



Componente formativo

Construcción y despliegue de contratos inteligentes

Breve descripción:

Mediante el presente componente, se brindarán los conceptos y recomendaciones necesarias para la creación de un contrato inteligente sobre una red de “blockchain”, haciendo uso de herramientas y lenguajes de programación especializados.

Área ocupacional:

Tecnologías de la información.

Abril 2023

Tabla de contenido

Introducción.....	3
1. Construcción de contratos inteligentes	4
1.1. Definición, características	4
1.2. Limitaciones de contratos inteligentes.....	8
1.3. Estructura de contrato.....	9
2. Plataforma Ethereum.....	12
2.1. EVM Ethereum Virtual Machine	13
2.2. Instalación y Configuración	15
3. Principios básicos de Solidity.....	23
3.1. Expresiones	24
3.2. Estructuras de control	35
3.3. Estructuras repetitivas.....	36
3.4. Funciones	39
4. Desplegar Contratos Inteligentes.....	41
4.1. Entorno de prueba y desarrollo	42
4.2. Entorno de ejecución	47
Síntesis	48
Material complementario	49
Glosario.....	50
Referencias bibliográficas.....	51
Créditos.....	52

Introducción

Los contratos inteligentes ofrecen funcionalidades de intercambio de información a través de redes de “blockchain”, estos contratos permiten extender las funcionalidades tanto de flexibilidad y seguridad de la información dado que se ejecutan bajo protocolos seguros propios de estas redes de “blockchain”, heredando las bondades de confiabilidad, disponibilidad y autenticidad de la misma. Se debe observar la introducción al componente formativo en el siguiente video:

Video 1. Construcción y despliegue de contratos inteligentes



[Construcción y despliegue de contratos inteligentes](#)

Síntesis del video: Construcción y despliegue de contratos inteligentes.

Cuando hablamos de contratos, imaginamos documentos extensos, cargados de letra pequeña, firmas en cada una de sus hojas, y usualmente bajo la intermediación de un abogado, o alguien que haga las partes de registro y normativa.

Sin embargo, un contrato es inteligente, debido a que tiene la capacidad de ejecutarse de manera automática una vez que se ha configurado basándose en los

acuerdos logrados entre las partes son guardianes del cumplimiento de los acuerdos efectuados, y se ejecutan de manera irreversible e incuestionable.

En este componente formativo, podrá conocer sobre la plataforma Ethereum, su instalación y configuración, lo que a su vez permitirá posteriormente desplegar los contratos inteligentes, reconociendo el entorno de prueba y desarrollo, así como el de ejecución. Sin olvidar los principios básicos de Solidity, las expresiones, y estructuras, de control y repetitivas, además de sus funciones. Todo esto, para entender y aplicar de forma global el tema de contratos inteligentes.

1. Construcción de contratos inteligentes

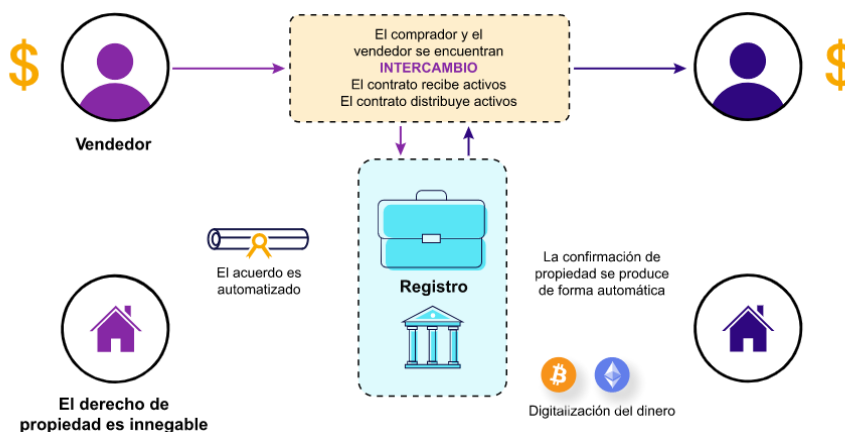
El proceso de construcción de un contrato inteligente requiere tener presente algunos conceptos importantes sobre su funcionamiento, herramientas tecnológicas, conceptos propios de las redes de “blockchain” y ciertos conocimientos básicos de programación preferiblemente de lenguajes de programación especializados como “Solidity”, se invita a continuar con el reconocimiento de estos conocimientos y de la estructura de un contrato inteligente.

1.1. Definición, características

Un contrato inteligente o “Smart Contract”, como su nombre lo indica, es un contrato semejante al contrato ordinario, en donde se requiere participar presencialmente, firmar y/o sellar, pero, ahora se cuenta con un mecanismo y agilidad que permite establecer acciones para una transacción, sin estar presente, bajo algunos controles y un bajo costo de transacción; para la creación de un contrato inteligente se hace uso de lenguajes de programación sobre los cuales se programan las reglas, condiciones y flujo de información para finalmente registrar dicha información sobre una red de “blockchain”.

