we will deploy a private network of type *Ethereum* on our own node, creating the first block of a chain to which

the following ones will be added as transactions are sent. *Ethereum* is the protocol that allows the development of decentralized applications, since it combines the logic of the block-chain with the infinite possibilities offered by the so-called *Smart Contracts*. These are codes that, once they are written and sent to a decentralized network address, cannot be altered, and they automate processes that usually require the validation of a third party.

A *Smart Contract* receives input information, processes it according to the rules established in the contract, and performs the actions that it marks. Contracts are a type of those known as accounts: *Ethereum* objects that are identified by the address they occupy in the network. Contracts are accounts that have their own code and are, in turn, controlled by code. They can have funds (the currency in *Ethereum* is called *ether*) that will be controlled only by their own internal code. In this way, the "global state" in *Ethereum* is nothing else than the ledger of accounts at each moment, its balance of funds and its code. The *blockchain* protocol, on the one hand, guarantees that this state is legitimate, and, on the other hand, it allows the analysis of all the historical transactions of the network.

The architecture of a decentralized application consists, on the one hand, of code deployed in the network through *Smart Contracts*, and, on the other hand, an interface that facilitates access to them by users. As there are many ways to interact with *Ethereum* accounts and contract code methods, it has been decided to develop an intuitive mobile application to access the platform.

Thus, once the private network has been configured and deployed with a new block chain on which the information and changes that occur on the platform are going to be added, four *Smart Contracts* have been developed and contain all the logic that is required by the decentralized application. The first one proves registration in the platform of a new user and manages their journeys (eventually it will be associated to the *wallet* of the individual and this to their digital identity), the second does the same with the insurance company and the third one is deployed at the time of the request for a new journey by a client and contains the conditions for the transfer of funds in one or another direction. The latter acts as a ledger of the application to facilitate the identification of data.

The application has been developed in *Xamarin's IDE* since the so-called *Nethereum* library brings the *Web3* library closer to a project in *.NET*. It offers an interface that facilitates communication with the node, enabling the deployment of intelligent contracts, access to calls to its functions and the sending of transactions ...



Figure 1: Request of a new journey

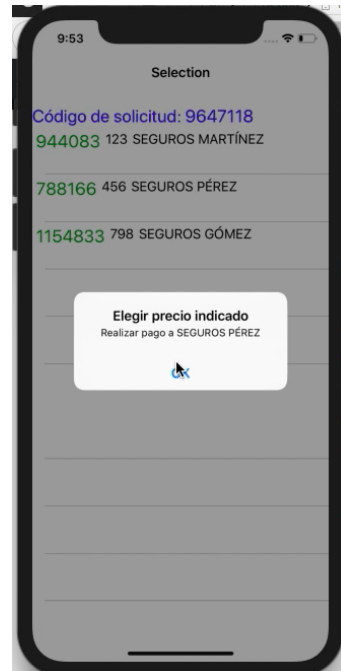


Figure 2: Company and price choice

The mobile application has been developed so that, in the first screen, it is possible to choose between proceeding to the signing up in the platform, if it has not been previously done, or the login page of the application with the ID and the password indicated then. If the *sign-up* page is accessed, all the corresponding data must be entered. The application requests the name, surname, ID, age, number of years with driver's license, address, country and password the user will use to login in future occasions. If the user wants to request a route, a map with two drag markers that indicate the departure and arrival points is displayed, a button that allows access to a drop-down to choose the type of coverage and, finally, a button to confirm the journey to advance to the next screen. It shows a list of all the companies in the platform and the price they offer, so that the user may select one and activate the insurance contract. Each company has its own weight coefficients for each parameter that intervenes in the price calculation equation (distance, type of coverage, age, number of years with the license) and could eventually change them for future contracts, in order for them to improve their profit margins.

The result is an application independent from a centralized entity, with special security in regard to the storage of information, with transparency and traceability in its operations and that offers the automation of a process that normally requires review by individuals. Thus, the objectives of free competition, continuous operation of the application, greater efficiency by eliminating intermediaries and guaranty of the integrity of information have all been achieved.



## *Índice del Proyecto*

<i>Parte I</i>	<i>Memoria.....</i>	<i>11</i>
<i>Capítulo 1</i>	<i>Introducción.....</i>	<i>11</i>
<i>Capítulo 2</i>	<i>Descripción de las tecnologías .....</i>	<i>13</i>
2.1	<i>La tecnología blockchain: una capa de datos común y segura.....</i>	<i>13</i>
2.1.1	<i>Entendiendo el Bitcoin para entender cualquier red blockchain.....</i>	<i>15</i>
2.1.2	<i>Funcionamiento de las transacciones. Soporte de la red: sistema de recompensas.....</i>	<i>17</i>
2.1.3	<i>No todo es criptomoneda: Ethereum.....</i>	<i>21</i>
2.1.4	<i>Redes públicas y redes privadas.....</i>	<i>23</i>
2.1.5	<i>Ventajas de la incorporación del protocolo Ethereum en una aplicación.....</i>	<i>24</i>
2.2	<i>Desarrollo de una DApp.....</i>	<i>26</i>
2.2.1	<i>Razón de ser de los Smart Contracts.....</i>	<i>27</i>
2.2.2	<i>Especificaciones técnicas.....</i>	<i>28</i>
<i>Capítulo 3</i>	<i>Estado de la cuestión .....</i>	<i>31</i>
3.1	<i>Blockchain y Seguros del automóvil.....</i>	<i>29</i>
3.2	<i>Otras DApps.....</i>	<i>31</i>
<i>Capítulo 4</i>	<i>Definición del Trabajo.....</i>	<i>35</i>
4.1	<i>Justificación.....</i>	<i>35</i>
4.2	<i>Objetivos.....</i>	<i>36</i>
<i>Capítulo 5</i>	<i>Desarrollo de una aplicación móvil sobre una red privada de Ethereum.....</i>	<i>39</i>



**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

<b>5.1</b>	<b><i>Análisis del sistema: desarrollo sobre Ethereum</i></b> .....	<b>40</b>
5.1.1	<i>Configuración de un cliente Geth</i> .....	41
5.1.2	<i>Conexión a la red privada. Interacción con comandos de Geth</i> .....	43
5.1.3	<i>Mejor acceso a las funcionalidades: Web3 y el servidor JSON-RPC</i> ...44	
<b>5.2</b>	<b><i>Arquitectura y diseño de la DApp</i></b> .....	<b>45</b>
5.2.1	<i>Elección de IDE: Xamarin y Nethereum</i> .....	45
5.2.2	<i>Definición de los roles en la aplicación descentralizada</i> .....	46
5.2.3	<i>Concepción de los contratos inteligentes: el contrato “maestro”</i> .....	48
<b>5.3</b>	<b><i>Desarrollo, pruebas e implementación</i></b> .....	<b>57</b>
5.3.1	<i>Desarrollo de las vistas de la aplicación</i> .....	57
5.3.2	<i>Entorno de pruebas de Remix</i> .....	63
5.3.3	<i>Integración de la red con Xamarin</i> .....	65
<b>Capítulo 6</b>	<b><i>Análisis de resultados</i></b> .....	<b>70</b>
<b>Capítulo 7</b>	<b><i>Conclusiones y trabajos futuros</i></b> .....	<b>77</b>
	<b><i>Bibliografía</i></b> .....	<b>80</b>
	<b><i>Referencias</i></b> .....	<b>81</b>
<b>Parte II</b>	<b><i>Código fuente</i></b> .....	<b>85</b>
<b>Capítulo 1</b>	<b><i>Código de ClienteUsuario.sol</i></b> .....	<b>85</b>
<b>Capítulo 1</b>	<b><i>Código de Compania.sol</i></b> .....	<b>91</b>
<b>Capítulo 1</b>	<b><i>Código de ClienteSolicitudUsuario.sol</i></b> .....	<b>96</b>
<b>Capítulo 1</b>	<b><i>Código de Report.sol</i></b> .....	<b>106</b>

## *Índice de figuras*

Ilustración 1: Transición de estados en Bitcoin [11] .....	- 18 -
Ilustración 2: Ejemplo gráfico de un fork [12] .....	- 21 -
Ilustración 3: Transición de estados en Ethereum [10] .....	- 23 -
Ilustración 4: Diagrama de la arquitectura de una DApp [36] .....	- 27 -
Ilustración 5: Conexión remota a la IP de un nodo (Paso 1) [Elaboración propia] ...	- 41 -
Ilustración 6: Conexión remota a la IP de un nodo (Paso 2) [Elaboración propia] ....	- 42 -
Ilustración 7: Conexión remota a la IP de un nodo (Paso 3) [Elaboración propia] ....	- 42 -
Ilustración 8: Conexión a un proceso geth que ya está en marcha [Elaboración propia] ..	- 43 -
Ilustración 9: Estructura de datos del contrato ClienteUsuario.sol [Elaboración propia] .	- 49 -
Ilustración 10: Constructor del contrato ClienteUsuario.sol [Elaboración propia] ....	- 49 -
Ilustración 11: Definición de un evento en ClienteUsuario.sol [Elaboración propia]	- 50 -
Ilustración 12: Función “checkPassword” del contrato ClienteUsuario.sol .....	- 50 -
Ilustración 13: Estructura de datos del contrato Compania.sol [Elaboración propia]	- 50 -
Ilustración 14: Vectores de direcciones del contrato Compania.sol [Elaboración propia]	- 50 -
Ilustración 15: Función “setPonderaciones” del contrato Compania.sol [Elaboración propia] .....	- 50 -
Ilustración 16: Estructura de datos del contrato ClienteSolicitudTrayecto.sol [Elaboración propia] .....	- 50 -
Ilustración 17: Función “setEleccion” del contrato ClienteSolicitudTrayecto.sol [Elaboración propia] .....	- 50 -
Ilustración 18: Estructuras de datos del contrato Report.sol [Elaboración propia] ....	- 50 -
Ilustración 19: Mapeos y vectores de direcciones del contrato Report.sol [Elaboración propia] .....	- 50 -
Ilustración 20: Vista del simulador de Xcode sin lanzar la aplicación [Elaboración propia] .....	- 50 -
Ilustración 21: Primera vista de la aplicación de Xamarin [Elaboración propia] .....	- 50 -

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

Ilustración 22: Vista de inicio de sesión del usuario en la aplicación [Elaboración propia]	- 50 -
Ilustración 23: Vista de registro del usuario en la aplicación [Elaboración propia]...	- 50 -
Ilustración 24: Menú desplegable para elegir el tipo de cobertura en el trayecto [Elaboración propia]	- 50 -
Ilustración 25: Vista de solicitud de un trayecto que incluye un mapa con marcadores [Elaboración propia]	- 50 -
Ilustración 26: Vista de elección de las compañías y el precio que ofrecen para al trayecto [Elaboración propia]	- 50 -
Ilustración 27: Mensaje de error que aparece al confirmar la solicitud del trayecto sin haber especificado el tipo de cobertura [Elaboración propia]	- 50 -
Ilustración 28: Mensaje de error que aparece al equivocarse el usuario en el inicio de sesión [Elaboración propia]	- 50 -
Ilustración 29: Herramientas de Remix para trabajar con contratos inteligentes [Elaboración propia]	- 50 -
Ilustración 30: Menú de compilación de contratos inteligentes en Remix [Elaboración propia]	- 50 -
Ilustración 31: “Resguardo” que se recibe en Xamarin tras el envío de una transacción [Elaboración propia]	- 50 -
Ilustración 32: Parte del código que contiene la ABI del contrato Report.sol [Elaboración propia]	- 50 -
Ilustración 33: Código necesario para el despliegue de un contrato desde Xamarin gracias a Nethereum [Elaboración propia]	- 50 -
Ilustración 34: Resguardo que se recibe en Xamarin tras el despliegue del contrato Report.sol [Elaboración propia]	- 50 -
Ilustración 35: Función de despliegue de ClienteUsuario.sol utilizando Nethereum desde Xamarin [Elaboración propia]	- 50 -
Ilustración 36: Función de llamada a una función de Compania.sol utilizando Nethereum desde Xamarin [Elaboración propia]	- 50 -
Ilustración 37: Ejemplo de modifier sencillo que comprueba que la cuenta que lance una transacción particular sea la misma que lanzó la del despliegue del contrato [Elaboración propia]	- 50 -

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

Ilustración 38: Función “pagarContrato” del contrato ClienteUsuario.sol) [Elaboración propia]..... - 50 -

Ilustración 39: Función “pagarCompany” del contrato ClienteSolicitudTrayecto.sol [Elaboración propia] ..... - 50 -

## Parte I: MEMORIA DEL PROYECTO

### 1. Introducción

Es una realidad que la revolución digital en la que se encuentran la mayoría de los mercados a día de hoy obliga a multitud de compañías de toda clase de sectores a estar en constante movimiento y a intentar reinventarse continuamente, con le objetivo de añadir valor a sus servicios. El sector de los Seguros de automóvil, sobre el que pone el foco este Proyecto, no escapa a esta corriente y también realiza inversiones importantes en la investigación y desarrollo de herramientas que utilicen lo que ofrecen las nuevas tecnologías.

Este Proyecto de Fin de Grado nació en el seno de la empresa en la que completé unas Prácticas en Empresa de cinco meses de duración. Realizadas a jornada completa entre los meses de abril a agosto de 2018, estas Prácticas se ubicaron en el departamento de Desarrollo Tecnológico de *Management Solutions*, firma española de Consultoría de Negocio.

La función de este departamento pasa por la conceptualización e implantación de soluciones tecnológicas complejas, teniendo como destino tanto el cliente como la propia firma. De esta manera, mi responsabilidad particular consistió en el desarrollo integral de una aplicación web para optimizar un proceso concreto. Para ello, hice uso, en concreto, de un entorno *ASP.NET* para la implantación de la aplicación y *SQL Server Management Studio* en lo que se refiere al acceso a los datos.

En este contexto de continuo estudio y análisis de tecnologías específicas, surge de manera tan natural como frecuente la necesidad de analizar distintas posibilidades y de revisar los avances que se van produciendo fuera de la empresa, para así contemplar su integración en las herramientas que se producen desde el departamento. De este modo, se realizan continuamente y de manera paralela al ejercicio del equipo diferentes labores de investigación.

Por otro lado, la relevante componente legal de las cuestiones que rodean mercado asegurador hace que no solo se busquen tecnologías eficientes, sino que garanticen la

confianza de la empresa en lo relacionado con la seguridad. Es por ello, entre otras razones, por lo que nos hemos interesado a la tecnología *blockchain* en particular como segundo centro de interés.

Esta tecnología, que será convenientemente explicada y analizada sustituye a los modelos tradicionales de almacenamientos centralizados de datos, ofreciendo muchas otras funcionalidades complementarias. Una de las más interesantes es la posibilidad de desplegar los llamados *Smart Contracts* (contratos inteligentes), programas que se establecen en la red descentralizada sobre los que quedan registrados las cláusulas del contrato, activando y desactivando permisos, por ejemplo, o realizando acciones de manera automática, conforme convenga. Esto permite eliminar participantes innecesarios del proceso de establecimiento y seguimiento de un contrato de seguros, el cual puede llevarse a cabo de manera automática en muchos aspectos, reduciendo los costes correspondientes.

Al realizar una operación de intercambio de datos y de “firma” de un contrato de un seguro dependemos de un intermediario que valida la operación. La tecnología *blockchain* permite transmitir y guardar esta información de forma automática. Con el desarrollo del contrato inteligente, las “reglas del juego” quedan registradas garantizando la confianza de todas las partes en un sistema de almacenamiento del que cada parte guarda en todo momento una copia idéntica.

Se trata de un tipo de tecnología de registro distribuido y que la firma en la que realizo las prácticas quiere conocer bien y controlar de cara a ofrecer en el futuro próximo servicios integrales basados en esta herramienta, debido a la multitud de ventajas que propone. La realización de proyectos de investigación y de experimentación de este tipo ofrece la oportunidad de avanzar junto a la firma en el aprendizaje de una tecnología interesante y disruptiva.

## 2. Descripción de las Tecnologías

### 2.1 La tecnología *blockchain*: una capa de datos común y segura

A grandes rasgos, se trata de una tecnología de almacenamiento y de transmisión de información, que no depende de un organismo central que la controle, que guarda estos datos de manera encriptada e indeleble, de modo que no se puedan manipular ni eliminar, y que cuenta con una serie de ventajas en materia de seguridad y transparencia, frente a las tecnologías tradicionales de almacenamiento de datos.

*Blockchain* es un tipo de tecnología de registro distribuido. Estas tecnologías consisten en sistemas de almacenamiento de datos que cuentan con la peculiaridad de albergar varios participantes y de no depender de un organismo central que medie y controle las acciones ni cuente con autoridad para realizar cambios unilateralmente. Tradicionalmente, la razón de ser de estos organismos tiene que ver con la falta de confianza en el sistema, haciendo necesaria la existencia de una entidad que regule la validez de las transacciones que se llevan a cabo. De este modo, los miembros del sistema depositan su confianza en una única entidad central.

Con *blockchain*, el registro “solo puede ser actualizado a partir del consenso de la mayoría de los participantes del sistema y, una vez introducida, la información no puede ser borrada”. [1] La cadena de bloques es “una base de datos transaccional globalmente compartida”. [2]

Como se explica en un interesante artículo, en su forma más simple, un registro contable distribuido es una “base de datos sostenida y actualizada de manera independiente por parte de cada uno de los participantes en una amplia red”. No se trata de un sistema en el que una autoridad comunica a los participantes los cambios que se van realizando en tiempo real, sino que las actualizaciones son “construidas y sostenidas por cada uno de estos nodos de manera independiente”. De este modo, cada participante de la red “procesa cada una de las transacciones, llegando a sus propias conclusiones y

votando sobre esas conclusiones para asegurar que la mayoría está de acuerdo con ellas”. Es lo que se conoce como consenso. [3]

Este es necesario debido a que, al tratarse de una red distribuida, “diferentes nodos pueden estar haciendo operaciones simultáneas que deben propagarse por la red y en ocasiones colisionan entre sí, se establece un sistema claro para determinar qué transacciones son válidas y se incorporan a la cadena y cómo se resuelven los conflictos y colisiones.” [4] Mientras que una transacción que has enviado es procesada, ninguna otra puede modificarla.

Una vez existe este consenso, “el registro distribuido ha sido actualizado, y todos los participantes mantienen su propia copia idéntica del registro.” Esto lo convierte en un sistema de registro que va más allá de una base de datos tradicional, aumentando su seguridad e interoperabilidad”. [4] Al estar todos los nodos participando de la validación de las transacciones, podemos hablar de una red *peer-to-peer*.

“Una red *peer-to-peer*, red de pares, red entre iguales, red entre pares o red punto a punto (*P2P*, por sus siglas en inglés) es una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes *P2P* permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados”. [5]

En muchas ocasiones, para explicar conceptos relacionados con la tecnología *blockchain*, se hace una analogía con un registro de contabilidad. Como se explica desde Azure, el servicio de creación de aplicaciones en la nube de Microsoft: “La tecnología consiste en un libro de contabilidad seguro y compartido de las transacciones distribuidas entre una red de PC, en lugar de tenerlas un solo proveedor“. [6] Es decir, *blockchain* es una red para gestionar un libro de cuentas único descentralizado y replicado en cada uno de los nodos que componen la red (en el ámbito financiero esto sería el libro mayor de contabilidad donde se anotan todos los movimientos).

De cara a entender la tecnología de registro contable distribuido basada en la cadena de bloques, es necesario comprender el funcionamiento de *Bitcoin*, primera plataforma de *DLT*. Aunque *blockchain* fue conceptualizado con la llegada de *Bitcoin*, desde entonces la tecnología “ha sido abstraída para referir a cualquier tipo de tecnología



de base de datos distribuida que almacena los datos en bloques continuamente encadenados mediante hash”. [7]

### 2.1.1 Entendiendo el *Bitcoin* para entender cualquier red *blockchain*

Se trata de una tecnología que nace como respuesta al creciente interés por parte de un gran número de colectivos en la eliminación de intermediarios en diferentes procesos. Existen muchos casos en los que un organismo central realiza una serie de tareas de mediación y de soporte de un sistema, pero bien podría ser sustituido por una red descentralizada de computadoras que asumiera esas funciones, abriendo la posibilidad de prescindir de estos intermediarios.

El ejemplo primero cuando hablamos de casos de uso de una red descentralizada de *blockchain* es, sin duda, el mundo de las criptomonedas. Sobre esta tecnología se desarrolló la primera de este tipo, el *Bitcoin*, llegando a poner en jaque a bancos y grandes corporaciones.

El euro, el dólar, el yen... son monedas fiduciarias, que “representan un valor que intrínsecamente no tienen” y que, por ello, dependen “del crédito y confianza que merezcan” [8]. Esta confianza la depositamos los usuarios, al fin y al cabo, sobre los países que emiten la moneda y, por extensión, sobre los organismos que gestionan las monedas y sus movimientos.

En este caso, estos organismos son los bancos y están centralizados, es decir, agrupan los recursos del sistema, los cuales son explotados por otros elementos de este sistema. El objetivo final de los creadores de *Bitcoin* era sacar a la luz un nuevo sistema de efectivo que fuera completamente *P2P* (de igual a igual), sin un tercero de confianza.

El dinero es, al fin y al cabo, una manera de registrar quien tiene que y quien debe a quien (un sistema de contabilidad). Pero se necesita una tercera parte en la que se confie para validar que el dinero es verdadero.

*Bitcoin* es exactamente eso: es un sistema de contabilidad. Una manera de registrar valores, transacciones... digitalmente, de modo que se pueda hacer directamente entre

usuarios, y todo quede registrado en un libro mayor de cuentas (*open ledger*). Al monitorizar y actualizar ese libro mayor de un modo colectivo y consensuado, se puede eliminar la necesidad de una tercera entidad (bancos, gobiernos) que sea el depositario de toda esa información. Esto pondría fin a la ineficiencia, a las comisiones o incluso a la corrupción, eventualmente (y todos los riesgos que vienen con el hecho de centralizar la información de ese modo).

*Bitcoin* coge la función de la entidad central y la automatiza, colocando la información en un libro mayor que está en línea, disponible para todo el mundo. Como ejemplo, uno se puede imaginar “una tabla que lista los balances de todas las cuentas en una divisa electrónica. Si se solicita una transferencia de una cuenta a otra, la naturaleza transaccional de la base de datos garantiza que la cantidad que es sustraída de una cuenta es añadida en la otra. Si por la razón que sea, no es posible añadir la cantidad a la cuenta de destino, la cuenta origen tampoco se modifica”. [2]

#### 2.1.1.1 Necesidad de un tercero fiduciario: el problema del “doble gasto” y su solución

Se habían desarrollado con anterioridad ideas de sistemas de moneda electrónica que trabajaran de igual a igual. Esto era posible gracias a las firmas digitales. Esto es, “cada usuario tiene una clave criptográfica privada asociada a otra clave pública. La clave privada es la que contiene toda la información sobre el usuario y garantiza su identidad, mientras que la clave pública solo muestra lo que el usuario desea que los demás puedan ver”. [9] De esta manera, en el momento en que un usuario desea enviar valor a otro, el primero necesita “acreditar que tiene en su poder la clave privada para demostrar que es quien dice ser y firmar con ella la transacción, mientras que para recibir dinero basta con proporcionar la clave pública”. [9]

Aunque gracias a las firmas digitales se consigue un sistema en el que la cadena de posesión de cada moneda es trazable y validable, no se trataba de sistemas realmente *P2P* porque seguía haciendo falta la necesidad de una institución financiera que diera solución a una eventual situación de “doble gasto”: ¿qué ocurre si dos transacciones existentes en la red quieren efectuar una gestión sobre una cuenta, como, por ejemplo, borrarla? ¿qué ocurre si dos transacciones quieren gastar la misma instancia de moneda? En otras palabras, aunque se pueda demostrar que la moneda que estoy recibiendo te

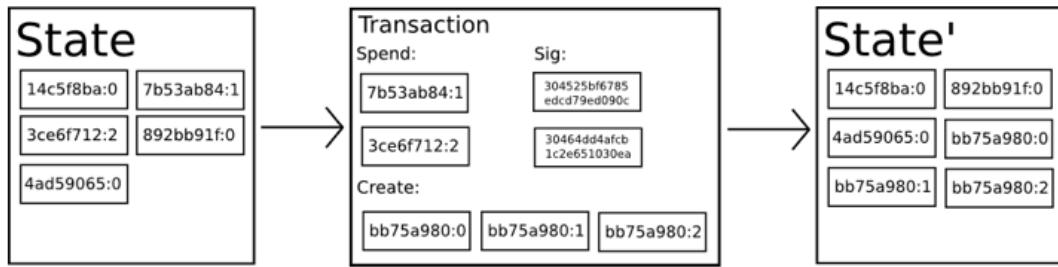
pertenecía antes, necesito una verificación de que no la has gastado ya dándosela a otra persona.