El término “Smart Contract” es asociado a contratos automatizados, lo cual es correcto. Pero, vale la pena aclarar que, si bien los orígenes del término y las intenciones de quien lo acuñó por primera vez quizás se centraron en permitir contratos automatizados, la implementación final sobre una red de “blockchain” los ubica más próximos a programas de computación de uso más o menos genérico, con la posibilidad de acceder a dinero (“token's”) y con algunas restricciones y costos de ejecución.

Figura 1. Cómo funcionan los contratos inteligentes



La inclusión de estos contratos inteligentes, en diferentes aplicaciones y soluciones de comercio electrónico, basadas en “blockchain”, facilitan actividades como:

- Gobierno y administraciones públicas.
- Mercados inmobiliarios.
- Transporte.
- Automóvil.
- Salud.
- Auditoría.
- Compañías de seguros.

- h) Banca.
- i) Notariado y Registro.

Al comparar los contratos tradicionales y los contratos inteligentes, existe una marcada diferencia, lo cual es un punto de apoyo para tornar a esta tecnología, sin embargo, vale la pena recordar en que se soporta esta diferencia y las principales ventajas que tiene:

Tabla 1. Contratos tradicionales vs. contratos inteligentes

Característica	Contratos inteligentes	Contratos tradicionales
Tiempo	Se procesan en cuestión de minutos.	Puede demorar entre 1 y 3 días el proceso.
Asignación	Las asignaciones se realizan de manera automática.	Las asignaciones se realizan de forma manual, lo cual puede llevar a equivocaciones.
Fideicomiso	Incorporar “blockchain” para validar que se cumpla la voluntad del fideicomitente y la autoridad competente, no es absolutamente necesario.	La seguridad pactada para proteger el cumplimiento de acciones establecidas en el contrato y que se denomina fideicomiso es necesario.
Costos	Se realiza por una fracción del costo de un contrato tradicional.	Los costos varían de acuerdo con el tipo de contrato, sin embargo, es un rubro a considerar siempre que se pacte un contrato tradicional.
Firmas	Las firmas se realizan de manera digital, por lo que el pacto se puede hacer entre personas que no estén presentes físicamente en un mismo lugar.	Es requisito que las personas interesadas, estén presentes para llevar a cabo el registro

Característica	Contratos inteligentes	Contratos tradicionales
		de firmas ante notario público.
Soporte legal	No requiere de acompañamiento legal, puede tramitarse directamente por personas naturales.	Requiere el acompañamiento de un abogado, bien sea al redactar el contrato, o al llevarlo a registro.

Elementos de los contratos inteligentes

La principal variedad en estos elementos radica en la forma en que se aplican, dado que, en el contrato inteligente, el consentimiento de los interesados se manifiesta de forma electrónica.

Adicional, a estos tres elementos básicos, los contratos inteligentes precisan de ciertos medios especiales para poder ejecutarse y también necesitan de la elaboración de definiciones concretas de los parámetros del contrato, ya que no se puede dar lugar al subjetivismo.

Los principales elementos o medios informáticos adicionales para el funcionamiento y ejecución de este tipo de contratos inteligentes son:

- 1) **Plataforma de acceso:** debe contar con una plataforma con acceso a tecnología de contabilidad distribuida y formada por varios nodos de red “blockchain”.
- 2) **Nodos “blockchain”:** estos nodos son ordenadores que custodian los registros, es decir “los libros contables”.
- 3) **Lenguaje computacional:** es muy importante contar con un lenguaje computacional encriptado que sea adecuado.
- 4) **Funciones hashes:** los mineros que son las propias personas que trabajan programando y sus hardware para autorizar la adición de los bloques de

transacciones a la cadena de bloques. Realizando funciones hashes correspondientes al protocolo de consenso entre las partes.

1.2. Limitaciones de contratos inteligentes

Como se observó anteriormente, el campo de acción donde se pueden implementar contratos inteligentes es extenso, y se pueden abordar infinidad de proyectos para el intercambio de información de manera segura a través de redes de blockchain.

Ahora bien, dada su aplicabilidad y su robustez generada por sus características, también se puede ver afectado por ciertas limitaciones o restricciones para su adopción, se debe revisar algunos casos en donde se puede presentar cierta limitación:

- 1) **Flexibilidad:** la ejecución basada en la lógica a menudo es un desafío para los bufetes de abogados. La creación de contratos tiene un elemento subjetivo y tiene un uso intencional de términos como "buena fe" o "razonable" o "mejores esfuerzos para tener espacio para la flexibilidad". Este tipo de contrato es necesario cuando las partes quieren crear un contrato relacional en lugar de uno transaccional. En tales escenarios, los contratos inteligentes no ofrecen utilidad.
- 2) **Ramificaciones legales:** dado que los contratos inteligentes reducen la dependencia de intermediarios y abogados, es imperativo que las partes involucradas sean conscientes de las ramificaciones legales del derecho público, privado, penal y mercantil.
- 3) **Transacciones retrasadas:** hay casos en los que la tecnología demuestra ser una pesadilla en lugar de una bendición. Uno de estos casos es que la congestión en una red "blockchain" puede resultar en una transacción retrasada y, finalmente, impulsar el costo transaccional por encima del costo incurrido en los contratos tradicionales.
- 4) **Dificultad para actualizar información:** cada contrato inteligente al desplegarse sobre una red de "blockchain" se propaga por todos sus nodos, y cualquier cambio en su versión desplegada es casi imposible, conllevando grandes dificultades actualizar o corregir alguna falla.

- 5) **Posibilidad de generar vacíos o lagunas:** de acuerdo con el concepto de buena fe, las partes tratarán de manera justa y no obtendrán beneficios poco éticos de un contrato. Sin embargo, el uso de contratos inteligentes hace que sea difícil garantizar que los términos se cumplan de acuerdo con lo acordado.
- 6) **Terceros:** a pesar de que los contratos inteligentes buscan reducir la participación de terceros, no es posible eliminarlos. Los terceros asumen roles diferentes de los que asumen en los contratos tradicionales. Por ejemplo, no se necesitarán abogados para preparar contratos individuales; sin embargo, los desarrolladores los necesitarán para comprender los términos para crear códigos para contratos inteligentes.
- 7) **Términos vagos:** dado que los contratos incluyen términos que no siempre se entienden, los contratos inteligentes no siempre pueden manejar términos y condiciones que son vagos.

1.3. Estructura de contrato

A diferencia de los contratos tradicionales, los contratos inteligentes son programas codificados en un lenguaje de programación especializado que se despliega sobre una red de blockchain, por lo cual su codificación obedece a una estructura sugerida de un documento electrónico.

No existe una estructura general definida por algún estándar, aunque sí hay algunos apartados importantes y obligatorios; a continuación, se puede observar de forma general los apartados correspondientes de un contrato inteligente básico:

```
1 //COMPILADOR
2 pragma solidity ^0.6.5;
3
4 //IMPORTAR OTROS CONTRATOS, INTERFACES O LIBRERIAS
5
6 //CUERPO DEL CONTRATO
7 contract PrimerContrato{
```

```
8
9    //LIBRERIAS
10
11    //DECLARACION DE VARIABLES
12    uint private myValue;
13
14    //EVENTOS
15    event LogSetMyvalue (uint _myNewvalue, address _sender);
16
17    //MODIFICADORES
18
19    //CONSTRUCTOR
20    constructor(uint _myvalue) public {
21        myvalue = _myvalue;
22    }
23
24    //FUNCIONES
25    //Lee el valor de la variable myValue
26    function getMyvalue()
27        public
28        view
29        returns (uint)
30    {
31        return myValue;
32    }
33
34    //Modifica el valor de la variable _myNewvalue
35    function setMyValue( uint _myNewvalue) public {
36        myValue = _myNewValue;
37        emit LogsetmyValue(myValue, msg.sender);
38    }
39 }
```

El contrato inteligente básico, cuenta con apartados que le dan consistencias, estos son:

- 1) Información de compilador: declara la versión mínima del compilador del contrato que se debe utilizar, para este caso será la versión 0.6.5 o superior, en caso de utilizar una versión menor, podría causar problemas por compatibilidad.
- 2) Contratos, interfaces o librerías a importar: en este apartado se deben de especificar todos aquellos archivos de otros contratos, interfaces o librerías que son necesarias para que el contrato inteligente pueda funcionar, se debe agregar las líneas con la función import seguido de la ruta relativa del archivo a importar.
- 3) Cuerpo del contrato: en este espacio podremos programar la lógica de nuestro contrato inteligente, teniendo en cuenta que podremos incluir: librerías variables, eventos, modificadores, constructores y funciones personalizadas.
- 4) Otros apartados a tener en cuenta:
 - a) Funciones externas: funciones almacenadas en archivos externos los cuales pueden extender las capacidades del contrato inteligente, o también podemos citar otros contratos.
 - b) Variables: las variables de estado son variables cuyos valores se almacenan permanentemente en el almacenamiento contractual.
 - c) Eventos: los eventos son interfaces de conveniencia con las instalaciones de registro de EVM.
 - d) Modificadores de funciones: los modificadores de funciones se pueden utilizar para modificar la semántica de las funciones de forma declarativa, la sobrecarga, es decir, tener el mismo nombre modificador con diferentes parámetros, no es posible.

Al igual que las funciones, los modificadores se pueden anular, a continuación, lo correspondiente al constructor:

Constructor:

1. Función opcional, declarada con la palabra clave que se ejecuta en el momento de creación del contrato, y en donde se puede ejecutar el código de inicialización del contrato.

2. Antes de ejecutar el código del constructor, las variables de estado se inicializan a su valor especificado, si se inicializan en línea, o a su valor predeterminado si no se hace.
3. Una vez ejecutado el constructor, el código final del contrato se implementa en la cadena de bloques.
4. Este código incluye todas las funciones que forman parte de la interfaz pública y todas las funciones a las que se puede acceder desde allí a través de llamadas a funciones. No incluye el código del constructor ni las funciones internas a las que solo se llama desde el constructor.
5. Si no hay ningún constructor, el contrato asumirá el constructor predeterminado.

Otro modificador, son las Funciones, estas son las unidades ejecutables de código. Se definen dentro de un contrato, pero también se pueden definir fuera de los contratos.

Las llamadas a funciones pueden ocurrir interna o externamente y tienen diferentes niveles de visibilidad hacia otros contratos. Las funciones aceptan parámetros y devuelven variables para pasar parámetros y valores entre ellas.