Shatoshi Nakamoto solucionó este problema incluyendo en el protocolo una funcionalidad que garantiza la validez del orden de las transacciones mediante el algoritmo de consenso conocido como Prueba de Trabajo. Una vez haya varias transacciones validadas, estas serán añadidas a un bloque que se colocará detrás del último bloque añadido, formando una nueva cadena, un poco más larga, que se enviará inmediatamente a todos los nodos de la red. “Si dos transacciones se contradicen, la que concluye en segundo lugar será rechazada y no formará parte del bloque”. [2] Este “mecanismo de selección de orden” se conoce como minería y funciona de modo que los bloques son incorporados a la *blockchain* en intervalos regulares.

### 2.1.2 Funcionamiento de las transacciones. Soporte de la red: sistema de recompensas.

El dinero es una manera de registrar quien tiene que y quien debe a quien (un sistema de contabilidad). Pero se necesita una tercera parte en la que se confíe para validar que el dinero es verdadero, que no hay fraude.

Como se explica en el White Paper de *Ethereum*, se puede entender el registro de efectivo de una moneda como un sistema de transición de estados. Básicamente, “en un sistema estándar de un banco, el estado es el balance, una transacción es una petición para mover X de A a B, y la función de transición de estados reduce el valor en la cuenta de A por valor de X y aumenta el valor de la cuenta de B en X. Si la cuenta de A originalmente tiene menos de X, la función de transición de estados devuelve un error”. [11]



*Ilustración 1: Transición de estados en Bitcoin [11]*

“El “estado” en Bitcoin, es la colección de todas las monedas que han sido minadas y no gastadas todavía (se conoce como resultados de transacciones no gastadas, o UTXO). Toda UTXO tiene una denominación y un poseedor asociados mientras que una transacción contiene una más entrada, con cada entrada conteniendo una referencia a una UTXO existente y una firma criptográfica producida por la clave privada asociada a la dirección del poseedor.

Dese modo, la función de transición de estado puede ser definida de la siguiente manera:

- Para cada entrada a la transacción, si la referencia a la UTXO no existe en el conjunto de monedas previo a la transacción, se devuelve un error lo mismo ocurre si la firma ofrecida no coincide con el dueño de la UTXO. (esto garantiza que quien envía la transacción no gaste monedas que no existen, ni monedas que no le pertenezcan).
- Si la suma de las denominaciones de todas las UTXO de entrada es menor que la suma de las denominaciones de todas las UTXO de salida, se devuelve un error. (esto garantiza la conservación del valor).
- Si todo va bien, se devuelve el siguiente “estado” del sistema eliminando de este las UTXO de entrada y añadiendo las UTXO de salida”. [11]

Un tercero centralizado sobre el que depositáramos nuestra confianza garantizaría en todo momento la validez del “estado” del sistema. Para poder prescindir de él, Bitcoin combina el sistema de transición de estados explicado con un sistema de consenso de modo que todo el mundo esté de acuerdo con las transacciones que se realizan.

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

Los miembros de la red son los llamados nodos, rol que puede desempeñar quien desee participar de la red y posea una mínima capacidad computacional. Destacan de entre estos nodos, los nodos mineros: máquinas que trabajan todos los días, el día completo, para validar la veracidad y cronología de las transacciones. Esto es, intentan encontrar la solución a la Prueba de Trabajo que se propone para cada bloque de transacciones que necesita de la validación de la red.

Para hacer minería, los mineros “comprueban la consistencia de la solicitud en la cadena de bloques a la cual pertenece, para evitar el gasto duplicado de quien dice ser dueño de tal o cual número de monedas, y descriptan el hash, para constatar la veracidad de la información”. [12] Para ello, básicamente, se encargan de buscar una fórmula matemática llamada *hash*, y que irá asociada al bloque que se va a añadir a la cadena. “Los *hashes* son claves informáticas que sintetizan mucha información en muy pocos caracteres. Cada bloque cuenta con un *hash* nuevo y con el *hash* del bloque inmediatamente anterior”. [13]

Cuando un minero ha encontrado la dirección *hash* válida, se procede a añadirse a la cadena. Añadir un bloque a la cadena, a fin de cuentas, quiere decir cambiar el estado de la red a uno nuevo. El algoritmo de consenso, que gestiona la transición de estados de la red descentralizada (hace las veces, como hemos visto, de entidad centralizada), comprueba que el bloque anterior existe y es válido, que la marca de tiempo que tiene es menor que la del que se pretende añadir, y que algún minero ha encontrado la Prueba de Trabajo y que esta es correcta.

Este ejercicio requiere una gran potencia computacional, por lo que el minero que completa la Prueba de Trabajo es recompensado. Se trata de un problema matemático que solo puede ser resuelto mediante fuerza bruta, lo que, en esencia, obliga a realizar un importante número de intentos. La razón de ser de la Prueba de Trabajo es que, al conllevar la incorporación de cada bloque de transacciones a la cadena un coste computacional, impide que un atacante pueda rehacer la cadena a su antojo.

Un atacante a la red tendría como objetivo el orden de las transacciones, ya que es el único elemento del sistema que no está encriptado. En el *White Paper* de *Ethereum* se relata un ejemplo que da explicación a la existencia de la Prueba de Trabajo. Un atacante envía 100 BTC a cambio de un producto, espera a recibirlo (se entiende que es

un activo digital y por tanto instantáneo) y rápidamente se envía a sí mismo esos 100 BTC, intentando convencer a la red de que el segundo envío tuvo lugar cronológicamente antes que el primero. Cuando se envía la primera transacción para que sea validada, tardará unos minutos en añadirse a un bloque que se incluirá en la red. Recordemos que, aunque se hayan añadido, pongamos, una hora después, unos cinco bloques más a la cadena, todos apuntan de manera indefinida (mediante los *hashes*) a la transacción que nos interesa y de ese modo están confirmándola. Habiendo recibido el bien, el atacante intenta enviarse 100 BTC (que no tiene) a sí mismo. Esto va a dar error porque el algoritmo de consenso detecta que se está intentando utilizar una moneda que no existe ya en el estado actual.

En ese momento, el atacante crea un *fork* de la cadena de bloques, intentando minar otra versión del bloque que albergó la transacción de hace una hora, apuntando al bloque que le precedía. Como los datos que alberga este bloque que se intenta minar son distintos al original, el algoritmo requiere la realización de la Prueba de Trabajo. El bloque que se está intentando minar, por supuesto, tiene un *hash* distinto al original, por lo que la cadena sigue ampliándose por el otro camino, con direcciones *hash* legítimas que se apuntan entre sí. Como en caso de producirse un *fork* la regla del protocolo marca que la cadena válida es la más larga, los mineros legítimos trabajan en esta última, mientras que el minero atacante trabaja en la que se está quedando atrás. Para conseguir que *su* cadena fuera la más larga, debería tener más poder computacional que el resto de la red para sobreponerse (el 51% del poder). Un minero puede decidir sobre qué cadena trabajar en cada momento, pero al encontrarse todos los mineros de la red intentando añadir bloques (que se añaden en periodos regulares) a la cadena más larga, puede que se diera que el atacante tuviera suerte y añadiera algún bloque más a esa cadena (poco probable), pero, a efectos prácticos, jamás podría competir con la mayoría de los miembros de la red compitiendo fuertemente por que continúe la cadena larga, verdadera, y conseguir superarla, sin tener, al menos, el 51% del poder computacional de la red.

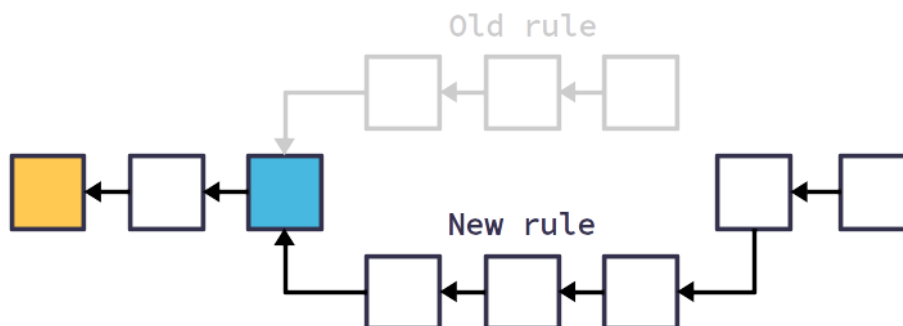


Ilustración 2: Ejemplo gráfico de un fork [12]

Todas y cada una de las transacciones que se realizan quedan registradas en la cadena de bloques, para siempre, y no se pueden borrar. Así, es posible para cualquier usuario de la red trazar todo el recorrido que ha realizado un *Bitcoin* cualquiera, observar cómo ha ido cambiando de manos a lo largo del tiempo. Sin embargo, los usuarios (las cuentas) están encriptados, de modo que no se sabe a quién corresponde la dirección registrada, garantizando su privacidad.

### 2.1.3 No todo es criptomoneda: *Ethereum*

Aunque comúnmente lo más conocido de todo lo nuevo que trajo *Bitcoin* es el hecho de disponer de una moneda digital (y, por tanto, sin valor intrínseco) y sin depender de un controlador, muchos fijaron su atención en el sistema que se encontraba detrás de todo esto: la tecnología de cadena de bloques como herramienta de consenso distribuido.

Vitalik Buterin, con su plataforma *Ethereum*, fue el que más ahondó en esta funcionalidad ofrecida por *Bitcoin*, observando que podía haber más potencial que únicamente el mercado de las criptomonedas. Se suele hacer la analogía de que los primeros protocolos *blockchain*, como *Bitcoin*, funcionaban como una calculadora, eran plataformas dedicadas a albergar una única aplicación, mientras que el protocolo *Ethereum* sería más bien un *Smartphone* con un sistema operativo con la capacidad de albergar infinidad de aplicaciones y de todo tipo, sin límites de casos de uso.

De este modo, como se explica en su *White Paper*, *Ethereum* “busca ofrecer una cadena de bloques con un lenguaje de programación incorporado a esta que pueda ser

utilizado para codificar complejas funciones [...] en unas pocas líneas de código”, permitiendo a los usuarios desarrollar todo tipo de funcionalidades, como registros de propiedad, instrumentos financieros...

La red *Ethereum* está en funcionamiento gracias a ordenadores que se encuentran en todo el mundo. Para recompensar el coste computacional de procesar los contratos y asegurar la red, hay una recompensa que se le da al ordenador que fue capaz de añadir el último bloque a la cadena. Más o menos cada quince segundos, un nuevo bloque es añadido a la cadena de bloques con las últimas transacciones procesadas por la red y el ordenador que genera el bloque será recompensado con 3 *ether*. Debido a la naturaleza de algoritmo de generación de bloques, este proceso (que genera una prueba de trabajo) está garantizado que será aleatorio. Por otro lado, las recompensas se entregan en proporción al poder computacional de cada máquina.

*Ether* es “un elemento necesario —el gas— para operar la plataforma distribuida *Ethereum*. Es una forma de pago hecha por los clientes de la plataforma a la máquina que ejecuta las operaciones solicitadas”. [14]

En la red *Ethereum* existen dos tipos de cuentas, tratadas de la misma manera por la *EVM*:

- Cuentas externas: cuentas controladas por una clave privada, que pueden tener *ether*. Si uno posee la clave privada asociada a esta cuenta, tienes la capacidad de enviar ether y mensajes desde esta.
- Contratos: cuentas que tienen código propio y que están, a su vez, controladas por código. Pueden tener *ether* que será controlado únicamente por su propio código interno.

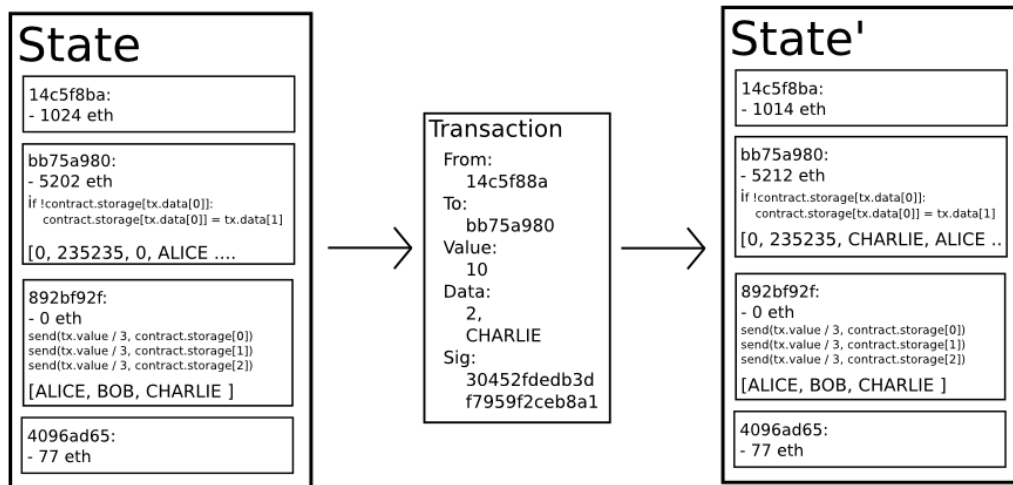
Del mismo modo, existen dos tipos de transacciones, teniendo ambas un coste de *ether* a la red, por parte de la cuenta externa que envía la transacción:

- Transferencias de *ether* (todas tienen una dirección que recibe el *ether* que se envía, excepto las transacciones de despliegue de contratos).
- Comunicaciones con *Smart Contracts*.

El “estado global” que se explicaba en *Bitcoin* es algo más complejo en *Ethereum*. Funciona con un mapeo de los objetos de tipo “cuenta de *Ethereum*” mediante una



dirección. Como ocurría con *Bitcoin*, las cuentas externas poseen un identificador y un balance de *ether*, mientras que los objetos de *Smart Contracts*, además, contienen código y espacio de almacenamiento interno. De este modo, el estado global es la lista de todas estas cuentas, para cada momento.



*Ilustración 3: Transición de estados en Ethereum [10]*

el momento de validarse la transacción. De esta manera, si la transacción es añadida a la cadena en el bloque A, todos los nodos que descargan y validan el bloque A ejecutarán el código.

### 2.1.4 Redes públicas y redes privadas

La red *Ethereum* es única, global, de libre acceso y está sostenida por la *EMV*. Sin embargo, su protocolo es accesible y descargable, ofreciendo la posibilidad de desplegar una red privada en una máquina particular, para hacer pruebas del código que se desea enviar eventualmente a *Ethereum*, sin enviarlo desde el primer momento, debido a todo lo que ello supone.

“Los *blockchains* privados son controlados por una única organización que determina quién puede leerlos, presentar transacciones en él y participar en el proceso de consenso. Dado que están 100% centralizados, los *blockchains* privados son útiles como entornos de prueba, pero no para producción efectiva”. [15]

Como hemos visto, cada transacción que solicitamos que sea añadida a la cadena de bloques supone un coste de gas que hay que pagar con nuestro *ether*. Si se es nuevo en el terreno de los contratos inteligentes y los protocolos *blockchain*, parece claro que la creación de una red privada, con nodos privados, y el despliegue de una cadena de bloques nueva, es una posibilidad más que interesante. Esto permitiría realizar todas las pruebas que fueran necesarias de manera gratuita, sin estar gastando *ether* real en todo momento.

### 2.1.5 Ventajas de la incorporación del protocolo *Ethereum* en una aplicación

La inclusión del protocolo *Ethereum* en nuestra aplicación (convirtiéndola, así, en una *DApp*), a priori, traerá consigo una serie de ventajas:

- Independencia de una entidad centralizada en la que los miembros de la red deban confiar. La validación de las transacciones y del orden en que son tratadas, gracias al algoritmo de consenso que utiliza *Ethereum*, permitiría que pudiéramos estar hablando de una verdadera red entre iguales (red *P2P*).
- Seguridad en el almacenamiento de la información. Al contrario que *Dropbox*, *Amazon* o *Google Drive*, que son, al fin y al cabo, servicios de almacenamiento en la nube centralizados, la aplicación de la tecnología *blockchain* ofrece la posibilidad de almacenar la información y replicarla inmediatamente en todos los nodos de la red. Esto prácticamente imposibilita el que un hacker ataque la red, o que se pierda información a causa de problemas ajenos a la red (catástrofes naturales, problemas técnicos...), pues habría que atacar al mismo tiempo a todos los nodos de la red.
- Por otro lado, “en la certificación de procesos y garantía de integridad de la información, la transparencia e inmutabilidad nativas de *blockchain*

permiten conocer el origen y propiedad de la información, garantizando la integridad de esta “. [16] Como se verá más adelante, aquí juegan un papel muy importante los llamados eventos de la *EVM*.

- Automatización de los procesos. Las condiciones de inmutables, a la par que transparentes, que tienen los contratos inteligentes, permiten que los usuarios de una aplicación concreta sostenida en *Ethereum* puedan consultar sus “reglas del juego” particulares y accedan libremente a acatarlas, si lo desean.
  
- Rapidez en las transacciones de dinero, al no depender del procesamiento por bancos. Esto es especialmente interesante si estamos hablando de compañías domiciliadas en el extranjero.

Estas razones han conducido a que este Proyecto se embarque en el desarrollo de una aplicación que ponga en contacto clientes y aseguradoras, depositando la lógica en los contratos inteligentes para obtener todos los beneficios que ofrecen las tecnologías de registro distribuido y el protocolo de la cadena de bloques.

## 2.2 Desarrollo de una *DApp*

Como se ha podido explicar, la iniciativa de realizar el Proyecto de aplicación móvil gracias a la ayuda de la tecnología *blockchain* ha respondido a la necesidad de plantear una estructura nueva que eliminara una serie de elementos que podían ser mejorados, como es la rapidez en el servicio, la automatización de las gestiones o la seguridad. Al decidir que lo adecuado es colocar la lógica en manos de los contratos inteligentes haciendo uso de una red de privada de *Ethereum*, lo que estamos haciendo, en realidad, es construir lo que es conocido como aplicación descentralizada.

Una aplicación descentralizada, *Dapp* o *DApp* es una aplicación que sirve a una finalidad específica para sus usuarios, pero que tiene una importante propiedad: la aplicación en sí no depende de ninguna entidad existente. Más que servir como *front-end* para vender u ofrecer los servicios de una entidad específica, una *DApp* es una herramienta para personas y organizaciones, a ambos lados de una interacción utilizada para poder juntarse sin depender de un intermediario centralizado. De esta manera, “incluso funciones intermediarias de las que, típicamente, son responsables las entidades centralizadas, como el filtraje, gestión de identidad y tratamiento de disputas, son tratados directamente por la red”. [17] Esto es posible gracias a la colocación de la lógica necesaria para el funcionamiento de la aplicación en manos de los contratos inteligentes, o *Smart Contracts*.

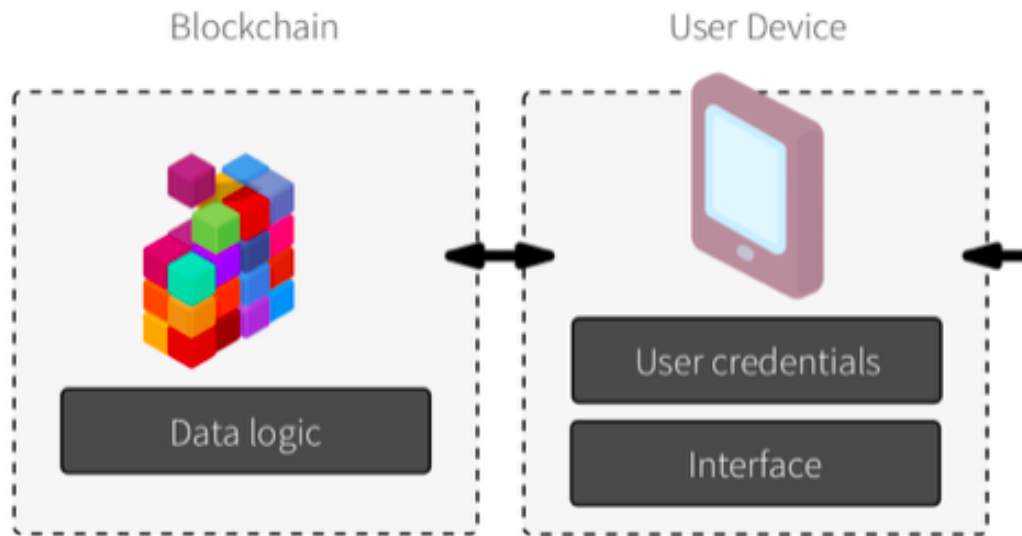


Ilustración 4: Diagrama de la arquitectura de una DApp [36]

Lo importante que hay que tener claro de los contratos inteligentes es que, al contar con el protocolo *Ethereum*, que utiliza la lógica de consenso de la cadena de bloques, un usuario que haga uso de una aplicación descentralizada sobre *Ethereum* puede estar seguro de que no se adultera la salida en el momento de ejecutarse el programa: para cierta entrada va a generarse cierta salida.

De esta manera, una *DApp* se puede definir como la combinación de uno o varios contratos inteligentes y una interfaz gráfica para utilizar dichos contratos

### 2.2.1 Razón de ser de los *Smart Contracts*

“Vitalik consideró que el lenguaje bajo el que opera la red *Bitcoin* llamado *Bitcoin Transaction Language* (BTL) era insuficiente para evolucionar dicha red”. [18] Por ello, introdujo el lenguaje de programación *Solidity*, que es sobre el que se escriben los contratos inteligentes que desarrolló.

Los contratos son códigos que una vez se escriben y se envían a una dirección de la red descentralizada no se pueden alterar, y que automatizan procesos que suelen

requerir la validación de una tercera parte. Muchas veces se hace una analogía con una máquina expendedora. “Es un contrato en el cual intercambiamos dinero en efectivo (1€) por un bien (refresco). La máquina se encarga de comprobar la validez de los fondos introducidos, siendo no un humano, sino software el que se encarga de verificar el proceso, y es el mismo quien valida o cancela la transacción. A pesar de esto, es posible que la máquina falle, y debido a que el producto se quede enganchado, no recibamos ni el bien ni nuestro dinero”. [19]

Como ocurría con *Bitcoin*, al final, se trata de niveles de confianza. “Si queremos llevar a cabo la transacción con un familiar o alguien conocido, en quien confiamos, no necesitaremos emplear este tipo de intermediarios de confianza, le daríamos directamente 1€ por el refresco”. [19]

Así, el objetivo de los contratos inteligentes es “simplificar en gran medida el número de procesos y factores que intervienen en una transacción, ya que es el protocolo “el que verifica y hace cumplir el contrato a ambas partes, sin siquiera requerir de una cláusula contractual firmada por las mismas”. [19]

De este modo, los *Smart Contracts* son, al fin y al cabo, “contratos capaces de auto ejecutarse una vez que se cumplen las condiciones fijadas en su programación”. [1] Ofrecen, así, “la capacidad para confiar en una red distribuida la confirmación de que un contrato de cualquier tipo ha sido cumplido sin revelar ningún tipo de información confidencial sobre las partes y/o naturaleza de la transacción”. [1]

## 2.2.2 Especificaciones técnicas

Los contratos inteligentes y su lenguaje, *Solidity*, son bastante sencillos de comprender, teniendo pocas herramientas y siendo estas muy intuitivas. Ofrecen la posibilidad de albergar variables de contrato, estructuras de datos y código de funciones que pueden ser ejecutadas tanto interna como externamente.