Se puede encontrar más información sobre la estructura y sintaxis de un contrato inteligente construido con Solidity, en la página oficial, al que se puede acceder, [ver aquí](#).

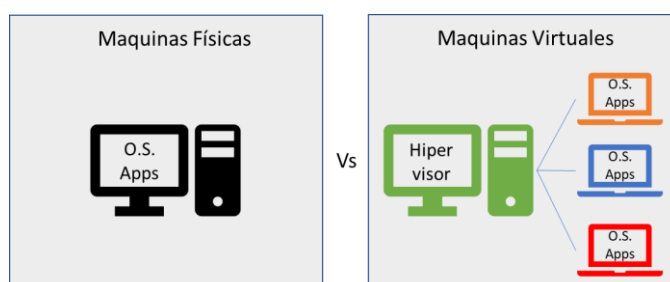
2. Plataforma Ethereum

La arquitectura de “blockchain” de Ethereum es una de las más grandes y utilizadas en proyectos de aplicaciones y contratos inteligentes sobre “blockchain”, esto se debe en un principio a que es una de las primeras redes, su nivel de innovación y que ha dispuesto de las herramientas necesarias para que la comunidad de desarrolladores y usuarios puedan hacer uso de la misma de manera fácil, su propuesta de Ethereum Virtual Machine (EVM) brinda una máquina virtual con la cual puede dar inicio al desarrollo y construcción de proyectos bajo esta plataforma.

2.1. EVM Ethereum Virtual Machine

Dado su nombre de Virtual Machine, es importante comprender inicialmente ¿qué es una máquina virtual?, consiste en un elemento compuesto por aplicaciones de software que realiza la emulación de un dispositivo real, aprovechando los recursos hardware de un dispositivo para servir diferentes sistemas o aplicaciones.

Figura 2. Máquinas Físicas vs Máquinas Virtuales



Una máquina virtual tiene las capacidades de emular todas las funciones de un computador normal y su capacidad es de acuerdo con los recursos de la maquina principal y de la distribución de recursos que le sean asignados, como son: ajustes de procesamiento, memoria RAM, almacenamiento, tráfico, entre otros más. Gracias a esto, puede ejecutar aplicaciones específicas sin dificultar el desarrollo normal de las demás máquinas virtuales.

“Ethereum Virtual Machine” (EVM), es una máquina con “software” diseñado para emular un mecanismo con funciones y capacidades que posibilitan el desarrollo de actividades relacionadas con la red de “blockchain” de “Ethereum”.

Esta máquina virtual posibilita el desarrollo de aplicaciones, ya que está construida en el lenguaje especializado de alto nivel llamado Solidity. A través de este lenguaje de programación se facilita la creación de contratos inteligentes. En primer lugar, se transforma el código Solidity a los códigos de operación (OP_CODES) y luego a un bytecode. Este bytecode se ejecuta por parte de la EVM para realizar las operaciones especificadas en un contrato inteligente. Esto hace que la EVM pueda funcionar como un computador de verdad, ejecutando desde las más sencillas hasta las más complejas operaciones.

A continuación, se presentan algunas de las características más relevantes de la EVM:

1. **Ayuda a proporcionar un alto nivel de seguridad:** al ser una máquina virtual con limitaciones en las instrucciones y en la forma en cómo se ejecutan, EVM es capaz de ejecutar códigos no confiables sin consecuencias desastrosas.
2. **EVM es una construcción completamente descentralizada:** cada nodo dentro de la red Ethereum ejecuta una copia de esta máquina virtual en conjunto y sincronía con el resto de nodos que forman la red. Esto garantiza que las instrucciones dadas las EVM se ejecuten siempre y cuando exista al menos un nodo en activo. Esto permite el acceso a dicho sistema desde cualquier parte del mundo, resistiendo la censura y garantizando el acceso a los recursos de la red. Además, no requiere de la participación de terceros, tampoco pueden ser modificadas ni alteradas.
3. **La EVM permite el desarrollo de una mayor cantidad de aplicaciones:** permite que éstas puedan ejecutarse sobre una misma red blockchain, sin afectar otras operaciones. Lo vemos claramente en el funcionamiento de las DApps de Ethereum y su explosión de desarrollo hasta la actualidad.
4. **EVM es capaz de hacer realidad a los “smart contracts” o contratos inteligentes de Turing completo:** estas son programaciones específicas e invariables que pueden ejecutarse y hacerse cumplir por sí mismos, de una manera autónoma y automática.
5. **La EVM posee la capacidad de ejecutar una serie de códigos de operación u “OP_CODES” bien definidos:** fuera de estos códigos la EVM no es capaz de realizar absolutamente nada. Pero para que esta ejecución sea posible, dichos “OP_CODES” deben transformarse en “bytecode” o código máquina. Esto permite que EVM tenga un mayor rendimiento en la realización de operaciones en comparación con aproximaciones como las de Bitcoin.