Las variables del contrato se pueden entender como “una parte única en una base de datos que puede ser consultada o modificada llamando a funciones del código que gestiona dicha base de datos”. [2]

#### 2.2.2.1 Mapping

Una de las prácticas más comunes es identificar estructuras de datos del contrato con la dirección en la que se encuentra ubicado este en la red. Esto es posible gracias a la definición de un mapeo.

#### 2.2.2.2 Constructor

Se trata de la función que se ejecuta en el momento del despliegue de la instancia de contrato a la red. Gran parte de la lógica del contrato suele recaer sobre el constructor. Debe llevar el mismo nombre que el propio contrato inteligente.

#### 2.2.2.3 Llamadas y transacciones

En cuanto a la interacción entre contratos inteligentes, se produce mediante el envío de mensajes, que pueden llevar dentro de sí datos y *ether*. Cuando un contrato recibe este mensaje, tiene la opción de inmediatamente devolver datos. Es, a todos los efectos, una llamada a una función.

En cuanto a las transacciones, como suponen un coste computacional al que van a hacer frente los mineros de la red, en el momento del envío se debe precisar la cantidad de cuota (conocida como gas) que se desea pagar por unidad de la cantidad que la propia transacción se asigna que va a costar. Si se indica un precio muy bajo es posible que los mineros rechacen procesar la transacción y no será añadida a la cadena de bloques.

Si la cuenta que recibe la transacción contiene código (cuenta de tipo *Smart Contract*), este es ejecutado.

Para desplegar un contrato inteligente, es necesario contar con el *bytecode* del mismo. Para interactuar con él, ya que es importante para la ejecución del código, se requiere de la interfaz binaria del contrato, o *ABI*.

#### 2.2.2.4 Eventos

Se trata de un tipo especial de funciones dentro de los contratos. Básicamente, el objetivo de los eventos es añadir los argumentos que reciben al *log* de la transacción. Es decir, gracias a ellos, es posible grabar lo que se desee en la inalterable cadena de bloques, suponiendo una de las herramientas más interesadas que ofrece una red *Ethereum*, pues permite customizar las informaciones que queremos que queden grabadas de forma inalterable (se pueden incluso organizar por temas).

“Con el uso de eventos es relativamente sencillo crear un “explorador de la blockchain” que monitorice las transacciones y los balances de la nueva moneda”. [2]

Para grabar eventos, estos deben ser llamados desde otras funciones en el contrato. “Los contratos no pueden acceder a los datos del log después de crearse, pero pueden ser accedidos desde fuera de la blockchain de forma eficiente”. [2]

#### 2.2.2.5 Modifiers

Ofrecen la posibilidad de ejecutarse antes de entrar en la función que corresponda, y comprobar, por ejemplo, si la dirección del wallet que envía la transacción que incluye a esa función coincide con la que tiene los permisos para ejecutarla. Se trata de una interesante herramienta que permite asegurar el acceso a ciertas funcionalidades con plena confianza.

#### 2.2.2.6 Pagos:

Recordemos que un contrato inteligente puede recibir *ether* y reenviarlo a una dirección. El *ether* que recibe un contrato va directamente a las funciones que lleven el indicador *payable*. Una vez dentro de esta función, el *ether* será enviado a uno u otro sitio (o incluso permanecer en el contrato) en función de lo que le ordene el código, sin poder alterar estas “reglas del juego”.

En el momento que se envía una transacción desde una cuenta, con destino un contrato inteligente, la primera no es consciente del código contenido dentro de esta, siendo “ciega” en lo que respecta a las funciones que puede haber dentro del contrato de destino. Es por ello que muchas veces la función no tiene nombre ni siquiera.



## 3 Estado de la cuestión

“La inversión en empresas emergentes, conocidas como *startups*, que se dedican al desarrollo de *blockchain* sumó 3.000 millones de euros en 2017, un 340% más que el año anterior, según el monitor del mercado global *Novum Insights*, de *Goldman Sachs Report*”. [20] El siguiente paso que nos ocupa es evaluar si esto se refleja en la industria de los Seguros.

### 3.1 Blockchain y Seguros del automóvil

Como ocurre con muchos sectores, a día de hoy, las compañías de seguros se encuentran en un permanente proceso de actualización, “no solo para dar respuesta a los clientes con pólizas de seguro de coche cada vez más completas, sino para lograr que la gestión de estas sea sencilla y práctica”. [21]

De este modo, las aseguradoras son “cada vez más conscientes de la importancia que tiene la presencia en el móvil”. Es por ello por lo que se esfuerzan “por facilitar la gestión integral de sus pólizas de seguro de coche desde estos dispositivos”. [21]

Por otro lado, las compañías de seguros están centrando su interés en particular en las tecnologías de registro distribuido; en especial, en las redes *blockchain*. De hecho, según un informe de la consultora *Capgemini*, “en torno al 41% de las aseguradoras tardará entre uno y tres años en adaptarse al uso del *blockchain*”, ya que la integración de esta tecnología en el sector asegurador sería “una nueva forma de distribuir información y almacenar datos”. Por una parte, esta tecnología facilita que las aseguradoras puedan “renovar las pólizas de sus clientes con más transparencia -al ser datos personales proporcionados por los propios asegurados-, así como agilizar la liquidación de siniestros y gestionar los cobros y pagos con más rapidez “.

El creciente interés de las compañías de seguros en la tecnología *blockchain* se demuestra con la iniciativa en la que decidieron embarcarse las principales aseguradoras europeas (*Aegon*, *Zurich*, *Allianz*, *Munich Re* y *Swiss Re*) en 2016, conocida como *B3i*. El acrónimo responde a *Blockchain Insurance Industry Initiative* y tiene como objetivo “proporcionar un punto de encuentro en el que las cinco empresas puedan intercambiar

sus ideas, probar diferentes casos de uso y desarrollar conceptos que, en última instancia, puedan transformar la forma en la que proporcionan sus servicios”. [22]

A estas reflexiones a las que van a tener que hacer frente las compañías de Seguros se le añade la necesidad de tener en cuenta los nuevos cambios en la movilidad tanto en ciudad como entre ciudades. La irrupción de modelos de negocio como *BlaBlaCar* y de los servicios de *car sharing free floating* (tipo de servicio ofrecido por *Car2go*, *Emov*, *Zity*, *Wibble...*) reducirán drásticamente el uso del coche particular. Como explica Fernando Izquierdo, presidente de *Emov*, "los coches privados están un 95% del tiempo parados. Compartir vehículo hace que la movilidad funcione". [23]

La disminución del uso del coche particular por parte de los ciudadanos tiene, naturalmente, una relación directa con el mercado de los Seguros de automóvil. Dejando a un lado el Seguro a Terceros con el que la Ley obliga a contar para poder circular, el cual sí debe estar contratado todo el año (y, en principio, con la misma compañía en todo momento), uno puede plantearse la posibilidad de abandonar la práctica tradicional de estar contratado todo el año con una única compañía de Seguros, independientemente de las horas de uso que se le da al coche, y embarcarse en otras posibilidades.

Una solución a esta situación podría ser una plataforma que supusiera un lugar de encuentro entre compañías de Seguros y usuarios que desean estar asegurados por estas compañías, cuando están conduciendo su coche, y no todo el año porque sí. Un sistema de contratos de seguros cortos, de un trayecto únicamente, desligaría al usuario de una única compañía y le permitiría beneficiarse de la competencia entre las empresas de la red. En un trayecto A, con una serie de características, podría ser que la compañía SEGUR le dé el precio más barato de todas las compañías presentes en la red, mientras que en un trayecto B, fuera, en cambio, la compañía ASEG la que ofreciera al usuario el menor precio de entre todos los devueltos por las aseguradoras participantes.

Precisamente, *Emov* ya tiene un sistema por el que pagas un euro para estar eliminar la franquicia del seguro durante el viaje. Se trata de una “función que permite al usuario eliminar la franquicia en caso de que haya producido daños en el vehículo. De esta manera, el cliente no tendrá que afrontar un potencial cargo de la franquicia en caso de un accidente en el que sea responsable”. [24] Sigue siendo una cantidad fija que no tiene en cuenta la distancia del viaje, pero tiene mucho que ver el concepto de estar

asegurado en un trayecto que sabes que vas a hacer y que podrías marcar en el mapa, realizando una gestión previa al arranque del automóvil.

## 3.2 Otras DApps

Por supuesto, existen ya proyectos en marcha, con más o menos éxitos, basados en el desarrollo de una aplicación descentralizada como la que se va a desarrollar en este proyecto.

- Un Proyecto Final de Carrera de la Universidad Politécnica de Cataluña, por ejemplo, ha consistido en el estudio de protocolos *blockchain* en sistemas de votación electrónica. [4]
  
- Un programador ha desarrollado un *Smart Contract* en Ethereum llamada *BeCode*. [25] Básicamente, el contrato acepta pagos de cualquier cuenta que se añaden a los fondos del contrato, y añade una serie de condiciones para que el programador, desde su *wallet*, tenga el derecho de retirarlos. En este caso, va asociado a otra aplicación con un dispositivo que le permite detectar cuánto corre de una tirada. Si consigue correr 60 kilómetros de una vez, le permite retirar los fondos que los donantes le han transferidos. Si no los consigue, estos son automática e inmediatamente devueltos a los donantes que correspondan.

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

## 4 Definición del Trabajo

### 4.1 Justificación

Evaluando otras aplicaciones descentralizadas y las posibilidades que ofrecen los contratos inteligentes, parece atractivo realizar la aplicación de Seguros sobre una red de tipo *Ethereum*. Por tanto, esto quiere decir que incorporaríamos los métodos de los protocolos basados en la cadena de bloques. La posibilidad de rastrear la historia completa y los movimientos de la información y los archivos en la cadena de bloques abre un amplio abanico de posibilidades para muchos mercados. En particular, llama la atención de todos aquellos en los que la componente legal juega un papel importante, debido a las garantías que ofrece *blockchain* en seguridad de la información, trazabilidad de esta y seguridad.

En un artículo de GFT (compañía dedicada al diseño y la implementación de soluciones de Tecnologías de la Información para el sector financiero) se reflexiona sobre la conveniencia de utilizar tecnologías de registro distribuido, como *blockchain*, en distintos casos de uso. Para ello, se plantean una serie de preguntas previas para tomar la decisión de embarcarse o no en un Proyecto de este tipo:

- “¿Tu caso de uso necesita confianza entre partes que normalmente no confiarían entre sí? ¿Es necesario llegar a un acuerdo o consenso?” [26] Al tener los *Smart Contracts* en su código las condiciones del Seguro, todas las partes todas las partes tienen acceso a las condiciones del contrato y tienen la capacidad de consultar el origen de los cambios que se produzcan.
- “¿Hay un intermediario que no está añadiendo valor por dinero?” [26] En efecto, en la versión tradicional de un contrato de Seguros “la toma de decisiones se hace de forma “colateral” al contrato. Se invocan procesos online que realizan cálculos y modifican valores, pero no forman parte del contrato en sí: necesitan que una de las partes realice este cálculo y lo haga

cumplir”. [27] Un contrato inteligente recibe información de entrada, la procesa según las reglas establecidas en el contrato y realiza las acciones que este marque.

- “¿Estás tratando de demostrar que la cadena de suministro en la que un activo ha estado (o va a estar) es válida y verdadera?” Efectivamente, se pueden consultar las transacciones que se deseen en la cadena de bloques, gracias a las herramientas que ofrecen los eventos de *Solidity*.

Tras analizar el momento en que se encuentra el sector de los Seguros de automóvil en la búsqueda de mejoras para la manera en la que ofrecen servicios a los usuarios mediante la aplicación de nuevas tecnologías como *blockchain*, así como los nuevos retos que presenta la movilidad dentro y fuera de las ciudades, se ha decidido desarrollar una aplicación descentralizada que ponga las cosas más fáciles a los usuarios últimos en lo relativo a los Seguros de automóvil únicos por trayecto.

## 4.2 Objetivos

El objetivo principal consiste en estudiar la aplicación de la tecnología *blockchain* en el ámbito de los Seguros. Comprender y analizar la conveniencia de esta tecnología en auge en general en diferentes casos de uso del día a día. Idealmente esto permitirá analizar sus posibilidades a la hora de dar respuesta a problemáticas de mejora o de optimización que existen tanto en los hogares como en las empresas. Los objetivos en materia de entregables son:

- El primer objetivo consiste en configurar una red que permita a los usuarios acceder a los diferentes precios ofrecidos por las diferentes aseguradoras desde un mismo lugar común, entregando previamente a la plataforma un registro de su forma de conducir (su “perfil del conductor”), asegurando el anonimato de sus datos. Esta red *blockchain*, en teoría, asegurará también la trazabilidad de toda transacción y de todo registro en la red, garantizando la eliminación del fraude de nuestro marketplace. Este

lugar común no es uno cualquiera, sino uno en el que se van a registrar de manera inmutable las cláusulas de los contratos de trabajo, garantizando la confianza tanto de los clientes como de las Firmas de Seguros. Esta red va a permitir la creación y la firma de contratos de manera automática, evitando la necesidad de la intervención en el proceso de partes cuyo trabajo realmente pueda omitirse, y haciendo que todo el desarrollo del proceso se agilice. La idea pasa por analizar las mejoras ofrecidas por nuestro esquema frente a los sistemas actuales, que generalmente trabajan con registros centralizados.

- El segundo objetivo es el desarrollo de un servicio web que permita a las empresas alterar las condiciones que desean en sus futuras cláusulas (naturalmente, no podrán alterar las condiciones de contratos en vigor). En principio, trabajaremos con un cierto número de variables (velocidad, edad del conductor, datos del coche...) que las empresas colocarán en unos determinados intervalos para ponderar las prioridades que dan a una o a otra variable para el cálculo del precio. En resumen, establecer las “reglas de juego” que ofrecen a los potenciales clientes. Aunque inicialmente trabajaremos con tres empresas aseguradoras que pertenecerán a la red y alterarán sus condiciones, en función de la dificultad que esto plantee, ofreceremos la posibilidad de que usuarios se registren como aseguradora y entren en la red para ofrecer sus servicios. Como he explicado anteriormente, no se trata de la aplicación de una aseguradora facilitando el acceso a su carta de paquetes, sino que se encarga de ofrecer al usuario un despliegue de diferentes ofertas de diferentes empresas, facilitando así la elección de la que considere más adecuada (con generalidad, la más barata).
  
- Aunque sería posible el desarrollo de una aplicación web encargada del registro de los usuarios en la red y sobre la cual se introducirían los datos del trayecto a realizar en cada situación, entendemos también que la manera más interesante de establecer este servicio, desde un punto de vista práctico, sería mediante la introducción de una aplicación para teléfono

móvil que permitiera el registro, el marcado del trayecto y la aceptación, por parte del usuario cliente, de la tarifa más interesante (en principio, la más barata) ofrecida por las diferentes empresas presentes en nuestro servicio. Al no contar con grandes conocimientos de programación de aplicaciones móviles, durante unas semanas se estudiará la viabilidad de esta posibilidad, y en caso de no poder realizarse, se desarrollará el servicio web con la misma funcionalidad. La idea es que la aplicación móvil (o, en su defecto, la aplicación web) le permita al usuario cliente seleccionar el trayecto que se dispone a realizar en un mapa de la interfaz (punto de inicio y de fin del recorrido). Estos datos del trayecto se unirían a los ya disponibles (datos de registro y datos del “perfil del conductor”) para realizar el cálculo del precio que ofrece cada una. De esta manera, se pierde la atadura a largo plazo de una firma en particular (en muchas ocasiones no se está realmente seguro de si se va a hacer uso de los “paquetes de kilómetros” o no y de si nos vale la pena hacerlo), para dar paso a la búsqueda continua del mejor precio, independiente de la compañía de que se trate, para el trayecto que queremos realizar.

Finalmente, existe un fuerte interés en entender si realmente el uso de *blockchain* es una moda y algo que suena interesante y actual y todo el mundo se quiere “subir al carro”, o estamos hablando de algo realmente útil. Si es así, me gustaría analizar si he visto utilidad o no de la tecnología en la escala en la que he desarrollado e implementado mi proyecto (tanto a nivel de plazos como de tecnología disponible y de conocimientos previos).



## 5 Desarrollo de una aplicación móvil sobre una red privada de *Ethereum*

### 5.1 Análisis del sistema: desarrollo sobre *Ethereum*

El desarrollo de una aplicación con un lenguaje y herramientas desconocidas hasta el momento requiere una preparación que nos introduzca en el contexto que rodea al sistema en cuestión. Una vez completado un aprendizaje que permitió la comprensión de las diferentes cuestiones explicadas previamente, se procedió a evaluar las posibilidades reales de conexión a la Máquina Virtual de *Ethereum*.

En cualquier caso, el “protocolo que mantiene la base de datos y rige la red *Ethereum* está codificado en programas que se comunican entre sí. A estos programas se les llaman *clientes* y están programados en varios lenguajes de programación. Los más famosos son *geth* (abreviado de *go-ethereum*) y *Parity*”. [28]

*Geth* ofrece la posibilidad de desplegar una red *blockchain* privada en una máquina personal del usuario, a la que podemos añadir nodos, en función de nuestras necesidades. A nivel de funcionalidades, se trata de una red idéntica a la red principal de *Ethereum*.

Desde *Github*, se define *geth* como “la interfaz de línea de comando que permite trabajar con un nodo *Ethereum* implementado en *Go* (lenguaje de programación inspirado en C). Permite, entre otras cosas, minar verdadero *ether*, transferir fondos entre direcciones, crear contratos y enviar transacciones, o explorar el historial de los bloques”. [29] Es posible descargarlo desde *GitHub* y, en este caso, ha sido ejecutado desde el Terminal (programa interno del Mac que permite trabajar con interfaces de líneas de comandos).

### 5.1.1 Configuración de un cliente *Geth*.

Ha estado a disposición de este Proyecto un ordenador portátil HP840G (4GB de RAM y 50GB de disco duro) en el que se ha instalado una máquina virtual gracias a *VirtualBox 5.2.4* de *Oracle*. De ese modo, la red se encuentra desplegada en un servidor virtual con *Ubuntu Linux 16.04 LTS*. En esta máquina se agrega el repositorio de software *Ethereum*, instalándolo. De esta manera, se consigue acceso a una red que simulará las características de la Máquina Virtual de *Ethereum (EVM)*.

La red se ubica en un emplazamiento de esta máquina virtual y antes de ser desplegada es posible describir las condiciones iniciales de la red *blockchain* y de la cadena de bloques, permitiendo la configuración manual de diferentes aspectos como el número del bloque en el que comienza la cadena, la dificultad de minería o el límite del coste del gas por bloque. Esto se lleva a cabo añadiendo al directorio de la red un archivo *JSON* que contenga las configuraciones deseadas para el bloque *génesis* de la red.

Al configurar la red, es posible crear una cuenta que pertenezca a la red desde el momento en que esta es creada. Aunque se puede configurar, la primera cuenta será por defecto la conocida como *etherbase*, es decir, aquella que se *nutra* de los fondos recogidos por el nodo cuando este mine transacciones.

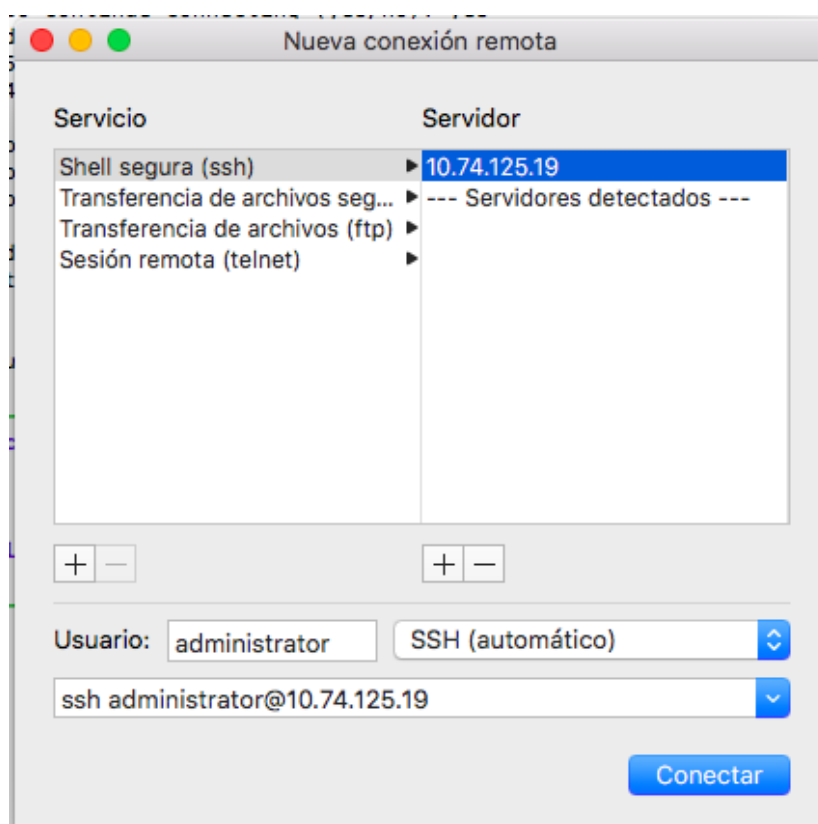
Al tratarse de una red privada, en el momento de desplegarla, existe también la posibilidad de otorgar *ether* a una o más cuentas en el momento inicial, en función de nuestros intereses (“cuenta pre-financiada”). Como veremos más adelante, por lo general, se trata de la cuenta utilizada para desplegar los diferentes contratos inteligentes a la red, pues está continuamente obteniendo fondos al minar (recordemos que la acción de desplegar también se considera una transacción y, por tanto, supone un coste de gas para la cuenta que la envía).

Una vez parametrizado el bloque inicial deseado para la red *blockchain* desplegada, es posible materializarlo gracias a un comando de *geth: init*. Entonces, es posible arrancar el cliente *Ethereum* para conectarse a la red que se ha inicializado con el bloque génesis. El proceso queda entonces ejecutándose.

Teniendo una red *blockchain* en una máquina, existe la posibilidad de añadirle nodos. El comando *geth* permite tanto configurarlos como ponerlos en marcha. En cuanto a la configuración del nodo, quedó habilitado el servidor *HTTP-RPC*. De esta manera, se configuró la interfaz de comunicaciones del servidor *HTTP-RPC* en el *localhost* del nodo (apuntándose a sí mismo), y se indicó como puerto de comunicación el 8545.

### 5.1.2 Conexión a la red privada. Interacción con comandos de *Geth*.

De cara a la realización de un Proyecto haciendo uso de la red, la primera cuestión es la de conectar un equipo al nodo que se encuentra funcionando en la red desplegada en la máquina virtual. Esto se realiza estableciendo una conexión remota vía Shell con la dirección IP del nodo (esto es, la interfaz de comunicaciones del servidor *HTTP-RPC* del nodo).



*Ilustración 5: Conexión remota a la IP de un nodo (Paso 1)*  
*[Elaboración propia]*

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
ramirorivera — ssh administrator@ — 80x24
Last login: Fri Aug 10 12:21:56 on ttys000
MacBook-Air-de-Ramiro:~ ramirorivera$ ssh administrator@
administrator@ ; password: 
```

*Ilustración 6: Conexión remota a la IP de un nodo (Paso 2) [Elaboración propia]*

```
ramirorivera — administrator@ethereum-node1: ~ — ssh administrator@10.74....
Last login: Fri Aug 10 12:21:56 on ttys000
MacBook-Air-de-Ramiro:~ ramirorivera$ ssh administrator@
administrator@ ; password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-81-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

244 packages can be updated.
124 updates are security updates.

*** System restart required ***
Last login: Fri Aug 10 12:22:02 2018 from
administrator@ethereum-node1:~$
```

*Ilustración 7: Conexión remota a la IP de un nodo (Paso 3) [Elaboración propia]*

De cara a interactuar con el nodo, desde el *Terminal* es necesario posicionarse sobre su emplazamiento en la máquina virtual a la que el equipo utilizado se ha conectado para ejecutar el comando “*geth attach*”, seguido de la ubicación del nodo. Esto permite la conexión al proceso *geth* que ya estaba ejecutándose, y da comienzo a un entorno interactivo *JavaScript*. La documentación de *GitHub* en lo que respecta a *geth* es extensa y propone un gran número de comandos que ofrecen la posibilidad de trabajar con la red Blockchain a través del nodo

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
ramirorivera — administrator@ethereum-node1: ~ — ssh administrator@10.74....
[administrator@: s password: ]
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-81-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

244 packages can be updated.
124 updates are security updates.

*** System restart required ***
Last login: Fri Aug 10 12:22:02 2018 from 
[administrator@ethereum-node1:~$ geth attach http:// 8545 ]
Welcome to the Geth JavaScript console!

instance: Geth/v1.6.5-stable-cf87713d/linux-amd64/go1.8.1
coinbase: 0x0b3eca6dffe4fff33dad10dae4cf8c4de489857
at block: 1335961 (Fri, 10 Aug 2018 12:21:51 CEST)
datadir: /var/lib/ethereum
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 tpx
ool:1.0 web3:1.0

> █
```

*Ilustración 8: Conexión a un proceso geth que ya está en marcha [Elaboración propia]*

### 5.1.3 Mejor acceso a las funcionalidades: *Web3* y el servidor *JSON-RPC*

Aunque la interfaz de líneas de comando del cliente *geth* sea muy interesante para acceder a las cuentas que están desplegadas, desbloquearlas, comprobar su balance... la realidad es que para las funciones que se desean realizar para desarrollar la aplicación (despliegue de contratos, acceso a métodos, recuperación de resultados...) serán necesarias otras maneras de interactuar con la red. Ha resultado muy útil hasta el momento, especialmente de cara a la comprensión de la plataforma *Ethereum* en su sentido más básico, pero se requiere una herramienta que permita el acceso más o menos intuitivo a los contratos inteligentes que se van a utilizar.

Es por ello por lo que, en el momento de despliegue de la red mediante *geth*, se especificó que nuestro nodo contara con la capacidad de ser accesible mediante una interfaz de programación de aplicaciones (en adelante, *API*, de sus siglas en inglés)

llamada *Web3*. Una *API* es “un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software“. [30]

La biblioteca *Web3* es muy interesante para el desarrollo de aplicaciones descentralizadas, pues ofrece una interfaz que facilita la comunicación con el nodo. En particular, permite acceder a los métodos de su puerto *RPC* y trabajar con los módulos de *JSON-RPC*, un “protocolo ligero sin estado de llamadas a procedimientos remotos (*RPC*, del inglés *Remote Procedure Call*). [...] En líneas generales, lo interesante de *RPC* es que permite la ejecución remota de funciones, aunque sea en otro lenguaje de programación.” [31] Por otro lado, *JSON-RPC* trabaja con *JSON*, un formato ligero de intercambio de datos fácil de entender, ofreciendo facilidades a las máquinas tanto en la generación como en la interpretación.

Existe mucha documentación de *Web3* disponible en *GitHub*, lo que ha facilitado la realización de este Proyecto.

## 5.2 Arquitectura y diseño de la *DApp*

Una vez se cuenta con una red *Ethereum* privada configurada como deseamos, desplegada y a la que podemos acceder, es posible comenzar el diseño de una *DApp* que materialice las funcionalidades que se desean.

Como se ha venido explicando, el peso de la lógica de la aplicación descentralizada recae sobre los contratos inteligentes. Los *scripts* de estos contratos son desplegados a la red y a partir de ese momento su código no puede alterarse. Una vez se encuentren en la red, las *reglas del juego* no podrán modificarse. Es por esto por lo que es muy importante que los contratos que se vayan a utilizar en la aplicación estén bien definidos y no tengan puntos débiles en su lógica.

Es importante entender que lo importante de aplicación descentralizada, lo que tiene realmente un interés detrás, son los contratos inteligentes. Como hemos visto con la interfaz de línea de comandos de *geth*, es posible acceder a estos contratos por distintas vías. Aunque el lenguaje *Solidity* es sencillo y no da pie a grandes complicaciones, es de vital importancia estar seguros de que ninguna de estas vías puede “jugar” con nuestras

“reglas del juego”. A partir de ahí, la implementación en una aplicación móvil o en un sitio web no deja de ser una manera de facilitar la interacción intuitiva del usuario medio con los contratos inteligentes, sin requerir conocimientos previos de programación.

El proceso que se ha llevado a cabo para este Proyecto a la hora de decidir la arquitectura de la aplicación descentralizada ha sido el de estudiar primero qué tipo de aplicación móvil se deseaba tener para comprender después la manera como el lenguaje en que esta se desarrollara accediera a la biblioteca *Web3*. Esto se ha realizaría sin hacer hincapié en un primer momento en el *front-end* de esta, especialmente al tratarse probablemente de un lenguaje desconocido, poniendo el foco en el acceso y comunicación con los *Smart Contracts*.

### 5.2.1 Elección de IDE: *Xamarin* y *Nethereum*

Varios meses de trabajo en Prácticas en la firma que ha propuesto este Trabajo de Fin de Grado me habían permitido adquirir ciertas nociones de programación en el lenguaje *C#*. Durante un tiempo había trabajado con *ASP.NET*, un “entorno para aplicaciones web desarrollado y comercializado por *Microsoft*. Es usado por programadores y diseñadores para construir sitios web dinámicos, aplicaciones web y servicios web *XML*.” [32] El entorno de desarrollo integrado (*IDE*) que utilizaba durante este periodo era *Microsoft Visual Studio*, que permite no solo trabajar con páginas web sino también con dispositivos móviles, consolas...

El paquete de herramientas de *Visual Studio* para el desarrollo de aplicaciones móviles, llamado *Xamarin*, resultó ser el más adecuado para la realización de este Proyecto, pues además del hecho de resultar familiar e intuitivo para un programador de *ASP.NET*, pues forma parte del *IDE* de *Visual Studio*, permitía su descarga tanto en PC como en Mac (máquina con la que se realizó este Proyecto).

La elección de *IDE* para trabajar únicamente quedaba supeditada a la localización de una biblioteca que permitiera acceder a las funciones de la biblioteca *Web3*, que permitiría la interacción vía *RPC* con el cliente *geth*, aplicándolas a un proyecto en *.NET*.

*Nethereum* reúne estas características. Es, en otras palabras, una implementación de la biblioteca *Web3* en lenguaje *C#*, permitiendo, mediante métodos de *JSON-RPC*, el

despliegue de contratos inteligentes, el acceso a las llamadas a sus funciones y al envío de transacciones, y la gestión de los eventos. Utilizando la documentación disponible en *GitHub*, pude desplegar en la red varios contratos de pruebas y acceder a las cuentas ya existentes en la red, como lo había hecho anteriormente con la consola de *geth* y el *Terminal*.

Una vez teniendo claras las posibilidades que ofrecía *Nethereum*, podría dar comienzo la programación del código de la aplicación descentralizada. *Xamarin* ofrece la posibilidad de desarrollar aplicaciones para diferentes sistemas operativos partiendo de una base común a todos. En este Proyecto se ha optado por la utilización del sistema operativo *iOS*, debido a las facilidades que ofrece *Visual Studio* en una máquina Mac, especialmente en lo que respecta al simulador en la pantalla (ofrece acceso al simulador de *iPhone* de *Xcode*).

## 5.2.2 Definición de los roles en la aplicación descentralizada

### 5.2.2.1 Usuarios tipo Cliente

La aplicación se ha concebido para ofrecer la posibilidad de registro de nuevos clientes que deseen operar en la plataforma. Una vez registrados por primera vez, habrán de realizar un *log-in* para acceder a la aplicación.

Los datos que se pedirán por pantalla al nuevo usuario serán: el nombre, los apellidos, el DNI, la dirección del domicilio, así como el país en el que reside, la edad y el número de años que han pasado desde que obtuvo el permiso de conducir. Lo que identificará a este tipo de usuarios será el DNI.

Una vez registrados o habiendo iniciado sesión en la plataforma, tendrán la posibilidad de solicitar, en cualquier momento, un Seguro para un trayecto en coche que vayan a realizar. Existen tres tipos de coberturas de seguros de entre las que elegir la que se desea para el trayecto concreto. La introducción del trayecto por parte del usuario se



lleva a cabo mediante punteros en el mapa, que calcularán la distancia total del recorrido que se va a realizar.

La aplicación calcula automáticamente, para cada empresa registrada como tal en la plataforma, el precio que estas ofrecen para su perfil y para ese trayecto con esa cobertura particular, y devuelve por pantalla al usuario una lista con los nombres de las compañías de Seguros y el precio correspondiente.

A partir de este momento, el cliente puede elegir la compañía que desea que le asegure durante el trayecto en coche (se entiende que se tratará de aquella que ofrezca el precio más bajo). Para ello, bastará con hacer clic sobre esta y confirmar la elección, lo cual activará el contrato de Seguros.

#### 5.2.2.2 Usuarios tipo Compañía

Las compañías, en esta plataforma, registran únicamente su nombre (razón social) y un identificador numérico.

Las compañías no tendrían, por ahora, acceso a una aplicación móvil, dándose por hecho que tendrían la posibilidad de acceder a una aplicación web sencilla en la que revisar sus contratos con los clientes. Eventualmente, tendrían también la posibilidad de cambiar las ponderaciones que desean para establecer los precios del servicio ofrecido, a la hora de dar más o menos peso a los diferentes parámetros con los que la aplicación calcula el precio de los trayectos.

Actualmente, al tratarse de un proyecto piloto, las compañías tendrían únicamente la posibilidad de calcular los precios basándose en cuatro parámetros: distancia del trayecto (calculada gracias a los marcadores que el usuario coloca en el mapa), tipo de cobertura escogida (Seguro a terceros ampliado, Seguro a todo riesgo con franquicia y Seguro a todo riesgo sin franquicia), la edad del usuario y el número de años que han pasado desde que el usuario obtuvo el permiso de conducir. Esto se producirá de manera automática, gracias a la inclusión de los contratos inteligentes en la aplicación. La confianza en la cadena de bloques permite que la compañía pueda colocar sus “condiciones” en la aplicación y se desentienda, sin tener que revisar cada contrato de seguros antes de activarlo.

De esta manera, la compañía se debe encargar de mejorar sus servicios y sus cálculos para mejorar sus ponderaciones de modo que tengan sentido para el mercado, y se trate de una competencia leal entre ellas. El usuario último, por supuesto, se beneficia de esto.

Las compañías podrían cambiar en cualquier momento sus ponderaciones. Naturalmente, no podrán alterar las ponderaciones de los contratos una vez estos han sido puestos en marcha, pero sí podrán cambiar las de los futuros contratos (estos cambios serán públicos a toda la red).

### 5.2.3 Concepción de los contratos inteligentes: el contrato “maestro”

#### 5.2.3.1 Contrato *ClienteUsuario.sol*

Este contrato hará las veces de “perfil” del usuario, acreditando su pertenencia a la plataforma. En el momento que un cliente de nuevo ingreso en la plataforma desee empezar a formar parte de esta, llevará a cabo un registro de usuario en la aplicación móvil. Se le pide por pantalla varios campos:

- Nombre
- Apellidos
- DNI
- Edad
- Número de años con carné
- Dirección
- País
- Contraseña

```
struct ClientPersonal {  
  
    address adr;  
    string name;  
    string surname;  
    uint personal_id;  
    uint edad;  
    uint numanosconcarntet;  
  
    string street;  
    string country;  
  
    string password;  
  
    uint256 registroDate;  
  
}
```

*Ilustración 9: Estructura de datos del contrato ClienteUsuario.sol [Elaboración propia]*

En el momento del despliegue del contrato, estos campos se pasan por parámetros al constructor, y este ejecuta dos funciones internas que añaden estos parámetros, así como la fecha y hora de registro (“registroDate”), a una estructura de datos del contrato llamada “ClientPersonal”.

```
//Esta funcion se ejecuta al crear el contrato, añadiendo toda la informacion del cliente  
  
function ClienteUsuario (string _name, string _surname, uint _personal_id, uint _edad,  
uint _numanosconcarntet, string _street, string _country, string _password) {  
  
    addClientePersonal(_name, _surname, _personal_id, _edad, _numanosconcarntet,  
        _street, _country, _password);  
  
    addClienteReport(_personal_id, address(this));  
  
}
```

*Ilustración 10: Constructor del contrato ClienteUsuario.sol [Elaboración propia]*

Por otro lado, cada vez que el usuario activa nuevos contratos de Seguros, las direcciones de esos contratos son añadidas al vector del contrato “trayectosQueHeContratado”, de manera que el cliente pueda acceder, eventualmente y mediante llamadas a la *blockchain*, los datos deseados del contrato *ClienteSolicitudTrayecto.sol* que corresponda.

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

Es importante destacar que la cuenta encargada de enviar la transacción de despliegue en la aplicación va a ser la *etherbase* de la red, pero en teoría sería el *wallet Ethereum* del usuario, lo cual facilitaría la inclusión de modifiers.

Uno de los elementos imprescindibles de este contrato es el evento “pago”, que enviará a la cadena de bloques el precio, la dirección de la compañía y el momento preciso, cuando se seleccione la elección para el trayecto particular, facilitando el posterior rastreo de la información en caso de necesitarse. Como este, podríamos introducir tantos eventos como deseáramos, para dejar constancia inmutable de hechos que ocurren.

```
event pago(address adr, string nombre, string signo,  
           uint amount, uint256 time, uint balance);
```

*Ilustración 11: Definición de un evento en ClienteUsuario.sol*  
*[Elaboración propia]*

Por último, es interesante la funcionalidad de la función “checkPassword”, que no es privada y es llamada desde fuera del contrato. Permite que, en el momento que un usuario haya decidido iniciar sesión (puesto que en otra ocasión ya completó el registro en la plataforma), se haya buscado la dirección en el contrato “maestro” que contiene todas las direcciones, y se haya accedido finalmente a esta función del contrato, se valide la correspondencia de la contraseña introducida con aquella que corresponde al cliente (la introducida en el momento del registro en la plataforma). A partir de aquí se limitará el acceso a las siguientes funcionalidades de la aplicación.

```
function checkPassword(address _adrClient, string _password)
    returns (bool coinc) {

    bool coincide=false;

    if(keccak256(_password)
        ==keccak256(clientDataPersonal[_adrClient].password)){
        coincide=true;
    }
    else{
        coincide=false;
    }

    return (coincide);
}
```

*Ilustración 12: Función “checkPassword” del contrato ClienteUsuario.sol*

### 5.2.3.2 Contrato *Compania.sol*

Este contrato inteligente será el que corresponda a una compañía que desea formar parte de la plataforma y ofrecer servicios de Seguros de coche a los usuarios. Como se ha explicado, este proyecto no desarrolla el front-end para el registro e inicio de sesión de una compañía, pero sí la lógica interna que requiere este *Smart Contract* para que eventualmente sí se desarrolle esta interfaz se puedan realizar todas las funciones a las que puede acceder un usuario de tipo Compañía.

Al igual que ocurre con el contrato inteligente *Cliente.sol*, en el momento del despliegue del contrato, se pasan una serie de campos por parámetros al constructor, y este ejecuta dos funciones internas que añaden estos parámetros, a una estructura de datos del contrato llamada “CompaniaData”. En este caso, se trata del nombre de la compañía, un código numérico único que la identifique (un “DNI” de la empresa) y una contraseña para permitir el desarrollo en un futuro de su propia interfaz de acceso.

```
uint256 indexPrecioDevuelto;  
uint256 indexContratoActivado;  
  
uint pond_edad = 2;  
uint pond_numanosconcarner = 2;  
uint pond_distancia = 2;  
uint pond_tipoCobertura = 1;  
  
struct CompaniaData{  
  
    string nombreCompania;  
    uint idCompania;  
    address adrCompania;  
    string pass;  
  
}
```

*Ilustración 13: Estructura de datos del contrato Compania.sol  
[Elaboración propia]*

Este contrato posee una función llamada “calcularYDevolverPrecio”. Esta función no es privada y puede ser llamada desde fuera del contrato. Recibe por parámetros las variables que sirven para el cálculo del precio que esta empresa en particular va a devolver por pantalla (distancia, tipo de cobertura, edad y número de años con carné) y las introduce en una sencilla ecuación en la que, por otro lado, intervienen las ponderaciones propias de cada empresa.

Como variables del contrato se pueden encontrar, por supuesto, las diferentes ponderaciones que intervendrán en la ecuación del cálculo del precio, así como los índices que permiten llevar una cuenta ordenada tanto de los precios que la empresa ha devuelto como de los contratos que se han activado con usuarios para trayectos solicitados.

De este modo, cada vez que se devuelve un precio o sea activa un contrato (esto último se produce cuando se añade un precio y un ID de compañía al contrato *ClienteSolicitudTrayecto.sol*, como se verá más adelante), estos son añadidos a dos vectores de códigos de solicitud (“numPreciosDevueltos” y “numContratosActivos”) particulares de la compañía. Esto permitiría a la compañía llevar un registro claro y fácilmente accesible de sus acciones particulares sobre el mercado de la plataforma y

evaluar mediante sus propias herramientas la conveniencia de sus ponderaciones de cara a mejorar la competitividad de la empresa.

```
uint[] public numPreciosDevueltos;  
uint[] public numContratosActivados;
```

*Ilustración 14: Vectores de direcciones del contrato *Compania.sol*  
[Elaboración propia]*

Se puede decir que todas las empresas que participan en la plataforma tienen acceso a esta ecuación (en una futura implantación de la aplicación esta sería más compleja, pero aún así única y pública), la conocen bien, la estudian y su manera de mejorar sus beneficios no es otra que mejorar sus propias ponderaciones siguiendo las “reglas del juego”, dando lugar a un punto de encuentro de empresas limpio y transparente. No olvidemos que en esta plataforma una compañía de Seguros coloca sus ponderaciones y se “desentiende” del resto del proceso, ya que la puesta en marcha de los contratos de Seguros se realiza de forma automática por la aplicación.

```
function setPonderaciones(uint _pond_edad, uint _pond_numanosconcarner,  
uint _pond_distancia, uint _pond_tipoCobertura)  
  
    onlyCompania{  
  
        pond_edad= _pond_edad;  
        pond_numanosconcarner= _pond_numanosconcarner;  
        pond_distancia= _pond_distancia;  
        pond_tipoCobertura= _pond_tipoCobertura;  
        string nombre = companiaDatos[address(this)].nombreCompania ;  
  
        statusComp(msg.sender, "Cambio de ponderaciones", block.timestamp);  
    }  
}
```

*Ilustración 15: Función “setPonderaciones” del contrato *Compania.sol* [Elaboración propia]*

Eventualmente, las empresas tendrían la posibilidad de acceder con su aplicación web o móvil a la función “setPonderaciones” que les permitiría cambiar las variables de su contrato *Compania.sol* correspondiente. Como ocurre con el contrato *ClienteUsuario.sol*, este contrato de las compañías debería haber sido desplegado por

parte del *wallet* correspondiente a la aseguradora. Así, se hubiera guardado su dirección en el momento de despliegue de la instancia de contrato al registrarse en la plataforma y se podría validar con *modifiers* el acceso a la función “setPonderaciones”. Recordemos que las transacciones las hemos enviado siempre desde la dirección del *wallet* de pruebas de la red (el creado en el momento del despliegue de la red, el *etherbase*). De este modo, se ha desarrollado el código de la lógica en el contrato inteligente, pero esta no se ha implementado en la aplicación, al tratarse de un Proyecto de prueba.

Se ha introducido también un evento “statusComp” que permite añadir a la cadena de bloques los argumentos, notificando así que se ha producido un cambio en las ponderaciones, pasando por parámetro, también, la dirección del contrato y el momento en que se ha producido.

#### 5.2.3.3 Contrato *ClienteSolicitudTrayecto.sol*

Este contrato inteligente será el que se desplegará en el momento de solicitarse un nuevo trayecto por parte de un usuario de la plataforma. En este instante se añaden al contrato (como ocurre anteriormente, esto se hace pasando parámetros al constructor) el DNI del usuario solicitante, así como sus datos que sean útiles para el cálculo del precio por parte de las compañías (edad, número de años con carné) y la fecha de creación. También se añade un código numérico que se muestra al usuario en el momento de la solicitud de un nuevo trayecto y que sirve para identificar esta solicitud como única. Por último, se añaden también a la estructura de datos del contrato (llamada “ClienteSolicitud”) los datos particulares de la solicitud como son la distancia del trayecto pedido y el tipo de cobertura deseado.



```
struct ClienteSolicitud {  
  
    address adrSolicitud;  
    uint codigoSolicitud;  
  
    uint distancia;  
    string tipoCobertura;  
  
    uint personal_id;  
    uint edad;  
    uint numanoscon carnet;  
  
    uint256 creationDateSolicitud;  
  
    uint idCompania;  
  
    uint256 precio;  
  
}
```

*Ilustración 16: Estructura de datos del contrato ClienteSolicitudTrayecto.sol [Elaboración propia]*

Eventualmente, una vez se devuelvan al usuario por pantalla los precios ofrecidos por todas y cada una de las empresas participantes de la plataforma, y se haya escogido uno, se llamará de forma externa a la función “setEleccion”. Esta permitirá añadir a la estructura de datos de la solicitud el ID de la compañía escogida para que asegure el trayecto del usuario y el precio devuelto por esta.

```
function setEleccion(uint _idCompania, uint256 _precio, uint _personal_id)  
    returns (uint idCompania, uint precio) {  
  
    // clientDataEleccion[address(this)] = ClienteEleccion({  
    clientDataSolicitud[address(this)].idCompania=_idCompania;  
    clientDataSolicitud[address(this)].precio = _precio;  
  
    return (clientDataSolicitud[address(this)].idCompania,  
        clientDataSolicitud[address(this)].precio);  
}
```

*Ilustración 17: Función “setEleccion” del contrato ClienteSolicitudTrayecto.sol [Elaboración propia]*

Naturalmente, la llamada externa a esta última función supone el envío de una transacción, ya que genera cambios en las variables del contrato. Una transacción que no envía valor, por supuesto (transacción de comunicación entre contratos inteligentes). Igual que ocurre con el despliegue, aunque esta función es enviada desde la *etherbase*,

como ocurre con los otros contratos *Compania.sol* y *ClienteUsuario.sol*, en teoría, en este caso, debería ser enviada por el *wallet* del cliente. En esta ocasión, no se han desarrollado los hipotéticos *modifiers*, puesto que no se iba a llevar a la práctica y ya se había estudiado esta herramienta en los otros contratos. Sin embargo, esta lógica es muy importante para que tenga sentido la aplicación descentralizada. En el momento de enviar la transacción de despliegue, el contrato guardaría en la variable de contrato “*adrClienteUsuario*” la dirección del *wallet* (lo mismo que ocurre en el contrato *ClienteUsuario.sol*), guardándole como “el que le creó”. El *modifier* correspondiente, que acompañaría a la función del contrato, validaría que la dirección que enviase una transacción que contuviera esa función coincidiese con la que tenía guardada como variable de contrato.