2.2. Instalación y Configuración

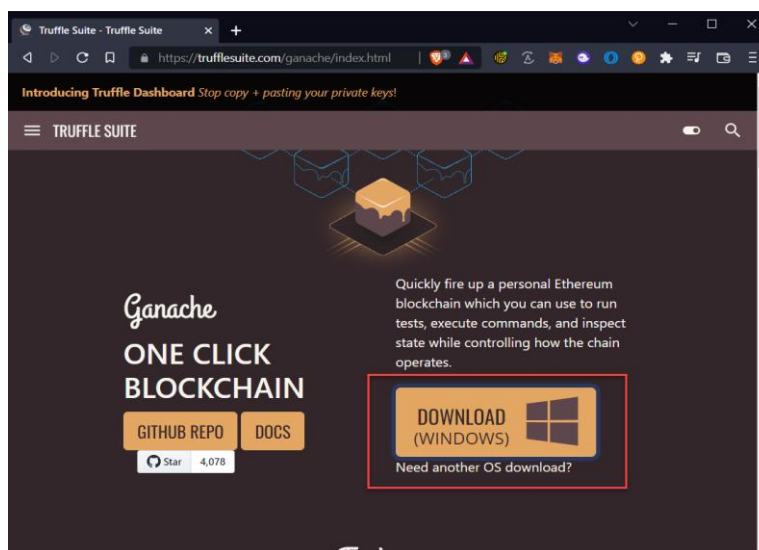
La red de Ethereum está abierta a la creación de nuevos nodos, además posibilita agregarse a la red de bloques, si es de nuestro interés, tan solo se debe hacer uso de sus herramientas para realizar un enlace, a continuación, se debe revisar cómo se realiza la creación de un nodo para la red Ethereum.

Para explicar la instalación y configuración básica de una red de bloques basada en Ethereum, se debe utilizar la solución “Ganache”, la cual es una aplicación que permite implementar un entorno de pruebas y simulación de una red basada en Ethereum. Se debe realizar la instalación como se explica a continuación.

Para obtener Ganache, se debe dirigir a su página oficial:

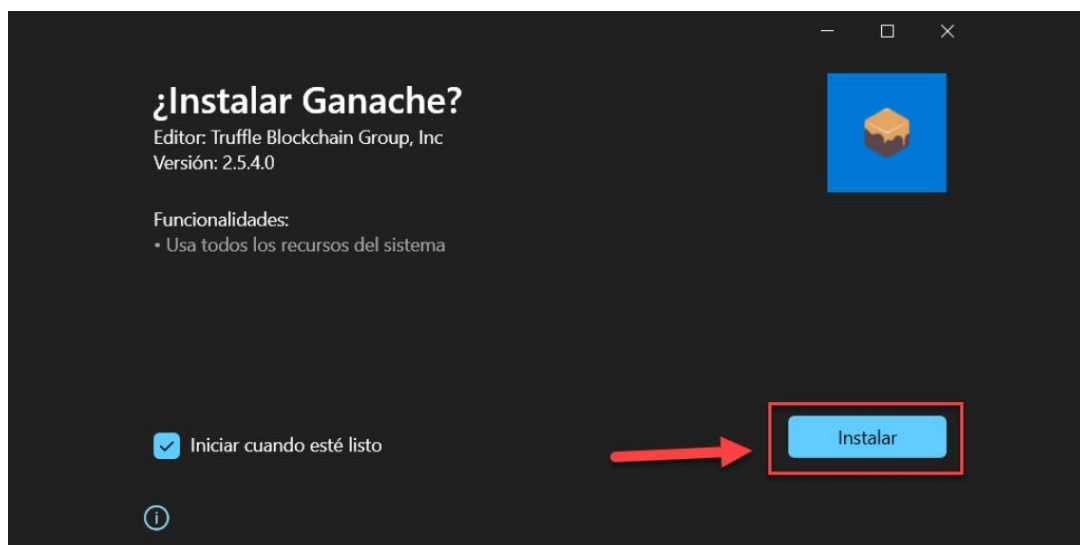
<https://trufflesuite.com/ganache/> y descargar la versión para Windows que se presenta por defecto, también se puede encontrar el instalador de la aplicación para otros sistemas operativos como Mac y Linux.

Figura 3. Página oficial para descargar la aplicación Ganache



Se descargará un paquete de instalación, al cual basta con darle doble clic para iniciar su instalación, así se iniciará una interfaz para abrir el asistente de instalación de Windows:

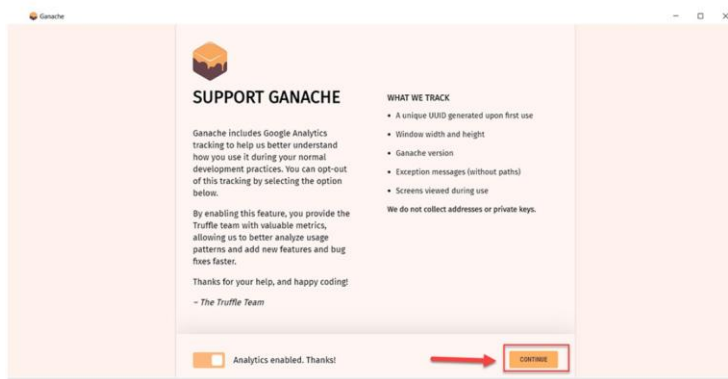
Figura 4. Lanzamiento de la instalación de Ganache



Se puede dejar marcada la opción de “Iniciar cuando esté listo” para que se realice el inicio tan pronto se termine la instalación.

A continuación, se abrirá la interfaz de Ganache y la primera vez, solicitará la confirmación para compartir información con Google Analytics; se puede dejar marcado para apoyar el proyecto y hacer clic en el botón “Continue”.

Figura 5. Primer inicio de la aplicación Ganache



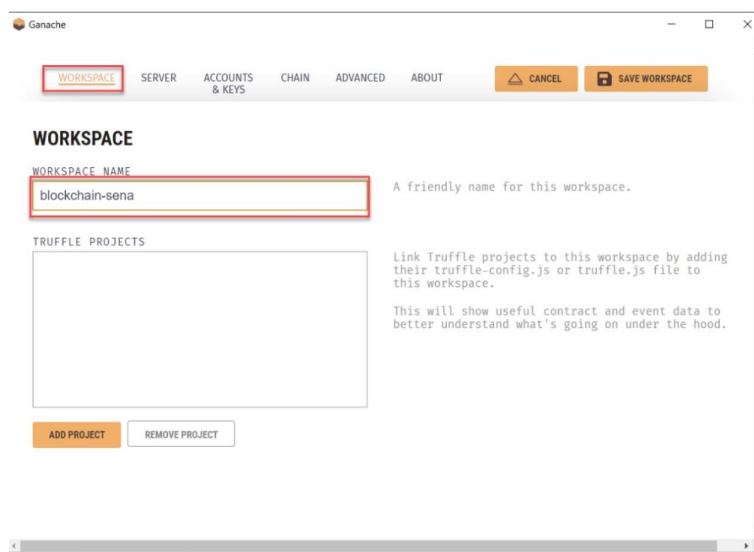
A continuación, se presentará la ventana de los “workspaces”, en la cual, se puede seleccionar la opción “New Workspace” y hacer clic en la flecha para desplegar las opciones; se debe elegir la opción “Ethereum” y finalmente en el botón “New Workspace”.

Figura 6. Selección del tipo de “workspace” a crear



Posteriormente, se continúa con las opciones para personalizar el “workspace”, en la primera pestaña de “Workspace”, se debe ingresar el nombre del espacio de trabajo.

Figura 7. Ingreso del nombre del espacio de trabajo

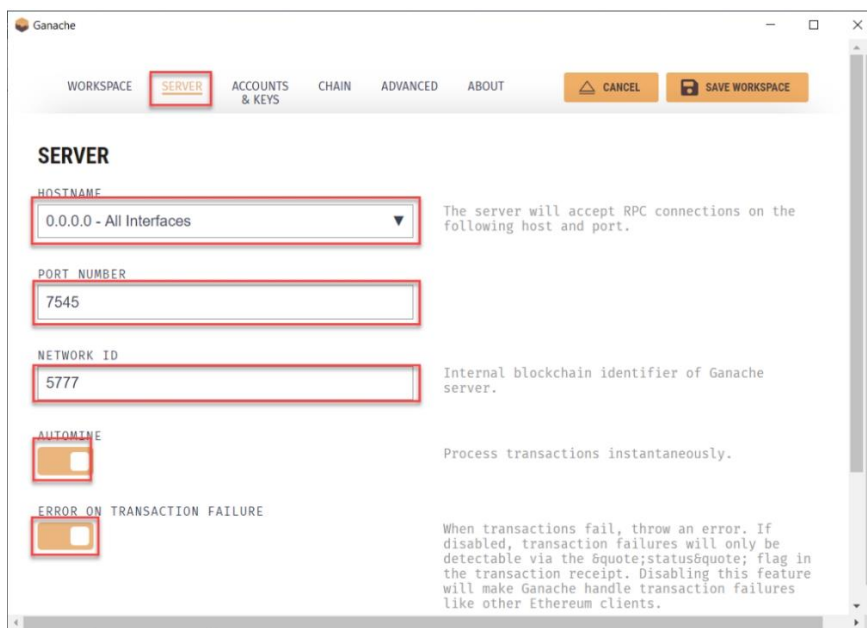


En la pestaña “Server” se puede personalizar el servicio para la conexión a la red “blockchain”, en la cual es posible configurar:

- “Hostname”**: selección de las interfaces de red por las que se necesita la conexión a la red “blockchain”, si es el caso que otros dispositivos requieran conectarse a la red, se debe seleccionar “0.0.0.0 All Interfaces”.
- “Port Number”**: número del puerto TCP que permitirá la conexión remotamente.
- “Network ID”**: número de identificador de la red “blockchain”.
- “Automine”**: para establecer si se realizara el auto minado.
- Error **“on Transaction Failure”**: presenta mensajes de error cuando una transacción no puede llevarse a cabo.