#### 5.2.3.4 Contrato *Report.sol*

Este contrato inteligente será el que facilite el funcionamiento de la lógica de la aplicación. Como hemos visto, una red *Ethereum* privada da cabida a la convivencia de numerosas aplicaciones al mismo tiempo. Con el objetivo de simplificar el acceso a los datos que corresponden exclusivamente a la lógica de esta aplicación particular, se ha desarrollado este contrato. *Report.sol* hace las veces de “base de datos particular de esta aplicación”, grabando en distintas estructuras duplas de identificadores y sus correspondientes direcciones en la cadena de bloques.

Así, en los constructores de los tres tipos de contrato inteligente que utiliza esta aplicación, se incluyen funciones que añaden al contrato *Report.sol*, elementos a vectores de direcciones. Esto permite que, en cualquier momento, con el identificador, podamos acceder a la dirección en la que se encuentra la instancia de contrato correspondiente.

```
struct ClientsAddress{
    uint personal_id;
    address adr;
}

struct ClienteSolTray{
    uint codigoSolicitud;
    address adrSolicitud;

    uint personal_id;
    uint idCompania;

    uint256 precio;
    // string confirmado;

    uint distancia;
    string tipoCobertura;

    uint edad;
    uint numanoscon carnet;
}

struct Comp {
    address adrCompania;
    uint idCompania;
}
```

*Ilustración 18: Estructuras de datos del contrato Report.sol  
[Elaboración propia]*

De ese modo, a grandes rasgos, con el identificador es posible consultar gracias a este contrato “maestro” todas las direcciones que se deseen en la cadena de bloques. Esto es posible mediante la definición de un mapeo de la estructura a través de este identificador. Al tratarse de un proyecto piloto que supone un primer contacto con la tecnología *blockchain*, se decidió incluir esta herramienta que facilitaría el acceso a la información que interesa a este Trabajo.

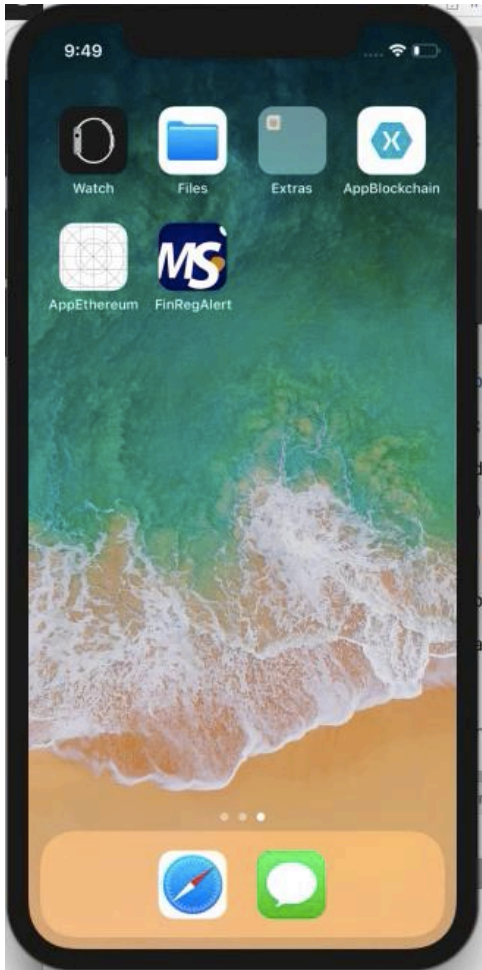
```
mapping (uint => ClientsAddress) public clienteAdr;  
mapping (uint => ClienteSolTray) public clientSolTrayAdr;  
mapping (uint => Comp) public compAdr;  
  
address[] numClientes;  
address[] numClientesSolicitudTrayecto;  
address[] numCompanias;
```

*Ilustración 19: Mapeos y vectores de direcciones del contrato Report.sol [Elaboración propia]*

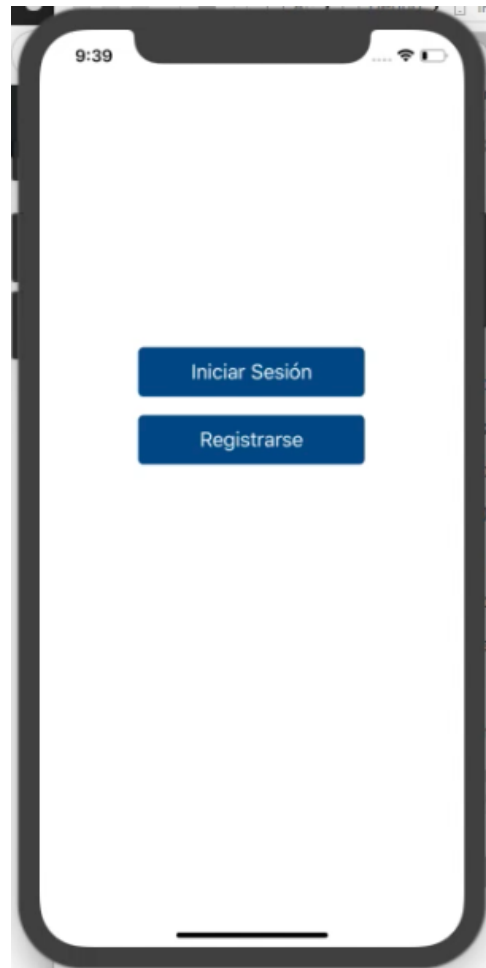
## 5.3 Desarrollo, pruebas e implementación

### 5.3.1 Desarrollo de las vistas de la aplicación

La aplicación móvil se ha desarrollado de modo que, en la primera pantalla, se pueda escoger entre proceder al registro en la plataforma, si no se ha realizado en una sesión previa, o bien al inicio de sesión en la aplicación haciendo uso del DNI y la contraseña indicados en su momento.



*Ilustración 20: Vista del simulador de Xcode sin lanzar la aplicación [Elaboración propia]*

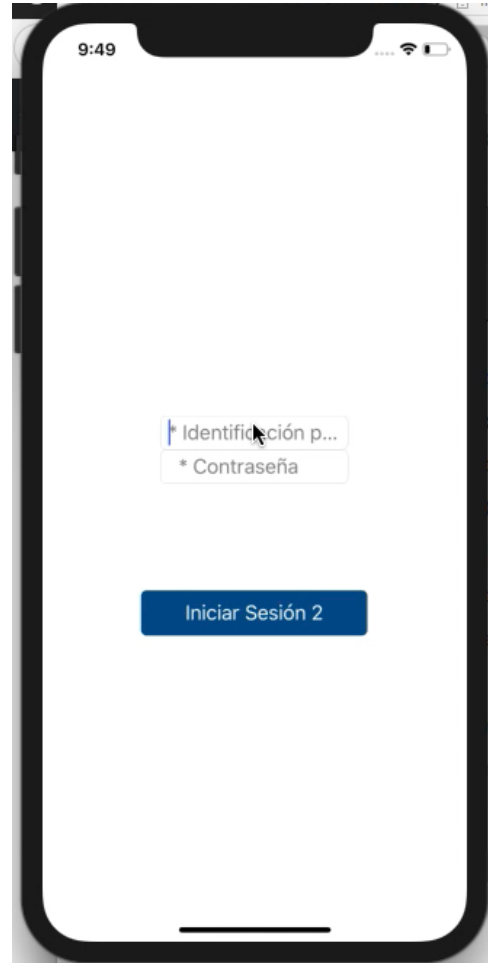


*Ilustración 21: Primera vista de la aplicación de Xamarin [Elaboración propia]*

Si se accede al registro, se deberá introducir todos los datos correspondientes al despliegue del contrato *Cliente.sol*, que acredita que el usuario forma parte (eventualmente, en un futuro, estará asociado al *wallet* de la persona física y este a su identidad digital). Se introduce el nombre, los apellidos, el DNI, la edad, el número de años con carné de conducir, la dirección, el país y la contraseña que se desee para acceder a la aplicación al iniciar sesión en otras ocasiones.



*Ilustración 23: Vista de registro del usuario en la aplicación [Elaboración propia]*



*Ilustración 22: Vista de inicio de sesión del usuario en la aplicación [Elaboración propia]*

Una vez habiendo accedido a la aplicación, ya sea por inicio de sesión o por registro en ella, se ofrece la posibilidad de solicitar un nuevo trayecto. Si se hace clic en el botón correspondiente, se puede acceder a la pantalla en la que se muestra un mapa con dos marcadores, un botón que permite acceder a un desplegable y, por último, un botón de confirmación del trayecto para avanzar a la siguiente pantalla. El marcador rojo es el que va a señalar el punto de partida, mientras que el marcador gris identifica las coordenadas del punto de llegada. Estos marcadores se pueden arrastrar, permitiendo colocarlos donde se desee.



Ilustración 25: Vista de solicitud de un trayecto que incluye un mapa con marcadores [Elaboración propia]

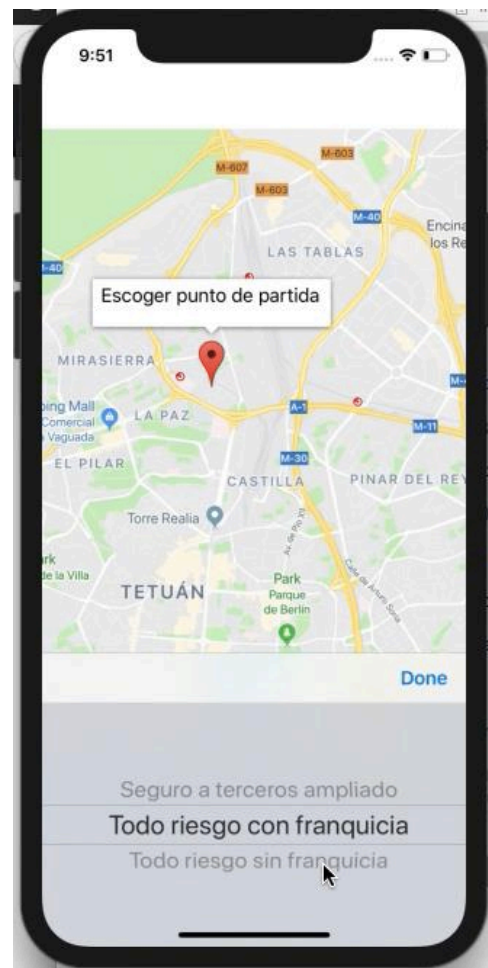
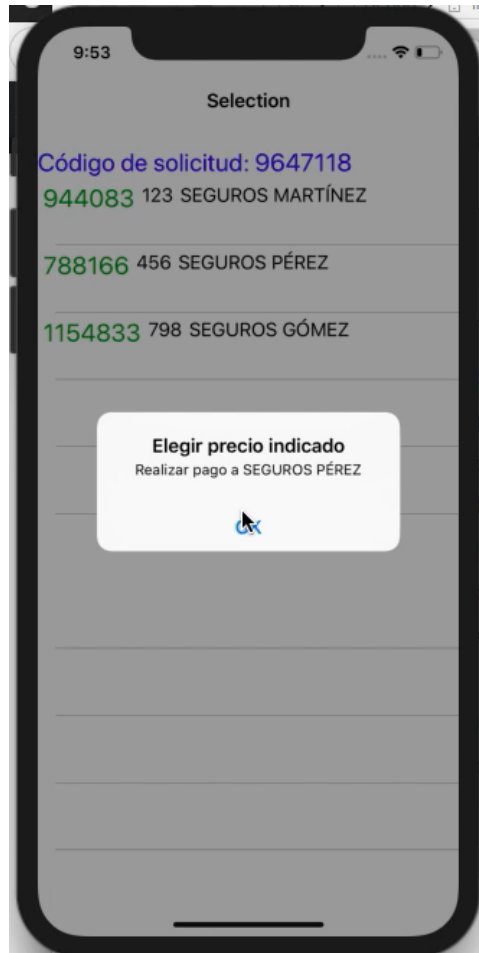


Ilustración 24: Menú desplegable para elegir el tipo de cobertura en el trayecto [Elaboración propia]

Una vez haya seleccionado el tipo de cobertura deseado, se hayan colocado los marcadores y, por último, se haya pulsado sobre el botón de “Confirmar trayecto”, se accede a una pantalla que ofrece una lista de todas las compañías de seguros presentes en la plataforma, con su identificador (un código equivalente al DNI de la compañía), y los precios que dan para el trayecto (en verde). El usuario puede pulsar ahora sobre la opción que desee; aunque en un futuro se añadirían más posibilidades a las ofertas, por ahora, el único factor que se utiliza es el precio ofrecido.

El precio que se muestra en la pantalla, naturalmente, es *ether*, la moneda de *Ethereum*. En concreto se trata de *gwei*, siendo 1 *ether* equivalente a 1 000 000 000 *gwei*. En el momento de la redacción de esta Memoria, 1 *ether* eran aproximadamente 235

euros. De esta manera, el precio medio que se muestra en el ejemplo es de 944 083 *gwei*, equivalente en aquel momento a unos 22 céntimos de euro.



*Ilustración 26: Vista de elección de las compañías y el precio que ofrecen para el trayecto  
[Elaboración propia]*

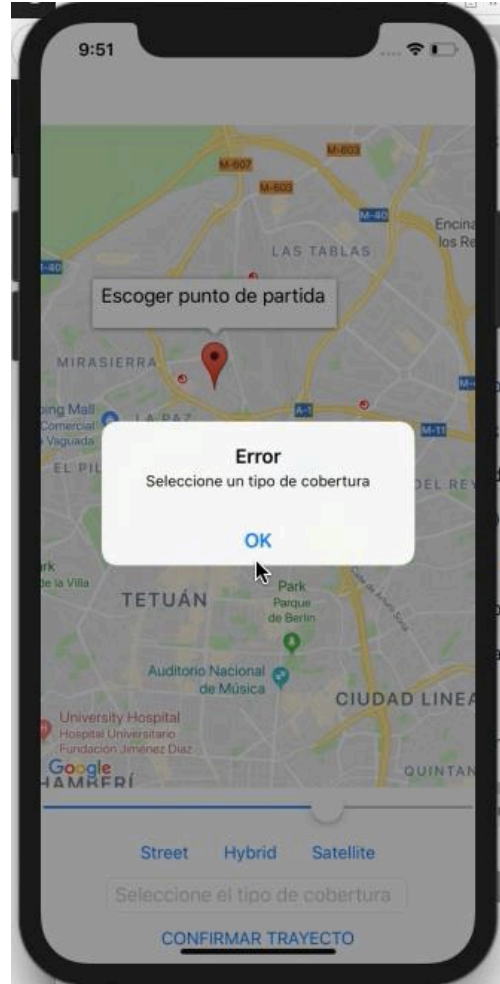
En la lógica de la aplicación de *Xamarin* se han incluido mensajes de error en caso de proceder por vías sin sentido para el correcto funcionamiento. El primer ejemplo es el del mensaje que recibimos cuando intentamos iniciar sesión con un ID de usuario o una contraseña incorrectos, ya que la aplicación, o bien no localiza el ID especificado en el contrato *Report.sol*, o bien, una vez localizado el contrato *ClienteUsuario.sol* que le corresponde, comprueba que la contraseña introducida por pantalla no se trata de la contraseña correcta del usuario. El segundo ejemplo es el del mensaje de error que se



muestra al intentar proceder a confirmar la solicitud de precios de Seguros para un trayecto sin haber especificado previamente el tipo de cobertura deseado.



*Ilustración 28: Mensaje de error que aparece al equivocarse el usuario en el inicio de sesión [Elaboración propia]*



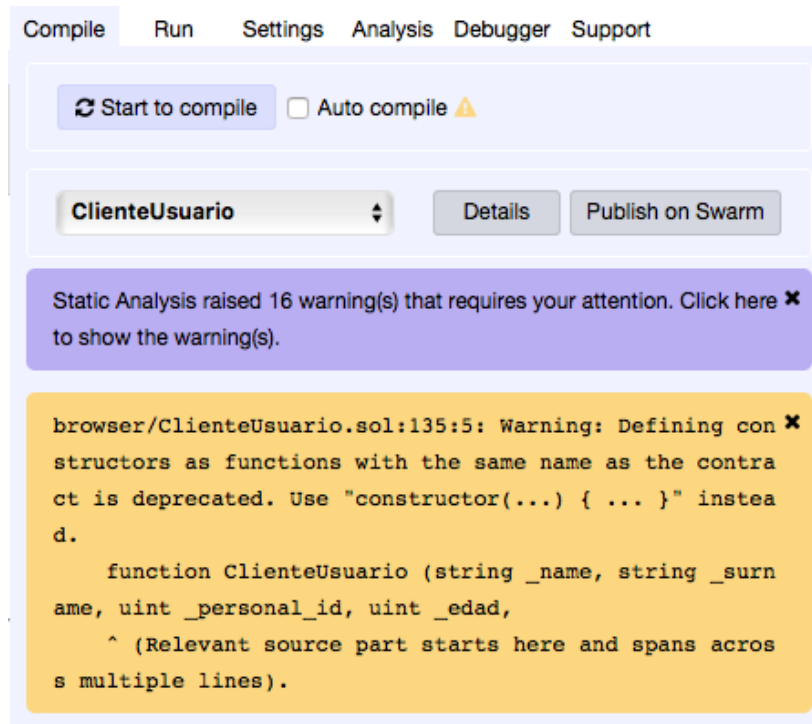
*Ilustración 27: Mensaje de error que aparece al confirmar la solicitud del trayecto sin haber especificado el tipo de cobertura [Elaboración propia]*

La inclusión de la interfaz del mapa en la aplicación ha sido posible gracias a la utilización de una *API* de *Google Maps*. Esta es gratuita y se puede habilitar desde el apartado para desarrolladores de *Google.com*.

### 5.3.2 Entorno de pruebas de *Remix*

Sin duda, una de las herramientas que más ha facilitado el desarrollo de esta aplicación descentralizada ha sido *Remix*. Se trata de un *IDE* basado en navegador web que permite escribir contratos inteligentes en el lenguaje *Solidity*, compilarlos y ejecutarlos. Se trata de una simulación del despliegue de contratos en la Máquina Virtual de *Ethereum* y de la interacción con ellos.

Es un recurso ideal para un Proyecto de estas características ya que, al haber un coste detrás de cada transacción en la red *Ethereum*, facilita mucho los primeros pasos con contratos inteligentes, garantizando que sean gratuitos. Aunque en este caso se trate de una red privada, y nosotros hayamos definido el ether disponible en la cuenta principal de la red, haciendo que dispongamos de mucho para gastar, permite no “despilfarrarlo” en situaciones de prueba y error.



*Ilustración 29: Herramientas de Remix para trabajar con contratos inteligentes [Elaboración propia]*

Una vez desplegados los contratos en direcciones de la cadena simulada, es posible interactuar con ellos mediante botones a la derecha de la pantalla. Se trata de una

interfaz muy intuitiva que permitió la comprensión del funcionamiento de los contratos inteligentes y de la lógica del lenguaje *Solidity* al inicio de este Proyecto.

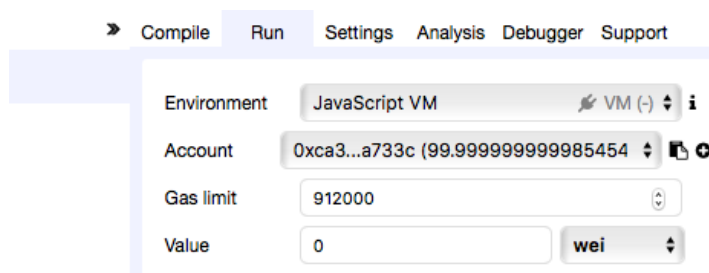
### 5.3.2.1 Problemas encontrados: limite de gas

Recordemos que las transacciones que se realizan en este proyecto son los despliegues de contratos (esto incluye la ejecución de las funciones internas de los constructores) y las colocaciones de la elección del usuario para un trayecto. Aunque no se haya integrado la lógica de los cambios de ponderaciones por parte de las compañías, esto también supondría una transacción.

Esto es así debido a que se trata de funciones que realizan cambios en los contratos, ya que, o bien añaden elementos a vectores propios del contrato, o bien realizan cambios en las variables del contrato (incremento de los índices que permiten el control de los vectores, cambio en las ponderaciones...).

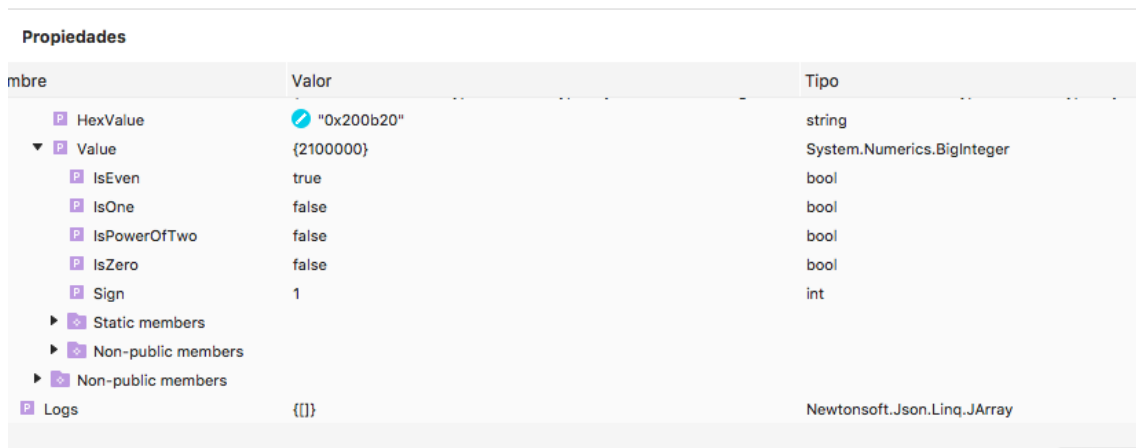
Como hemos visto, en *Ethereum* el envío de una transacción supone un coste de gas. En un principio se han dado numerosos errores que tenían que ver con esta situación, ya que es necesario indicar un límite de gas para la transacción, pues si no, por defecto, este límite es muy bajo, pero es importante indicar una cantidad con sentido, para no malgastar *ether* de la *etherbase* en caso de estar subiendo contratos con código.

*Remix* ha ofrecido la posibilidad de evaluar el coste gas simulándolo en el compilador online, para después colocar en *Xamarin* límites con sentido. Mediante prueba y error, se ha probado en repetidas ocasiones el despliegue de los distintos contratos hasta dar con la cifra aproximada del gas empleado. Así, se ha colocado en nuestro código en *Xamarin*, límites de gas con sentido para cada transacción.



*Ilustración 30: Menú de compilación de contratos inteligentes en Remix [Elaboración propia]*

Por supuesto, a posteriori, es posible comprobar el gas que se ha empleado en la transacción.





Nombre	Valor	Tipo
HexValue	"0x200b20"	string
Value	{2100000}	System.Numerics.BigInteger
IsEven	true	bool
IsOne	false	bool
IsPowerOfTwo	false	bool
IsZero	false	bool
Sign	1	int
Static members		
Non-public members		
Non-public members		
Logs	{[]}	Newtonsoft.Json.Linq.JArray

*Ilustración 31: “Resguardo” que se recibe en Xamarin tras el envío de una transacción  
[Elaboración propia]*

### 5.3.3 Integración de la red con *Xamarin*

Como se ha visto, para interactuar con los contratos inteligentes es necesario contar con la *ABI* en *JSON* del contrato. Para el despliegue de los *Smart Contracts* a la red, se debe tener, además, su *bytecode*. La manera de conseguir estos datos fue mediante la compilación del contrato en *Remix*, para después copiar (desde la ventana de *Details*) y pegar las cadenas de código en *Xamarin*.

WEB3DEPLOY  

```
var reportContract = web3.eth.contract([{"constant":false,"inputs":[{"name":"_codigoSolicitud","type":"uint256"}, {"name":"_adrSolicitud","type":"address"}, {"name":"_personal_id","type":"uint256"}],"name":"anadirClientesolicitudReport","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":false,"inputs":[{"name":"_codigoSolicitud","type":"uint256"}, {"name":"_precio","type":"uint256"}, {"name":"_idCompania","type":"uint256"}],"name":"anadirClienteSolicitudEleccionReport","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":false,"inputs":[{"name":"_idCompania","type":"uint256"}, {"name":"_adrCompania","type":"address"}],"name":"anadirCompaniaReport","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":false,"inputs":[{"name":"posicion_array","type":"uint256"}],"name":"getClientesAtIndex","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":false,"inputs":[{"name":"_idCompania","type":"uint256"}],"name":"getDireccionCompania","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":true,"input
```

*Ilustración 32: Parte del código que contiene la ABI del contrato Report.sol  
[Elaboración propia]*

### 5.3.3.1 Despliegue de Report.sol

Un momento importante en el desarrollo del Proyecto fue el del despliegue del contrato *Report.sol*. Recordemos que, aunque haya numerosas instancias de los contratos *Compania.sol*, *Cliente.sol* y *ClienteSolicitudTrayecto.sol*, el contrato “maestro” es único y sobre el quedan registradas todas las nuevas incorporaciones a la plataforma.

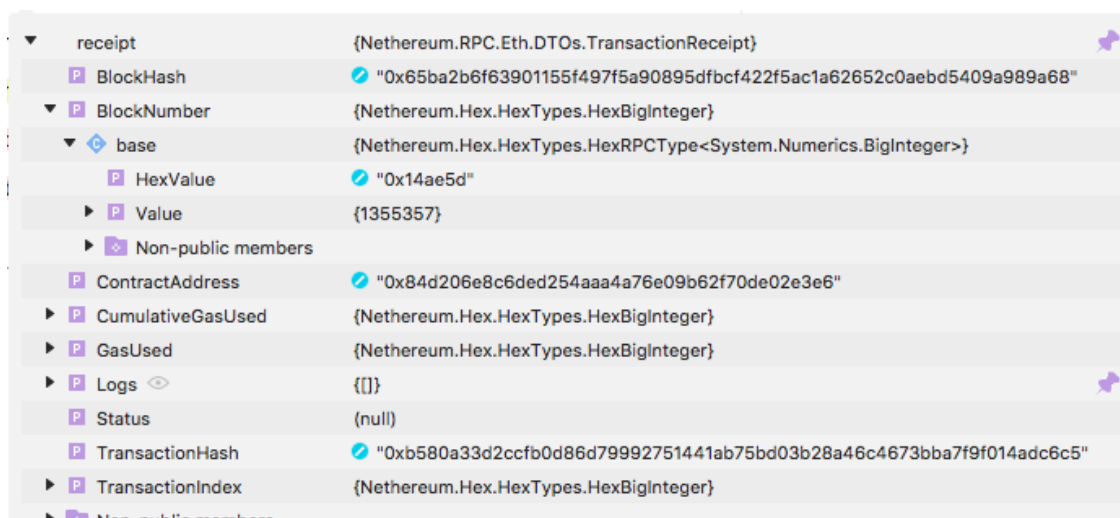
Para el despliegue de este contrato, ya que no pertenece al uso normal de la aplicación (solo se hace una vez), se desarrolló una clase sencilla auxiliar, llamada al lanzarse la aplicación.

```
var senderAddress = "0x0b3eca6dffe4fff33dadcd10dae4cf8c4de489857" ;
var abi = @"[{"constant":false,"inputs":[{"name":"","_codigoSolicitud":"","type":"","uint256":""},{name":"","_adrSolicitud":"","type":"","uint256":""}],"outputs":[{"name":"","type":"","uint256":""}],"payable":true,"stateMutability":"nonpayable","type":"function"}];
var bytecode = "0x608060405234801561001057600080fd5b50610e29806100206000396000f30060806040526000436106100c5576000357c010000";
var web3 = new Web3("http://10.74.125.19:8545/");
Nethereum.Hex.HexTypes.HexBigInteger gass = new Nethereum.Hex.HexTypes.HexBigInteger(1300000);
var listView = new ListView();
var receipt = await web3.Eth.DeployContract.SendRequestAndWaitForReceiptAsync(abi, bytecode, senderAddress, gass, null);
var contractAddress = receipt.ContractAddress;
var contract = web3.Eth.GetContract(abi, contractAddress);
var address = receipt.ContractAddress;
```

*Ilustración 33: Código necesario para el despliegue de un contrato desde Xamarin gracias a Nethereum [Elaboración propia]*

En el momento de desplegar un contrato, existe la posibilidad de “pedir un recibo” de la transacción realizada, lo que permite observar la dirección de la transacción en la cadena de bloques, el número de bloque en que se ha añadido esta, la dirección en la que se encuentra el contrato...

Una vez se dispone de la dirección del contrato *Report.sol*, al ser este único, se ha introducido esta en los tres contratos, permitiendo la comunicación entre contratos y con ello la capacidad de añadir la dupla identificador-dirección en el momento de desplegar cada contrato de cliente, de compañía o de solicitud.



*Ilustración 34: Resguardo que se recibe en Xamarin tras el despliegue del contrato Report.sol [Elaboración propia]*

### 5.3.3.2 Interacción con la lógica de los contratos inteligentes

Se han dividido las clases e interfaces en Modelos, Servicios y Vistas, En lo que respecta a los Servicios, para conectar la lógica de los contratos inteligentes a la aplicación de *Xamarin*, se ha desarrollado una clase “ClienteUsuarioEthereumServices.cs”, en la que se han desarrollado métodos que poseen las funcionalidades equivalentes a los métodos de los contratos. En la interfaz que le corresponde, “IClienteUsuarioEthereumServices.cs” se definen todos sus métodos. Se ha desarrollado, también, una clase auxiliar “AñadirCompañías” para desplegar contratos inteligentes acreditando la entrada de compañías a la plataforma, simulando la función que en un futuro desarrollo realizaría un sitio web dedicado a esto.

Un ejemplo de transacción es el del despliegue de un contrato *ClienteUsuario.sol* en el momento de producirse el registro. El envío de transacciones con Nethereum es posible gracias al método “SendRequestAndWaitForReceiptAsync”.

```
public async Task<string> ReleaseContract_ClienteUsuario(Usuario _usuario)
{
    var unlockAccountResult = await web3.Personal.UnlockAccount.SendRequestAsync(
        senderAddress,
        "password", 120, null);

    var receipt = await web3.Eth.DeployContract.SendRequestAndWaitForReceiptAsync
        (abiClienteUsuario, bytecodeClienteUsuario, senderAddress,
        new Nethereum.Hex.HexTypes.HexBigInteger(gasClienteUsuario) ,
        new Nethereum.Hex.HexTypes.HexBigInteger(10000000), null,
        _usuario.name, _usuario.surname, _usuario.personal_id, _usuario.edad,
        _usuario.numanosconcarnt, _usuario.street, _usuario.country,
        _usuario.password);

    var contract = web3.Eth.GetContract(abiClienteUsuario, receipt.ContractAddress);

    var address = receipt.ContractAddress;

    Console.WriteLine("Desplegado nuevo contrato ClienteUsuario.sol en la dirección: " + address);

    return Convert.ToString(address);
}
```

*Ilustración 35: Función de despliegue de ClienteUsuario.sol utilizando Nethereum desde Xamarin [Elaboración propia]*

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

Un ejemplo de llamada, de consulta, en la que no se altera el estado de la red (no se envían fondos, no se despliegan cuentas ni contratos, no se cambian variables de contrato...) y que, por tanto, no se considera una transacción, es el método que permite calcular el precio que una compañía ofrece al trayecto para, posteriormente, devolverlo por pantalla en la aplicación móvil. Como se puede comprobar, las llamadas se realizan haciendo uso del método “CallDeserializingToObjectAsync”, que permite convertir un objeto de *Solidity* en un objeto de una clase de *Xamarin*.

```
public async Task<CompaniaYPrecio> GetPrecioCompania(string _adressCompania,
                                                    int _personal_id,
                                                    int _edad,
                                                    int _numanosconcarnet,
                                                    float _distancia,
                                                    int _tipoCobertura,
                                                    int _codigoSolicitud)
{
    var unlockAccountResult = await web3.Personal.UnlockAccount.SendRequestAsync(
        senderAddress, "password", 120, null);

    // var addressCompania = direccionContratoCompaniaDesdeReport(_idCompania);

    var contract = web3.Eth.GetContract(abiCompania, _adressCompania);
    var function = contract.GetFunction("calcularYDevolverPrecio");

    Nethereum.Hex.HexTypes.HexBigInteger gass = new Nethereum.Hex.HexTypes.HexBigInteger(1700000);

    var resultadoCompaniaPrecio = await function.CallDeserializingToObjectAsync<CompaniaYPrecio>(
        _personal_id, _edad, _numanosconcarnet, _distancia, _tipoCobertura, _codigoSolicitud);

    CompaniaYPrecio companiaprecio = resultadoCompaniaPrecio;

    return companiaprecio;
}
```

*Ilustración 36: Función de llamada a una función de Compania.sol utilizando Nethereum desde Xamarin [Elaboración propia]*

Para la correcta comunicación con los *Smart Contracts*, se ha desarrollado unos modelos de consulta distintos de los de envío de datos, ya que hay que ser muy precisos para poner en contacto dos lenguajes de programación diferentes.

Por último, al acceder al código fuente se podrá comprobar que la cuenta encargada de despliegue de contratos y de alteración de variables ha sido la etherbase. Esto no sería así en una implementación real de la aplicación, tanto en una red privada de *Ethereum* con muchos *wallets* funcionando, como, por supuesto, en la red pública. Es una tarea que correspondería a los *wallets* de los usuarios y de las empresas.



## 6 Análisis de resultados

En cuanto al servicio que se deseaba conseguir al inicio del Proyecto, la realidad es que se han cumplido los objetivos marcados. Se ha conseguido comprender el funcionamiento de la implementación de la biblioteca *Nethereum* para *.NET*, así como de *Web3*, y a partir de ahí una aplicación para teléfono móvil. Se ha obtenido una aplicación que garantiza la puesta en marcha de los contratos de seguro de manera automática, sin necesidad de un intermediario que no aporte valor, gracias a la confianza en la validación de las transacciones por el consenso de la red, y en la que todos los cambios o movimientos que se realicen pueden ser rastreados y consultados en la cadena de bloques, gracias a la grabación de eventos.

En otras palabras, todos los miembros de la red, que quieran participar de la aplicación que aquí se propone, pueden estar seguros, gracias a los algoritmos de consenso con los que funciona el protocolo de cadena de bloques de *Ethereum*, de que las “reglas del juego” escritas en el código van a ser cumplidas. Los usuarios, así, están depositando su confianza en la propia red y en sus reglas criptográficas, y no en una entidad controladora que tenga más poder que el resto de los miembros de la red, o que pueda ser coaccionado o *hackeado*.

Una vez que se tienen los conocimientos suficientes de la cadena de bloques y se comprenden los principios básicos de la programación en el lenguaje *Solidity*, la realidad es que este Proyecto puede servir de guía para el despliegue de una red de registro distribuido privada que simule el funcionamiento de la que sostiene la Máquina Virtual de *Ethereum*. Como hemos visto, en esta Memoria se pueden encontrar también las indicaciones necesarias para el desarrollo de una serie de contratos inteligentes cuyo código pueda ser subido a la cadena de bloques privada tras realizar las pruebas convenientes en *Remix*. Por último, se facilitan las instrucciones para el acceso a la biblioteca *Nethereum* y el código que permita desplegar contratos desde *Visual Studio* y acceder a las funciones incluidas en estos (tanto transacciones como llamadas).

De este modo, este Proyecto puede utilizarse como base para el desarrollo de todos los tipos de aplicaciones y servicios que ofrece *Visual Studio* y, en especial, *Xamarin* (aplicaciones web, aplicaciones móviles...).

La aplicación móvil no llegó a ser descargada en un teléfono *iPhone* para probar su uso porque el acceso al nodo depende del acceso a la conexión inalámbrica de la compañía en la que he realizado las Prácticas y con la que se propuso el Proyecto, *Management Solutions*, y no era posible acceder a ella sin una serie de permisos que no era posible establecer en el dispositivo móvil (sí en el ordenador sobre el que se desarrolló el Proyecto). Sin embargo, como hemos podido comprobar, el simulador de *XCode* funciona muy bien, ofreciendo una experiencia de mucha calidad, dando la sensación de estar utilizando un teléfono móvil.

### 6.3 Acceso a los eventos

Como se ha explicado, en este Proyecto se han desarrollado los contratos inteligentes de manera que posean la lógica más adecuada, aunque más tarde muchas de las potenciales funcionalidades que ofrece la lógica incluida no hayan sido implementadas en la aplicación móvil. Esto es lo que ocurre con la consulta a los eventos que se han ido grabando desde las funciones. La cadena de bloques se está llenando de los datos que le hemos ido enviando, garantizando la trazabilidad de los datos que interesan a la aplicación, pero en este Trabajo no se ha desarrollado el código para acceder a ellos. De todas formas, esto podría hacerse no solo desde *Nethereum* y *Xamarin*, sino que un miembro de la red podría, en cualquier momento, acceder a la consola de *geth*, conectarse a la red mediante un nodo y ejecutar comandos que le permitiera acceder a los eventos.

Aunque se le ha dado importancia a los eventos en el momento de desarrollar los *Smart Contracts*, tras mucho leer se ha comprendido que realmente son la base de la transparencia ofrecida por Ethereum, por lo que, en un futuro, introduciríamos más eventos y con *topics* complejos que facilitasen el filtrado a la hora de buscar una transacción concreta pasada en la cadena de bloques.

#### 6.4 Posibles mejoras

Como se ha explicado, en el momento de desplegar la cadena de bloques y el nodo, existe la posibilidad de añadir *ether* a una cuenta que creamos. Se trata de la *etherbase* y ha sido la cuenta utilizada para el despliegue de los contratos. Recordemos que un contrato inteligente no puede enviar transacciones. Únicamente tienen esta capacidad las cuentas de la red.

Es por ello por lo que la idea de la aplicación es que el contrato *Cliente.sol* lleve un *wallet* de *Ethereum* asociado. De este modo, hubiera sido posible añadir a la aplicación la lógica de los conocidos como *modifiers*. Estos permiten que una función que suponga una transacción en la red (cambios en variables de contrato, funciones con pagos entre contratos...) tenga un acceso limitado y dependiente de la cuenta que envía la transacción.

```
modifier onlyOwner{  
  
    require(msg.sender==adrClienteUsuario);  
    -;  
}
```

*Ilustración 37: Ejemplo de modifier sencillo que comprueba que la cuenta que lance una transacción particular sea la misma que lanzó la del despliegue del contrato [Elaboración propia]*

De este modo, es importante remarcar que, aunque en todo momento utilizamos la cuenta “madre” *etherbase* por que es aquella de la que estamos seguros de que tiene *ether* en todo momento (recordemos que es la que recibe la compensación a cambio del trabajo computacional que el nodo está continuamente ofreciendo a la red al añadir transacciones a bloques y estos a la cadena), la implementación natural de la lógica de la aplicación debería ser distinta. Una solución hubiera podido ser crear cuentas con *ether* en el momento de *nacer* y, de manera ficticia, haberlos asociado a identidades digitales de usuarios presentes en la red *Ethereum* antes de haber decidido incorporarse a nuestra aplicación en particular dentro de esta red.

Hubiera sido interesante explorar estas posibilidades, pues son una gran herramienta que ofrece el lenguaje de programación Solidity para sus contratos inteligentes.

#### 6.5 La desactivación del contrato temporal en el trayecto

Sin duda, existen otras funcionalidades que podrían ser añadidas, ya que en este Proyecto se han desarrollado contratos inteligentes sencillos que garantizan el funcionamiento básico. En este caso, recordemos, el contrato *ClienteSolicitudTrayecto.sol* se despliega en el momento que el usuario cliente solicita precios a las compañías de la plataforma para estar asegurado en un trayecto que va a realizar, activándose en el momento en que elige una compañía y el precio que esta le ofrece de forma automática. El siguiente paso, sería desarrollar la funcionalidad de desactivación del seguro, de dejar de estar legalmente asegurado como conductor en el momento de terminar el trayecto, o, incluso de cancelar el vínculo legal si el asegurado realiza una serie de acciones indicadas en el contrato del Seguro.

Los avances en tecnología *blockchain* suelen ir de la mano de lo que se conoce como el Internet de las cosas (en inglés, se utilizan las siglas *IoT*, de *Internet of Things*), lo que, básicamente, hace referencia a las nuevas tecnologías centradas en conectar los objetos de la vida cotidiana a Internet y dotarlos de dispositivos de identificación. Un pequeño dispositivo telemático que pudiera ir incluido en el coche, recogiendo datos de la conducción del vehículo y ofreciéndolos a la plataforma, no solo ofrecería la posibilidad de dotar a la “ecuación” de cálculo de precios de más parámetros y por tanto más precisión en la personalización de estos, sino que además permitiría de manera sencilla enviar un mensaje a la aplicación -en tiempo real con *GPS*, por ejemplo-, de notificación de que el coche ha llegado al destino.

Ya existen seguros con cláusulas conocidas como *Pay As You Drive*, que recogen con dispositivos datos del coche para personalizar las pólizas e incluso dar beneficios a los “buenos conductores”. La incorporación de ambos campos (tecnologías de cadena de bloques y componentes conectados) garantizaría al completo la seguridad de la aplicación

descentralizada y con ello la confianza en ella. Permitiría, de este modo, añadir condiciones, a los contratos haciendo uso de las variables recogidas del coche. Especialmente, ofrecería mejoras para la gestión del momento de pago.

Recordemos que, aunque no se ha implementado esta lógica, al no contar con *wallets* más allá de la *etherbase*, el funcionamiento de la aplicación consistiría en la activación, en el momento de cumplirse una serie de condiciones recibidas del dispositivo telemático, de la función “pagarContrato” de *ClienteUsuario.sol*. Esta se encargaría de transferir la cantidad que estuviera indicada en el precio del contrato *ClienteSolicitudTrayecto.sol* correspondiente a la última dirección añadida a su vector de direcciones “trayectosQueHeContratado”, lo que se restaría al balance del contrato.

```
function pagarContrato()
payable // payable so it accepts value: from sender
returns(bool success)
{
    uint codigoSolicitud= trayectosQueHeContratado[indextrayectoQueHeContratado];
    uint precio =clienteusuarioSolTrayAdr[codigoSolicitud].precio;
    if(msg.value != precio) throw;

    address adrSolicitud = clienteusuarioSolTrayAdr[codigoSolicitud].adrSolicitud;
    registrarPago(
        adrSolicitud,
        "clienteusuarioSolTrayAdr[codigoSolicitud].idCompania",
        msg.value);

    adrSolicitud.transfer(msg.value); // sends the funds

    return true;
}
```

*Ilustración 38: Función “pagarContrato” del contrato ClienteUsuario.sol*  
*[Elaboración propia]*

Así, el contrato *ClienteSolicitudTrayecto.sol*, correspondiente al trayecto recibiría la cantidad del precio vía su método “pagarCompany”, pues es el que recibe los pagos, y sería el que reenviase, o no, esta cantidad a la dirección del *wallet* asociado a la compañía escogida para el trayecto, en función de si el coche ha llegado a su destino, o no. Se podría, por supuesto, aumentar la complejidad de las condiciones.

```
function pagarCompany()
  payable // payable so it accepts value: from sender
  returns(bool success)
{
  if(msg.value != clientDataSolicitud[address(this)].precio) throw;

  address adrCompany =report.getAddressCompania(
    clientDataSolicitud[address(this)].idCompania
  );

  registrarPago(
    adrCompany,
    clientDataSolicitud[address(this)].idCompania,
    msg.value);
  adrCompany.transfer(msg.value); // sends the funds to the seller

  return true;
}
```

*Ilustración 39: Función “pagarCompany” del contrato ClienteSolicitudTrayecto.sol*  
*[Elaboración propia]*

En el código podemos comprobar cómo el método da error si el precio que está recibiendo la función (es de tipo *payable*) no coincide con el que el contrato sabe que está “hablado” y que debería estar recibiendo. Es importante entender que los fondos se están guardando en el propio contrato inteligente entre el momento en que se empieza a ejecutar la función y el código llega al comando transfer. Se podrían incluir cientos de condiciones relacionadas con el dispositivo telemático que reconoce parámetros del coche en tiempo real, permitiendo establecer condiciones claras de envío o devolución de dinero en una dirección u otra. Es decir, se podría incluir el pago automático e instantáneo, por parte de la compañía y con destino el *wallet* del cliente, correspondiente a las condiciones de la cláusula. Por ejemplo, si un dispositivo telemático registra un siniestro total en un coche en el trayecto en el que una aseguradora y un usuario se han comprometido con un contrato a todo riesgo, se pagaría la cantidad reflejada en el *Smart Contract* al *wallet* del usuario, automáticamente.

## 7 Conclusiones y trabajos futuros

Se ha podido concluir que la tecnología *blockchain* sí tiene sentido con los intereses que se buscaban en la aplicación particular de Seguros ideada al comienzo de este Proyecto. La componente legal que acompaña de forma intrínseca a los servicios ofrecidos por las compañías de Seguros obliga a ir necesariamente con cuidado en ciertos aspectos. Sin embargo, “al permitir crear registros inmutables y hacer un seguimiento de un documento o una cadena de sucesos, la *blockchain* permite [...] verificar la autenticidad de cualquier documento que haya sido registrado en ella, eliminando la necesidad de que una autoridad centralizada o tercero lo certifique”. [9]

Como se ha podido explicar, los casos de uso en los que se pueda hacer uso de las herramientas que ofrece *Ethereum* son infinitos. La aplicación en este Proyecto de esta tecnología de cadena de bloques al sector particular de los Seguros de automóvil muestra solamente una de sus muchas posibilidades. Como es natural, hay muchas mejoras que se han planteado al Proyecto, y que quedan pendientes como trabajos futuros.

### 7.1 La identidad digital

Se está investigando mucho acerca de las posibilidades de utilizar tecnologías de registro distribuido para desarrollar verdaderas identidades digitales, a prueba de manipulación. Se están planteando, así, las posibilidades de que, en un futuro próximo, estas sustituyan a los nombres de usuario y contraseña en línea. Esto permitiría utilizar esta *identidad blockchain* para acceder a aplicaciones, sitios web, firmar documentos de forma digital...