Lo anterior, se presenta en la siguiente figura:

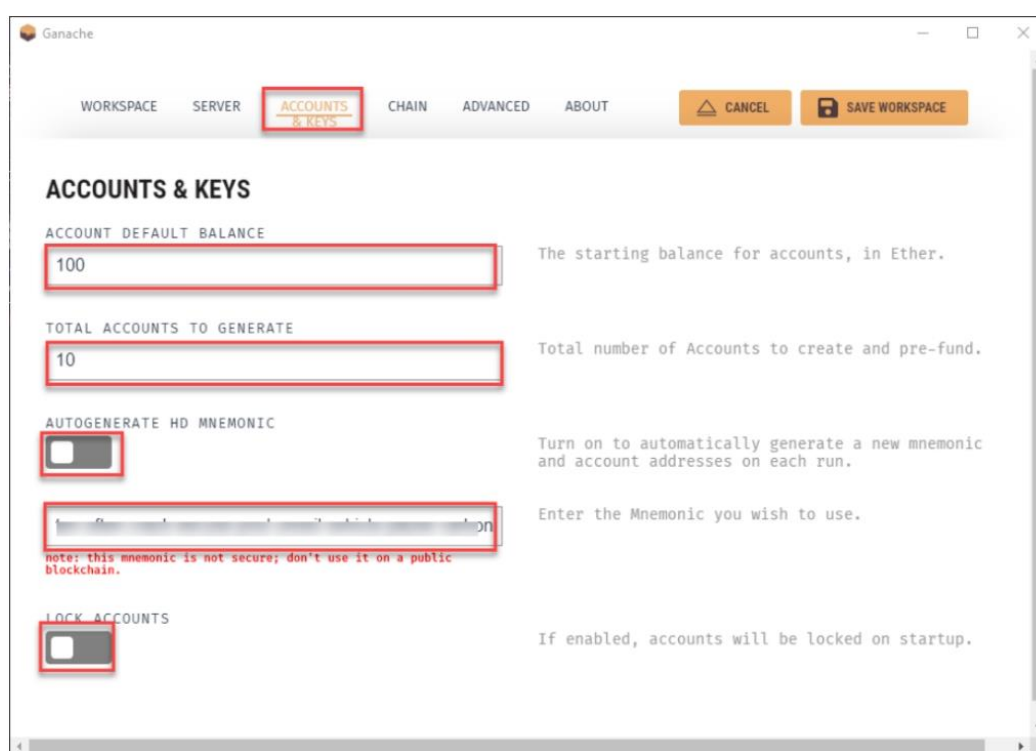
Figura 8. Ingreso del nombre del espacio de trabajo



En la pestaña de cuentas y claves, se pueden configurar las cuentas que serán creadas en la red, y se establecen los siguientes parámetros, como se indica a continuación:

- a) **“Account Default Balance”**: Balance inicial en Ether de cada una de las cuentas.
- b) Total, **“Accounts to Generate”**: cantidad de cuentas a crear.
- c) **“Autogenerate HD Mnemonic”**: seleccionar si se generará una llave privada o se le asignará una, en caso de seleccionar No: se debe suministrar la llave.
- d) **“Lock Accounts”**: configura si las cuentas iniciarán en estado bloqueado.

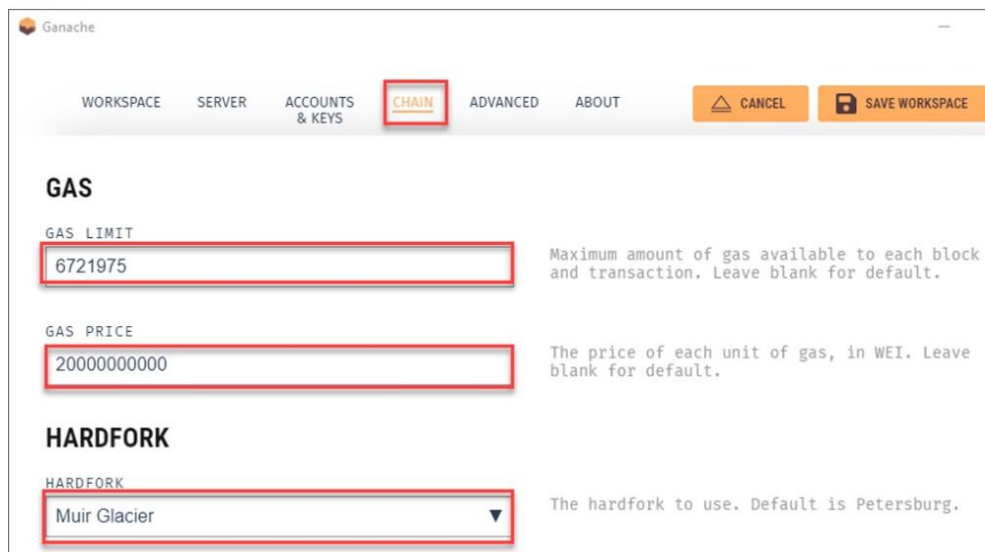
Figura 9. Gestión de las cuentas para crear en la red



En otras pestañas que trae la interfaz “Chain”, se puede configurar el “hardfork” de la red, es decir sobre la versión base, así como los gastos por comisión Gas:

- a) **Gas “Limit”**: monto máximo por cada bloque o transacción, valor en Ether.
- b) **Gas “Price”**: precio por cada unidad de Gas, valor en WEI.
- c) **“HardFork”**: versión del “hardfork” para la red.