La evolución natural de esta aplicación pasa por incorporar este tipo de ideas, y que, como miembro de la red *Ethereum*, identidad digital en mano, un usuario sea capaz de iniciar sesión en la aplicación particular de Seguros de coche con las credenciales que correspondan, del mismo modo que iniciarás sesión en otra aplicación distinta que también trabaje sobre esta red. Este sistema reemplazaría al actual, en el que los usuarios somos poseedores de numerosas cuentas con distintas direcciones de correo, distintas contraseñas... para cada aplicación. “La identidad digital de individuos facilita un

proceso de embarque en la aplicación instantáneo y también la simplificación de la contratación de servicios.” [16]

Del mismo modo, como miembro de la red *Ethereum*, la idea es que un usuario sea poseedor de un *wallet* con el que sea capaz de realizar los distintos pagos que requieren algunas aplicaciones, como ocurre con esta. La lógica de los contratos inteligentes desarrollados en esta aplicación está pensada para que sea esta cuenta la que pague al contrato *ClienteSolicitudTrayecto.sol* en el momento de activar el Seguro para el trayecto.

Una vez se haya explorado la tecnología *blockchain*, precisamente con Proyectos como este, en un futuro en el que los pagos que no pasen necesariamente a través de entidades bancarias estén a la orden del día el que los contratos inteligentes sean legalmente vinculantes, esta aplicación descentralizada será más segura que la equivalente centralizada, ya que, entre otras cosas, permitirá verificar la autenticidad de los documentos digitales que correspondan a los contratos, eliminando la necesidad de que un tercero lo certifique.

## 7.2 La red Alastria

Por supuesto, la seguridad del sistema reside en que haya un número suficiente de nodos con una potencia de cálculo elevada minando para generar nuevos bloques. El próximo paso en este Proyecto, buscando estar seguros de la conveniencia de ciertas partes del código de los contratos y de la manera como está definida la aplicación en general, sería la incorporación de la aplicación a una red *blockchain* semipública, como es *Alastria*. Se trata del “primer Consorcio multisectorial promovido por empresas e instituciones para el establecimiento de una infraestructura semipública *Blockchain/DLT*, que soporte servicios con eficacia legal en el ámbito español y acorde con la regulación europea”. [33]

El objetivo de *Alastria* consiste en la habilitación de un “espacio digital común para sus miembros en el que estos desarrollen sus servicios y productos *blockchain/DLT*”. El consorcio se define a sí mismo como abierto, multisectorial (cuenta con habilitar “la infraestructura tecnológica que soporte los modelos más avanzados de *Smart Contracts*



con un enfoque completamente multisectorial y con garantía de privacidad”) e independiente (se trata de una red “independiente y neutral sobre la que los socios podrán lanzar sus productos y servicios; [...] la plataforma no tiene modelo de negocio”).

Precisamente a esta red está conectado el primer nodo universitario blockchain en España, desarrollado por esta Universidad. “Es la primera que realizará experimentación educativa en *blockchain*, por lo que ya está conectada al nuevo nodo de la Universidad San Pablo CEU”. Se cree que el uso de esta tecnología “cambiará la manera en que los alumnos se relacionan con la universidad, y la que tienen las universidades de interactuar entre ellas”. [34]

El mercado se divide entre escépticos y apasionados de las tecnologías de registro distribuido, especialmente de las que trabajan con *blockchain*. Es frecuente encontrar el argumento que defiende que, aunque el código de los contratos inteligentes de *Ethereum* es público (es decir, la definición del funcionamiento y reglas de las aplicaciones), no todo el mundo tiene los conocimientos de programación para entender con total certeza sus condiciones y posibilidades. Es por esto por lo que se está trabajando en el desarrollo de lenguajes funcionales que faciliten la lectura y comprensión del código para los menos entendidos.

En definitiva, es cierto que se trata de tecnologías que se encuentran en el momento de salir del laboratorio, pero en el momento en que tenga una base fuerte y hayan sido probadas con un importante número de casos de uso, no cabe duda de que demostrarán todo su potencial. El de este Proyecto es solo un caso de los muchos que son realizables y que tienen sentido. Se trata de un ejemplo de acercamiento de las tecnologías de registro distribuido al usuario medio, facilitando su comprensión por parte de este y ofreciendo una guía para el desarrollo y la implementación de una aplicación descentralizada.

## Bibliografía

- Ethereum.org, <https://www.ethereum.org/ether>
- Satoshi Nakamoto. Bitcoin White Paper, <https://bitcoin.org/bitcoin.pdf>
- Vitalik Buterin. Ethereum White Paper, <https://github.com/ethereum/wiki/wiki/White-Paper>
- Github.com. Ethereum Development Tutorial, <https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial>
- Solidity-es. “Introducción a los Contratos Inteligentes”, <https://solidity-es.readthedocs.io/es/latest/introduction-to-smart-contracts.html>
- GitHub. Ethereum, Go-Ethereum, <https://github.com/ethereum/go-ethereum/wiki/geth>

## Referencias

- [1] Estamos Seguros. “Entrevista: Así cambiará el *blockchain* el mundo de los seguros”, <https://www.estamos-seguros.es/entrevista-asi-cambiara-el-blockchain-el-mundo-de-los-seguros/>
  
- [2] Solidity-es. “Introducción a los Contratos Inteligentes”, <https://solidity-es.readthedocs.io/es/latest/introduction-to-smart-contracts.html>
  
- [3] Coindesk.com. “What is a Distributed Ledger”, <https://www.coindesk.com/information/what-is-a-distributed-ledger/>
  
- [4] Proyecto Final de Carrera de Antonio Marín Bermúdez, “Estudio de la utilización de protocolos *blockchain* en sistemas de votación electrónica”, [https://upcommons.upc.edu/bitstream/handle/2117/98545/PFC%20Blockchain\\_Evoting\\_v01.pdf?sequence=1&isAllowed=y](https://upcommons.upc.edu/bitstream/handle/2117/98545/PFC%20Blockchain_Evoting_v01.pdf?sequence=1&isAllowed=y)
  
- [5] Indra. Redes P2P, <https://www.indracompany.com/es/blogneo/peer-peer-p2p-modelos-negocio-ciudadanos>
  
- [6] Microsoft Azure. Blockchain, “Desarrolle, pruebe e implemente aplicaciones de cadena de bloques seguras”, <https://azure.microsoft.com/es-es/solutions/blockchain/>
  
- [7] Eric Wall y Gustaf Malm. “Using *Blockchain* Technology and *Smart Contracts* to Create a Distributed Securities Depository”, <http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8885750&fileOId=8885765>
  
- [8] Diccionario de la Real Academia Española.

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

- [9] Fin-tech.es. “15 aplicaciones de la tecnología *blockchain* más allá de *Bitcoin*”,  
<https://www.fin-tech.es/2016/10/aplicaciones-de-la-tecnologia-blockchain.html>
- [10] Vitalik Buterin. Ethereum White Paper,  
<https://github.com/ethereum/wiki/wiki/White-Paper>
- [11] Satoshi Nakamoto. Bitcoin White Paper, <https://bitcoin.org/bitcoin.pdf>
- [12] Blogs.iadb.org. Puntos sobre la i, “*Blockchain*, la revolución de la confianza encriptada”,  
<https://blogs.iadb.org/puntossobrelai/2017/12/21/blockchain-la-revolucion-de-la-confianza-encriptada/>
- [13] El País. Retina, “Qué es un minero de bitcoin... y por qué llegas tarde al negocio”,  
[https://retina.elpais.com/retina/2017/07/28/tendencias/1501236974\\_154734.html](https://retina.elpais.com/retina/2017/07/28/tendencias/1501236974_154734.html)
- [14] Ethereum.org, <https://www.ethereum.org/ether>
- [15] Sap.com. "Blockchain y tecnología de libros contables distribuidos".  
<https://www.sap.com/spain/products/leonardo/blockchain.html>
- [16] Criptonoticias. “Blockchain permitirá contratar micro seguros desde el coche para desplazamientos concretos”  
<https://www.criptonoticias.com/entrevistas/blockchain-permitira-contratar-micro-seguros-coche-desplazamientos-concretos/>
- [17] Github.com. Ethereum Development Tutorial,  
<https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial>
- [18] MiEthereum. Solidity, “Todos los recursos sobre el lenguaje de programación de los Smart Contracts.”, <https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial>

- [19] Bit2me. “Smart Contracts: Una guía para principiantes”,  
<https://blog.bit2me.com/es/smart-contracts/>
- [20] Expansión. “El blockchain permitirá ahorrar a las aseguradoras de automóviles hasta un 12% en siniestros”, <http://www.expansion.com/economia-digital/innovacion/2018/02/17/5a88084e268e3e561c8b463d.html>
- [21] Expansión. “Las mejores aplicaciones móviles de los seguros de coche”,  
<http://www.expansion.com/ahorro/2016/08/06/57a1b519268e3e9f038b45e2.html>
- [22] Fin-Tech. “B3i: 5 grandes aseguradoras europeas se alían para estudiar la tecnología blockchain”, <https://www.fin-tech.es/2016/10/b3i-alianza-aseguradoras-europeas-blockchain.html>
- [23] El Confidencial. “Movilidad compartida, la apuesta de las nuevas generaciones para el futuro”, [https://brands.elconfidencial.com/sociedad/2018-08-23/movilidad-ciudades-coche-compartido-bra\\_1606512/](https://brands.elconfidencial.com/sociedad/2018-08-23/movilidad-ciudades-coche-compartido-bra_1606512/)
- [24] Voz Pópuli. “Emov permite eliminar el seguro de 500 euros pagando un euro más por trayecto”, [https://www.vozpopuli.com/economia-y-finanzas/empresas/Emov-permite-eliminar-seguro-500-euros-pagando-un-euro-mas-por-trayecto\\_0\\_1106289696.html](https://www.vozpopuli.com/economia-y-finanzas/empresas/Emov-permite-eliminar-seguro-500-euros-pagando-un-euro-mas-por-trayecto_0_1106289696.html)
- [25] BeCode, <https://medium.com/@vanderstraeten.thomas/a-crypto-fundraising-for-a-charity-on-the-ethereum-net-with-a-strava-gps-oracle-8a24167c1dad>
- [26] GFT. “Blockchain y criptomonedas... ¿realidad o exageración?”,  
[https://blog.gft.com/es/2018/04/13/blockchain-y-criptomonedas-realidad-o-exageracion/?\\_ga=2.213839255.2088514140.1534082636-1471440005.1529692436](https://blog.gft.com/es/2018/04/13/blockchain-y-criptomonedas-realidad-o-exageracion/?_ga=2.213839255.2088514140.1534082636-1471440005.1529692436)
- [27] Paradigma Digital. “Smart Contracts: ¿algo nuevo o más de lo mismo?”,  
<https://www.paradigmadigital.com/techbiz/smart-contracts-algo-nuevo-o-mas-de-lo-mismo/>

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

- [28] Albert Palau. Medium, “Desarrollando sobre Ethereum. Cómo configurar un cliente”, <https://medium.com/@albpalau/desarrollando-sobre-ethereum-cómo-configurar-un-cliente-16968601170b>
  
- [29] GitHub. Ethereum, Go-Ethereum, <https://github.com/ethereum/go-ethereum/wiki/geth>
  
- [30] Wikipedia. Interfaz de programación de aplicaciones, [https://es.wikipedia.org/wiki/Interfaz\\_de\\_programación\\_de\\_aplicaciones](https://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones)
  
- [31] GitHub. Ethereum, JSON-RPC, <https://github.com/ethereum/wiki/wiki/JSON-RPC#json-rpc-api>
  
- [32] Wikipedia. ASP.NET, <https://es.wikipedia.org/wiki/ASP.NET>
  
- [33] Alastria.io. <https://alastria.io/#1>
  
- [34] Universidad Pontificia Comillas. “Comillas presenta el primer nodo universitario blockchain”, <https://www.comillas.edu/es/noticias-comillas/16882-comillas-presenta-el-primer-nodo-universitario-blockchain>
  
- [35] Ledger Projects. Bitcoin, “What is a fork? Why is it dangerous?”, <http://www.ledgerprojects.com/tag/bitcoin/>
  
- [36] Ethereum blog. “How to build server less applications for Mist”, <https://blog.ethereum.org/2016/07/12/build-server-less-applications-mist/>

## Parte II: CÓDIGO FUENTE

### 1 Código de *ClienteUsuario.sol*.

```
//Contrato de cliente
contract ClienteUsuario{

//////////VARIABLES Y ESTRUCTURAS

//variables propias del contrato

address adrClienteUsuario = 0x0;

//Estructura que contiene datos del usuario en el momento del registro
(identificativos + interesantes estáticos)

struct ClientPersonal {

    address adr;
    string name;
    string surname;
    uint personal_id;
    uint edad;
    uint numanoscon carnet;

    string street;
    string country;

    string password;

    uint256 registroDate;

}

struct ClienteUsuarioSolicitudTrayecto {

    uint codigoSolicitud;

    /////

    address adrSolicitud;

    ///

    uint idCompania;

    ///
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
uint256 creationDateSolicitud;

uint precio;
uint256 creationDateEleccion;
string confirmado;

}

uint256 indextrayectoQueHeContratado;

uint[] public trayectosQueHeContratado;

//Mapeo de las estructuras con la direccion del contrato(cliente)
mapping (address => ClientPersonal) public clientDataPersonal;
mapping (uint => ClienteUsuarioSolicitudTrayecto) public
clienteusuarioSolTrayAdr;

event status(address adr, string estado, uint256 time);

////////////////////////////////////

modifier onlyOwner{

    require(msg.sender==adrClienteUsuario);
    _;
}

////////////////////////////////////

event pago(address adr, string nombre, string signo,
uint amount, uint256 time, uint balance);

function registrarPago(address adr, string name, uint amount){

    pago(adr, name, "+", amount, block.timestamp, this.balance);
}

////////////////////////////////////CONSTRUCTOR

//Esta funcion se ejecuta al crear el contrato, añadiendo toda la
informacion del cliente

function ClienteUsuario (string _name, string _surname, uint _personal_id,
uint _edad,
uint _numanosconcarnet, string _street, string _country, string _password)
{

    addClientePersonal(_name, _surname, _personal_id, _edad,
_numanosconcarnet,
_street, _country, _password);
}
```



**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
        addClienteReport(_personal_id, address(this));

        adrClienteUsuario= msg.sender;
    }

////////////////////////////////////

        //INICIO DE GESTIONES PARA EMPEZAR A GOBERNAR EL REPORTE.SOL
////////////////////////////////////

        //Si tenemos el contrato reporting en otro documento, se hace de la
siguiente manera:

        Report report = Report(0x412bedd745f41cece781096836fec53a4908ffc7); //
es la buena 18/08

        //FINAL DE GESTIONES PARA EMPEZAR A GOBERNAR EL REPORT.SOL
////////////////////////////////////

////////////////////////////////////

////////////////////////////////////FUNCIONES PARA AÑADIR AL REPORT PARTE DE TODA LA INFO QUE
TENEMOS

        function addClienteReport(uint _personal_id, address _adrClient) private{

            report.anadirClienteReport(_personal_id, _adrClient);
        }

////////////////////////////////////

        //ME INTERESA
        //funcion que añade AL PROPIO CONTRATO los datos personales recibidos

        function addClientePersonal(string _name, string _surname, uint
_personal_id, uint _edad, uint _numanoscon carnet, string _street, string
_country, string _password) private{

            //ESTA ES LA MANERA DE RELLENAR UNA ESTRUCTURA (DE MAPEO) ENTERA, SIN
IR COLUMNA A COLUMNA

            clientDataPersonal[address(this)] = ClientPersonal({

                adr: address(this),
                name : _name,
                surname: _surname,
                personal_id: _personal_id,
                edad: _edad,
                numanoscon carnet: _numanoscon carnet,

                street: _street,
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
country: _country,
password: _password,

//fecha de creacion del contrato
registroDate : block.timestamp

});

}

function pagarContrato()
payable // payable so it accepts value: from sender
returns(bool success)
{

uint codigoSolicitud=
trayectosQueHeContratado[indextrayectoQueHeContratado];
uint precio =clienteusuarioSolTrayAdr[codigoSolicitud].precio;
if(msg.value != precio) throw;

address adrSolicitud =
clienteusuarioSolTrayAdr[codigoSolicitud].adrSolicitud;
registrarPago(
    adrSolicitud,
    "clienteusuarioSolTrayAdr[codigoSolicitud].idCompania",
    msg.value);

adrSolicitud.transfer(msg.value); // sends the funds

return true;
}

////////// FUNCIONES QUE SE SOLICITAN DESDE FUERA, CON INTENCION
DE CONSULTAR INFO DE LAS ESTRUCTURAS, pasando SOLO el id de mapeo

//HE TENIDO QUE QUITAR STREET, PROVINCE Y TAL PORQUE ME DABA ERROR
PORQUE DECIA QUE HABIAN MUCHAS VARIABLES LOCALES
function getPersonalDataClient(address _adrClient) returns (string
name, string b, uint c, uint d, uint e, uint256 k) {

return (clientDataPersonal[_adrClient].name,
clientDataPersonal[_adrClient].surname,
clientDataPersonal[_adrClient].personal_id,
clientDataPersonal[_adrClient].edad,
clientDataPersonal[_adrClient].numanosconcarnet,
clientDataPersonal[_adrClient].registroDate
//OBIAMENTE AQUI NO MUESTRO LA CONTRASEÑA
);

}

function getPersonalDataClientExtra(address _adrClient) returns (string
name, string b, uint c, string f , string g,uint256 k) {
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
        return (clientDataPersonal[_adrClient].name,
clientDataPersonal[_adrClient].surname,
clientDataPersonal[_adrClient].personal_id,
        clientDataPersonal[_adrClient].street,
clientDataPersonal[_adrClient].country,
        clientDataPersonal[_adrClient].registroDate
        );
    }

    function getPassword(address _adrClient) onlyOwner returns (string name,
string b, uint c, string PW) {

        return (clientDataPersonal[_adrClient].name,
clientDataPersonal[_adrClient].surname,
clientDataPersonal[_adrClient].personal_id,
        clientDataPersonal[_adrClient].password
        //AQUI SI LA MUESTRO
        );
    }

    function checkPassword(address _adrClient, string _password)
        returns (bool coinc) {

        bool coincide=false;

        if(keccak256(_password)
            ==keccak256(clientDataPersonal[_adrClient].password)){

            coincide=true;
        }
        else{
            coincide=false;
        }

        return (coincide);

    }

    ///LLAMADA DESDE CLIENTESOLICITUDTRAYECTO.SOL

    function anadirEleccionAClienteUsuario(uint _codigoSolicitud, address
_adrSolicitud, uint256 _precio, uint _idCompania) {

        clienteusuarioSolTrayAdr[_codigoSolicitud].codigoSolicitud =
_codigoSolicitud;
        clienteusuarioSolTrayAdr[_codigoSolicitud].adrSolicitud =
_adrSolicitud;
        clienteusuarioSolTrayAdr[_codigoSolicitud].precio = _precio;
        clienteusuarioSolTrayAdr[_codigoSolicitud].idCompania = _idCompania;

        indextrayectoQueHeContratado = trayectosQueHeContratado.length++;
trayectosQueHeContratado[indextrayectoQueHeContratado]=_codigoSolicitud;
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
    }

    //////////////////////////////////////////////////

}

//INICIO DE TODO LO QUE HAY QUE AÑADIR PARA QUE FUNCIONE LA COMUNICACION CON
REPORTCLIENT.SOL (en general, SIN entrar en INSTANCIAS)

//Si queremos comunicarnos con el smartcontract cliente, debemos incluir en el
mismo documento.sol el smartcontract pero únicamente el constructor y
//la definición de las funciones a las que vamos a hacer referencia, NO es
necesario incluir todo el código de lógica

//Contrato de reporting

contract Report{

    uint256 indexClienteUsuario;

    struct ClienteAddress{

        uint personal_id;

        address adr;

    }

    mapping (uint => ClienteAddress) public clienteAdr;

    address[] numClientesUsuarios;

    function anadirClienteReport(uint _personal_id, address _adr){

        clienteAdr[_personal_id].adr = _adr;
        clienteAdr[_personal_id].personal_id = _personal_id;

        indexClienteUsuario = numClientesUsuarios.length++;
        numClientesUsuarios[indexClienteUsuario]=_adr;

    }

}

//FINAL DE TODO LO QUE HAY QUE AÑADIR PARA QUE FUNCIONE LA COMUNICACION CON
REPORTCLIENT.SOL (en general, SIN entrar en INSTANCIAS)
```

## 2 Código de *Compania.sol*

```
//SmartContract que se crea cada vez que una compañía se da de alta en el
sistema

contract Compania{

    uint256 indexPrecioDevuelto;
    uint256 indexContratoActivado;

    uint pond_edad = 2;
    uint pond_numanoscon carnet = 2;
    uint pond_distancia = 2;
    uint pond_tipoCobertura = 1;

    struct CompaniaData{

        string nombreCompania;
        uint idCompania;
        address adrCompania;
        string pass;

    }

    struct CompaniaSolicitudTrayecto{

        uint codigoSolicitud;

        uint distancia;
        string tipoCobertura;

        uint personal_id;
        uint edad;
        uint numanoscon carnet;

        uint256 creationDateSolicitud;

        uint256 precio;
        string confirmado;

    }

    uint[] public numPreciosDevueltos;
    uint[] public numContratosActivados;

    event statusComp(address adr, string estado, uint256 time);

    event pago(address adr, string name, string signo, uint amount, uint256
time, uint balance);

    //////////////////////////////////////
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
////////// MODIFIERS

modifier onlyCompania{

    require(msg.sender==address(this));
    _;
}

//////////

//Esta función es el "constructor", SIEMPRE se debe llamar igual que el
contract, se ejecuta al desplegar el contrato y puede recibir argumentos
//a la hora de desplegarse (en este caso: _id, _nombre, pass). Si queremos
que nuestro SmartContract pueda recibir ether, el constructor
//debe ser de tipo "payable". Dentro del constructor podemos llamar a más
funciones.

function Compania(uint _idCompania, string _nombreCompania, string pass) {

    addCompania(_nombreCompania, _idCompania, pass); //guarda la
información de la compañía

    addCompaniaReport(_idCompania, address(this)); //guarda el
identificador y la dirección del smartcontract en el contrato maestro

}

mapping (address => CompaniaData) public companiaDatos;
mapping (uint => CompaniaSolicitudTrayecto) public compSolTrayAdr;

////////////////////////////////////TROZO CON INFO SOBRE EL PAGO

//Este evento nos permite grabar todos los pagos en blockchain, se
grabará en un bloque determinado y con los parámetros que definamos.
//Después podremos recuperar un log de eventos de pago desde node js

function registrarPago(address adrCompania, string nombreCompania, uint
amount){

    pago(adrCompania, nombreCompania, "+", amount, block.timestamp,
this.balance);
}

////////////////////////////////////

//el sc va a recibir ether
function() payable {

    //AQUI SE INTRODUCIRIA LA LOGICA DE RETIRADA DE FONDOS ALMACENADOS EN
EL CONTRATO
    //POR PARTE DEL WALLET QUE LO DESPLEGO, CON AYUDA DEL MODIFIER
onlyCompania

}

}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
//función que guarda la información de la compañía a una estructura de la
instancia de SC compañía

function addCompania(string _nombreCompania, uint _idCompania, string
_pass) private{

    companiaDatos[address(this)] = CompaniaData({

        adrCompania: address(this),
        nombreCompania : _nombreCompania,
        idCompania: _idCompania,
        pass: _pass

    });

}

//Referencia a smartcontract Report para poder almacenar la información de
la nueva compañía en el contrato "Maestro"

//Esta dirección hash la debemos cambiar si desplegamos un nuevo
smartcontract Report

Report report = Report(0x412bedd745f41cece781096836fec53a4908ffc7); // es
la buena 16/08

function addCompaniaReport(uint _idCompania, address _adrCompania) private
{

    report.anadirCompaniaReport(_idCompania, _adrCompania);

}

function anadirSolicitudACompania(uint _personal_id, uint
_codigoSolicitud, uint _precio) {

    compSolTrayAdr[_codigoSolicitud].personal_id = _personal_id;
    //compSolTrayAdr[_codigoSolicitud].adrSolicitud = _adrSolicitud;

    compSolTrayAdr[_codigoSolicitud].precio = _precio;

    indexPrecioDevuelto = numPreciosDevueltos.length++;
    numPreciosDevueltos[indexPrecioDevuelto]=_codigoSolicitud;

}

function anadirEleccionACompania(uint _codigoSolicitud) {

    compSolTrayAdr[_codigoSolicitud].confirmado = "si";

    indexContratoActivado = numContratosActivados.length++;
    numContratosActivados[indexContratoActivado]=_codigoSolicitud;

}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
function calcularYDevolverPrecio(uint _personal_id, uint _edad, uint
_numanosconcarner, uint _distancia, uint _tipoCobertura, uint
_codigoSolicitud) returns (uint256 RPrecioADevolver, uint RidCompania, string
RnombreCompania){

    uint precioADevolver = 0;

    //LA DISTANCIA VIENE EN METROS

    precioADevolver=(pond_distancia*_distancia)*(pond_tipoCobertura*_tipoCobertura
)*10 + (pond_edad*_edad)*100000/(pond_numanosconcarner*_numanosconcarner);

    anadirSolicitudACompania(_personal_id,_codigoSolicitud,
precioADevolver);

    //EVENTUALMENTE AQUI HABRÁ QUE INTRODUCIR DATOS EN EL REPORT,
AÑADIENDO A VECTORES PARA EL "HISTORICO DE PRECIOS OFRECIDOS", DE CARA, POR
EJEMPLO A LAS REDES NEURONALES PARA EL CÁLCULO ÓPTIMO DE PRECIOS

    return (precioADevolver, companiaDatos[address(this)].idCompania,
companiaDatos[address(this)].nombreCompania);

}

//FUNCION PARA EVENTUALMENTE CAMBIAR LAS PONDERACIONES DESEADAS POR CADA
COMPañIA (ponemos modifier onlyCompania)

function setPonderaciones(uint _pond_edad, uint _pond_numanosconcarner,
uint _pond_distancia, uint _pond_tipoCobertura)

    onlyCompania{

        pond_edad= _pond_edad;
        pond_numanosconcarner= _pond_numanosconcarner;
        pond_distancia= _pond_distancia;
        pond_tipoCobertura= _pond_tipoCobertura;
        string nombre = companiaDatos[address(this)].nombreCompania ;

        statusComp(msg.sender, "Cambio de ponderaciones", block.timestamp);
    }

////

function getPreciosDevueltosAtIndex(uint indice) returns (uint){

    return numPreciosDevueltos[indice];
}

function getContratosActivadosAtIndex(uint indice) returns (uint){

    return numContratosActivados[indice];
}

}
```



**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
////////////////////////////////////

//Si queremos comunicarnos con el smartcontract cliente, debemos incluir en el
mismo documento.sol el smartcontract pero únicamente el constructor y
//la definición de las funciones a las que vamos a hacer referencia, NO es
necesario incluir todo el código de lógica

//Comunicación con el contrato de Report

contract Report{

    uint256 indexCompania;
    uint256 indexClienteSolicitudTrayecto;

    struct Comp {

        address adrCompania;
        uint idCompania;

    }

    struct ClienteSolTray{

        uint personal_id;
        uint idCompania;

        address adr;
        address adrCompania;

        uint256 precio;
        string confirmado;

        uint distancia;
        string tipoCobertura;

        uint edad;
        uint numanoscon carnet;

        uint256 creationDateSolicitud;
        uint256 creationDateEleccion;

    }

    mapping (uint => ClienteSolTray) public clientSolTrayAdr;
    mapping (uint => Comp) public compAdr;

    address[] numCompanias;
    address[] numClientesSolicitudTrayecto;

    function anadirCompaniaReport(uint _idCompania, address _adrCompania){

        compAdr[_idCompania].idCompania = _idCompania;
        compAdr[_idCompania].adrCompania = _adrCompania;
    }
}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
indexCompania = numCompanias.length++;  
numCompanias[indexCompania]=_adrCompania;  
  
}  
  
}
```

### 3 Código de *ClienteSolicitudTrayecto.sol*

```
pragma solidity ^0.4.0;

//SmartContract que se crea cada vez que un solicitante se da de alta en el
//sistema, este smartcontract será útil para tener registrados dentro de
//blockchain a cada solicitante, teniendo un smartcontract para cada
//solicitante y cada uno con su dirección hash. Estos smartcontracts estarán
//comunicados con el smartcontract "maestro" (ReportCliente.sol) donde se
//almacenará en un mapping el dni del solicitante y con su dirección
//hash (de smartcontract) correspondiente.

//Contrato de clientesolicitudtrayecto

contract ClienteSolicitudTrayecto{

    //variables propias del contrato
    address public adr;

    address adrClienteUsuario = address(this);

    //Estructura que contiene datos del usuario en el momento del registro
    (identificativos + interesantes estáticos)

    struct ClienteSolicitud {

        address adrSolicitud;
        uint codigoSolicitud;

        uint distancia;
        string tipoCobertura;

        uint personal_id;
        uint edad;
        uint numanosconcarne;

        uint256 creationDateSolicitud;

        uint idCompania;

        uint256 precio;

    }

    //Mapeo de las estructuras con la direccion del
    contrato(clientesolicitudtrayecto)

    mapping (address => ClienteSolicitud) public clientDataSolicitud;

    //////////////////////////////////////

    ////////////////////////////////////// EVENTOS
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
    event pago(address adr, uint id, string signo, uint amount, uint256 time,
uint balance);

////////////////////////////////////

////////////////////////////////////CONSTRUCTOR

    //Esta funcion se ejecuta al crear el contrato, añadiendo toda la
informacion de la SOLICITUD

    function ClienteSolicitudTrayecto(uint _personal_id, uint _edad, uint
_numanoscon carnet, uint _distancia, string _tipoCobertura, uint
_codigoSolicitud) {

        //PARTE OBTENIDA POR PANTALLA (LLAMO A FUNCIONES QUE SOLO AÑADE):
añado los datos recibidos tanto al SC como al Report.sol

        // adrClient= address(this);

        //COMENTO:
        addClienteSolicitud(_personal_id, _edad, _numanoscon carnet,
_distancia, _tipoCobertura, _codigoSolicitud);
        addClienteSolicitudReport(_codigoSolicitud, address(this),
_personal_id);

    }

    //Esta función nos servira para grabar en blockchain el evento de cuando
un ayuntamiento recibe el pago de una ONG, ya que luego a la hora de
//buscar los eventos por el blockchain, debemos buscar eventos
referenciados a una dirección de smartcontract, si no grabasemos esto en este
//SmartContract no tendríamos traza de los pagos que rebice un
smartcontract Ayuntamiento.

    function registrarPago(address adr, uint id, uint amount){

        pago(adr, id, "+", amount, block.timestamp, this.balance);
    }

    //el sc va a recibir ether mediante esta funcion,
//la cual al mismo tiempo va a pagar lo que recibe a la Compania

    function pagarCompany()
    payable // payable so it accepts value: from sender
    returns(bool success)
    {
        if(msg.value != clientDataSolicitud[address(this)].precio) throw;

        address adrCompany =report.getAddressCompania(
            clientDataSolicitud[address(this)].idCompania
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
);

    registrarPago(
        adrCompany,
        clientDataSolicitud[address(this)].idCompania,
        msg.value);
    adrCompany.transfer(msg.value); // sends the funds

    return true;
}

//Esta función nos devuelve el balance del smartcontract,
//es decir la cantidad de ether que contiene
function balances() public view returns (uint) {
    return this.balance;
}

////////////////////////////////////

//INICIO DE GESTIONES PARA EMPEZAR A GOBERNAR EL REPORTCLIENT.SOL

//Si tenemos el contrato reporting en otro documento, se hace de la
siguiente manera:

    Report report = Report(0x412bedd745f41cece781096836fec53a4908ffc7); // es
la buena 17/08

//FINAL DE GESTIONES PARA EMPEZAR A GOBERNAR EL REPORTE.SOL

//añadir al report

    function addClienteSolicitudReport(uint _codigoSolicitud, address
_adrSolicitud, uint _personal_id) private{

        // VA EN EL CONSTRUCTOR, NO HA SIDO CREADO TODAVIA, HAY QUE PASARLE LA
ADDRESS

        report.anadirClienteSolicitudReport(_codigoSolicitud, _adrSolicitud,
_personal_id);

    }

//añadir al Compania.sol que corresponda

//función para colocar la ELECCION del solicitante EN SU COMPANIA.SOL
(TODO DEBE RECIBIRSE MEDIANTE MAPEO AQUÍ!!!!)

    function addEleccionACompania(uint256 _codigoSolicitud, address
_adrCompania) returns (bool ok){

        bool h = false;
        //nos comunicamos con "nuestra compania", que previamente ya ha
asociado una address de ClienteSolicitudTrayecto al codigo
        //(esto se ha producido en el momento en que esta compania devuelve un
precio asociado a esta solicitud de trayecto de ESTE cliente)
        Compania compania = Compania(_adrCompania);
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
    compania.anadirEleccionACompania(_codigoSolicitud);

    h=true;

    return h;
}

//añadir al Cliente.sol que corresponda

//función para colocar la ELECCION de compania EN SU CLIENTEUSUARIO.SOL
(TODO DEBE RECIBIRSE MEDIANTE MAPEO AQUÍ!!!!)

function addEleccionAClienteUsuario(uint _codigoSolicitud, address
_adrSolicitud, uint256 _precio, uint _idCompania, address _adr) returns (bool
ok){

    bool h = false;
    //nos comunicamos con nuestro "contrato de usuario"
    ClienteUsuario clienteusuario = ClienteUsuario(_adr);

    clienteusuario.anadirEleccionAClienteUsuario(_codigoSolicitud,
_adrSolicitud, _precio, _idCompania);

    h=true;

    return h;
}

//función para colocar la ELECCION del solicitante EN EL REPORT.SOL (TODO
DEBE RECIBIRSE MEDIANTE MAPEO AQUÍ!!!!)

//está usando una funcion DENTRO que EXISTE en Report.sol

//OBVIAMENTE PARA UTILIZAR setEleccion hace falta tener la address del
contrato ClienteSolicitudTrayecto (para instanciar la transaccion)
//para tener eso hay que tener el codigoSolicitud
//el codigo solicitud solo lo tienen en pantalla los dueños del contrato

function setEleccion(uint _idCompania, uint256 _precio, uint _personal_id)
returns (uint idCompania, uint precio) {

    clientDataSolicitud[address(this)].idCompania=_idCompania;
    clientDataSolicitud[address(this)].precio = _precio;

    return (clientDataSolicitud[address(this)].idCompania,
    clientDataSolicitud[address(this)].precio);
}

////////////////////////////////////
function getPrecio () returns (uint256 _precio){

    return(clientDataSolicitud[address(this)].precio);
}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
    }

    //funcion que añade los datos de la SOLICITUD

    function addClienteSolicitud(uint _personal_id, uint _edad, uint
    _numanosconcarner, uint _distancia, string _tipoCobertura, uint
    _codigoSolicitud) private{

        //ESTA ES LA MANERA DE RELLENAR UNA ESTRUCTURA (DE MAPEO) ENTERA, SIN
        IR COLUMNA A COLUMNA

        clientDataSolicitud[address(this)] = ClienteSolicitud({

            adrSolicitud: address(this),

            codigoSolicitud: _codigoSolicitud,

            distancia: _distancia,
            tipoCobertura: _tipoCobertura,

            personal_id: _personal_id,
            edad: _edad,
            numanosconcarner: _numanosconcarner,

            creationDateSolicitud: block.timestamp,

            precio:0,

            idCompania:0

        });
    }

    //funcion que devuelve la informacion de un cliente, se le pasa como
    parametro la direccion del cliente
    //desde node js la direccion del cliente nos la dará el contrato de
    reporting cuando se lo solicitemos
    //introduciendo el personal_id del cliente. A esta función solo la puede
    llamar el cliente y la compañía.

    function getDataSolicitud(address _adrSolicitud) returns (uint a, uint b,
    string c, uint d, uint e, uint256 f) {

        return (clientDataSolicitud[_adrSolicitud].personal_id,
        clientDataSolicitud[_adrSolicitud].distancia,
        clientDataSolicitud[_adrSolicitud].tipoCobertura,
        clientDataSolicitud[_adrSolicitud].edad,
        clientDataSolicitud[_adrSolicitud].numanosconcarner,
        clientDataSolicitud[_adrSolicitud].creationDateSolicitud
        );
    }

    function getDataEleccion(address _adrSolicitud) returns (uint a, uint256
    b) {
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
        return (clientDataSolicitud[_adrSolicitud].idCompania,
                clientDataSolicitud[_adrSolicitud].precio
                );
    }

//AQUI SE INTRODUCIRIAN EVENTUALMENTE LOS MODIFIERS

}

//INICIO DE TODO LO QUE HAY QUE AÑADIR PARA QUE FUNCIONE LA COMUNICACION CON
REPORTCLIENT.SOL (en general, SIN entrar en INSTANCIAS)

//Si queremos comunicarnos con el smartcontract cliente, debemos incluir en el
mismo documento.sol el smartcontract pero únicamente el constructor y
//la definición de las funciones a las que vamos a hacer referencia, NO es
necesario incluir todo el código de lógica

//Comunicacion con el contrato Report

contract Report{

    uint256 indexClienteSolicitudTrayecto;

    struct ClienteSolTray{

        uint codigoSolicitud;
        address adrSolicitud;

        /////

        uint personal_id;
        uint idCompania;

        uint256 precio;
        string confirmado;

        uint distancia;
        string tipoCobertura;

        uint edad;
        uint numanoscon carnet;

    }

    struct Comp {

        address adrCompania;
        uint idCompania;

    }

    struct ClientsAddress{
```



**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
    uint personal_id;

    address adr;
}

mapping (uint => ClienteSolTray) public clientSolTrayAdr;
mapping (uint => Comp) public compAdr;
mapping (uint => ClientsAddress) public clienteAdr;

address[] numClientesSolicitudTrayecto;

function anadirClienteSolicitudReport(uint _codigoSolicitud, address
_adrSolicitud, uint _personal_id){

    clientSolTrayAdr[_codigoSolicitud].adrSolicitud = _adrSolicitud;
    clientSolTrayAdr[_codigoSolicitud].codigoSolicitud = _codigoSolicitud;
    clientSolTrayAdr[_codigoSolicitud].personal_id = _personal_id;

    indexClienteSolicitudTrayecto = numClientesSolicitudTrayecto.length++;
    numClientesSolicitudTrayecto[indexClienteSolicitudTrayecto]=
_adrSolicitud;
}

function anadirClienteSolicitudEleccionReport(uint _codigoSolicitud,
uint256 _precio, uint _idCompania) {

    clientSolTrayAdr[_codigoSolicitud].idCompania = _idCompania;
    clientSolTrayAdr[_codigoSolicitud].precio = _precio;

}

//////////

////////////////////////////////////

////////// PARA SACAR LAS ADDRESSES DE LA COMPAÑIA Y DEL CLIENTEUSUARIO A LA
HORA DE COLOCAR LA ELECCION
// DEFINITIVA EN LOS 3 CONTRATOS (comp, client, report)

function getAddressCompania(uint _idCompania) returns (address) {

    return(

        compAdr[_idCompania].adrCompania
    );
}

function getAddress(uint _personal_id) returns (address) {

    return(

        clienteAdr[_personal_id].adr
    );
}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
}  
  
}  
  
contract Compania{  
  
    struct CompaniaData{  
  
        string nombreCompania;  
        uint idCompania;  
        address adrCompania;  
        string pass;  
  
        uint numPreciosDevueltos;  
        uint numContratosActivados;  
  
    }  
  
    struct CompaniaSolicitudTrayecto{  
  
        uint codigoSolicitud;  
  
        /////  
  
        address adrSolicitud;  
  
        uint personal_id;  
        uint edad;  
        uint numanoscon carnet;  
  
        uint256 creationDateSolicitud;  
  
        uint256 precio;  
        uint256 creationDateEleccion;  
        string confirmado;  
  
    }  
  
    uint256 indexPrecioDevuelto;  
    uint256 indexContratoActivado;  
  
    mapping (uint => CompaniaSolicitudTrayecto) public compSolTrayAdr;  
  
    uint[] numPreciosDevueltos;  
    uint[] numContratosActivados;  
  
    function anadirSolicitudACompania(uint _personal_id, address  
_adrSolicitud, uint _codigoSolicitud, uint _precio){  
  
        compSolTrayAdr[_codigoSolicitud].personal_id = _personal_id;  
        compSolTrayAdr[_codigoSolicitud].adrSolicitud = _adrSolicitud;  
  
        compSolTrayAdr[_codigoSolicitud].precio = _precio;  
  
        indexPrecioDevuelto = numPreciosDevueltos.length++;  
  
    }  
  
}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
    numPreciosDevueltos[indexPrecioDevuelto]=_codigoSolicitud;

}

function anadirEleccionACompania(uint _codigoSolicitud){

    compSolTrayAdr[_codigoSolicitud].confirmado = "si";

    indexContratoActivado = numContratosActivados.length++;

//numContratosActivados[indexContratoActivado]=compSolTrayAdr[_codigoSolicitud
].adrSolicitud;
    numContratosActivados[indexContratoActivado]=_codigoSolicitud;

}

}

//FINAL DE TODO LO QUE HAY QUE AÑADIR PARA QUE FUNCIONE LA COMUNICACION CON
REPORTCLIENT.SOL (en general, SIN entrar en INSTANCIAS)

//Contrato de cliente
contract ClienteUsuario{

//////////VARIABLES Y ESTRUCTURAS

    //Estructura que contiene datos del usuario en el momento del registro
(identificativos + interesantes estáticos)

    struct ClientPersonal {

        address adr;
        string name;
        string surname;
        uint personal_id;
        uint edad;
        uint numanoscon carnet;

        string street;
        string country;

        string password;

        uint256 registroDate;

    }

    struct ClienteUsuarioSolicitudTrayecto {

        uint codigoSolicitud;

        /////
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
address adrSolicitud;

///

uint idCompania;

///

uint256 creationDateSolicitud;

uint256 precio;
uint256 creationDateEleccion;
string confirmado;

}

uint256 indextrayectoQueHeContratado;

uint[] public trayectosQueHeContratado;

//Mapeo de las estructuras con la direccion del contrato(cliente)

mapping (address => ClientPersonal) public clientDataPersonal;
mapping (uint => ClienteUsuarioSolicitudTrayecto) public
clienteusuarioSolTrayAdr;

//Este evento nos permitirá grabar dirección, estado y fecha cada vez que
le invoquemos
//para después poder recuperar un log de eventos desde node js

event status(address adr, string estado, uint256 time); ///PREGUNTAR A
IVAN: ENTIENDO QUE LA GRACIA ES SOLO REGISTRAR LA CREACIÓN DE UN NUEVO
SOLICITANTE

////////////////////////////////////

function anadirEleccionAClienteUsuario(uint _codigoSolicitud, address
_adrSolicitud, uint256 _precio, uint _idCompania){

    clienteusuarioSolTrayAdr[_codigoSolicitud].codigoSolicitud =
_codigoSolicitud;
    clienteusuarioSolTrayAdr[_codigoSolicitud].adrSolicitud =
_adrSolicitud;
    clienteusuarioSolTrayAdr[_codigoSolicitud].precio = _precio;
    clienteusuarioSolTrayAdr[_codigoSolicitud].idCompania = _idCompania;

    indextrayectoQueHeContratado = trayectosQueHeContratado.length++;
trayectosQueHeContratado[indextrayectoQueHeContratado]=_codigoSolicitud;

}
}
```

## 4 Código de *Report.sol*.

```
pragma solidity ^0.4.0;

contract Report{

    uint256 indexClienteUsuario;
    uint256 indexClienteSolicitudTrayecto;
    uint256 indexCompania;

    struct ClientsAddress{

        uint personal_id;

        address adr;

    }

    struct ClienteSolTray{

        uint codigoSolicitud;
        address adrSolicitud;

        uint personal_id;
        uint idCompania;

        uint256 precio;

        uint distancia;
        string tipoCobertura;

        uint edad;
        uint numanoscon carnet;

    }

    struct Comp {

        address adrCompania;
        uint idCompania;

    }

    ///// LISTO

    mapping (uint => ClientsAddress) public clienteAdr;
    mapping (uint => ClienteSolTray) public clientSolTrayAdr;
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
mapping (uint => Comp) public compAdr;

address[] numClientes;
address[] numClientesSolicitudTrayecto;
address[] numCompanias;

///// LISTO

function anadirClienteReport(uint _personal_id, address _adr){

    clienteAdr[_personal_id].adr = _adr;
    clienteAdr[_personal_id].personal_id = _personal_id;

    indexClienteUsuario = numClientes.length++;
    numClientes[indexClienteUsuario]=_adr;
}

function anadirCompaniaReport(uint _idCompania, address _adrCompania){

    compAdr[_idCompania].adrCompania = _adrCompania;
    compAdr[_idCompania].idCompania = _idCompania;

    indexCompania = numCompanias.length++;
    numCompanias[indexCompania]=_adrCompania;
}

function anadirClienteSolicitudReport(uint _codigoSolicitud, address
_adrSolicitud, uint _personal_id){

    clientSolTrayAdr[_codigoSolicitud].adrSolicitud = _adrSolicitud;
    clientSolTrayAdr[_codigoSolicitud].codigoSolicitud = _codigoSolicitud;
    clientSolTrayAdr[_codigoSolicitud].personal_id = _personal_id;

    indexClienteSolicitudTrayecto = numClientesSolicitudTrayecto.length++;
    numClientesSolicitudTrayecto[indexClienteSolicitudTrayecto]=
_adrSolicitud;
}

function anadirClienteSolicitudEleccionReport(uint _codigoSolicitud,
uint256 _precio, uint _idCompania) {

    clientSolTrayAdr[_codigoSolicitud].idCompania = _idCompania;
    clientSolTrayAdr[_codigoSolicitud].precio = _precio;

}

////////////////////////////////////

//TROZO CON FUNCIONES FACILES DE GET(keyvalue del mapping), cogiendo
//datos de las estructuras de este .sol --> tienen RETURNS

function getAddress(uint _personal_id) public returns (address) {
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
        return(clienteAdr[_personal_id].adr);
    }

    function getAddressCompania(uint _idCompania) returns (address) {

        return(

            compAdr[_idCompania].adrCompania

        );
    }

    function getAddressClienteSolicitudTrayect(uint _codigoSolicitud) returns
(address) {

        return(

            clientSolTrayAdr[_codigoSolicitud].adrSolicitud

        );
    }

//////// LISTO

    function getClieteAtIndex(uint posicion_array) returns (address){

        return numClientes[posicion_array];
    }

    function getCompaniaAtIndex(uint indice) public returns (address) {

        return numCompanias[indice];
    }

    function getClietesSolicitudTrayectoAtIndex(uint indice) returns
(address){

        return numClientesSolicitudTrayecto[indice];
    }

    function getNumClientes() returns (uint){

        return numClientes.length;
    }

    function getNumCompanias() returns (uint){

        return numCompanias.length;
    }

    function getNumClientesSolicitudTrayecto() returns (uint){

        return numClientesSolicitudTrayecto.length;
    }
}
```

**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**

```
}
```

```
//// LISTO
```

```
}
```



**UNIVERSIDAD PONTIFICIA DE COMILLAS**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)**  
**GRADO EN INGENIERÍA ELECTROMECÁNICA**