Figura 10. Configuración de la cadena y valor del Gas de comisión



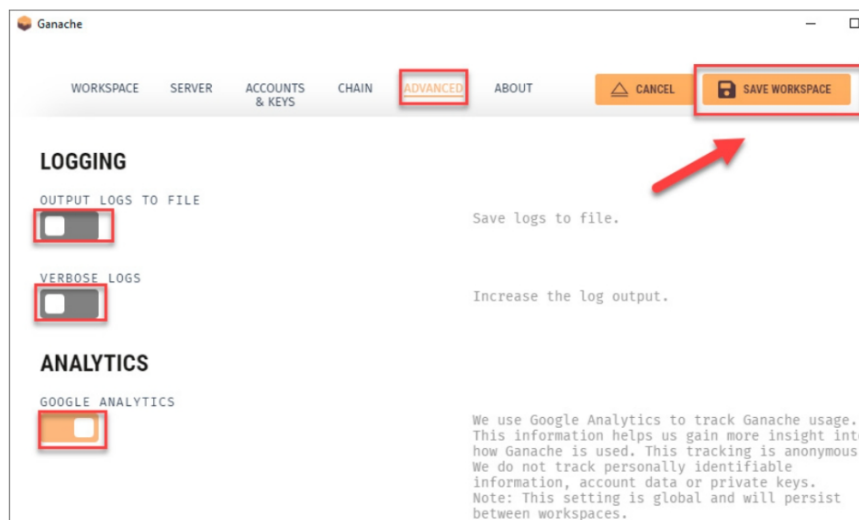
The screenshot shows the 'CHAIN' tab in the Ganache application. The 'GAS' section has two input fields: 'GAS LIMIT' with the value '6721975' and 'GAS PRICE' with the value '20000000000'. The 'HARDFORK' section has a dropdown menu with 'Muir Glacier' selected. The 'CHAIN' tab is highlighted in the top navigation bar. The 'SAVE WORKSPACE' button is visible in the top right corner.

En la pestaña “Advanced” de la figura anterior, se pueden configurar los parámetros de “log's” y nuevamente configurar si se comparte información con Google Analytics, así:

1. “Output Longs to File”: determina si guardara los registros de log, si seleccionamos que si debemos indicar la ruta para su registro.
2. “Verbose Longs”: permite logs con mayor detalle.
3. Google Analytics configura si comparte información con Google Analytics o no.

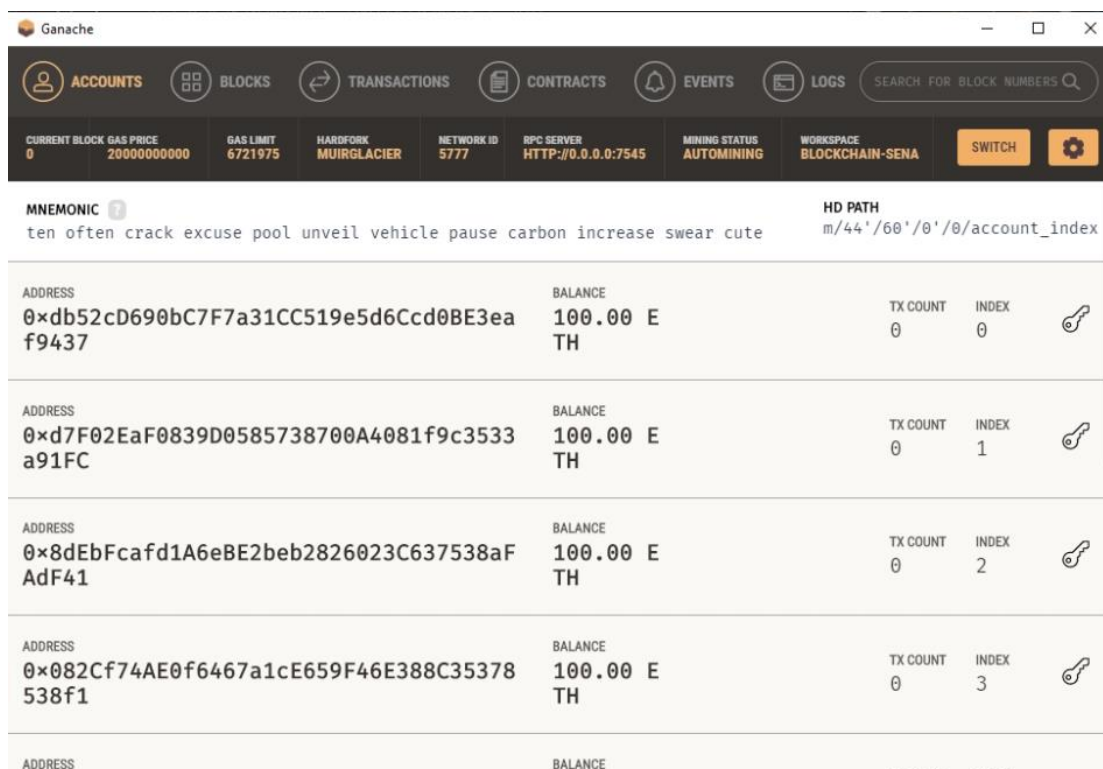
Finalmente, cuando se haya realizado y revisado la configuración, se puede hacer clic en el botón “Save Workspace”, para iniciar el espacio de trabajo.

Figura 11. Configuración avanzada de la red y finalización de la configuración



Una vez se haya culminado el proceso de configuración, se puede visualizar una interfaz del espacio de trabajo, sus cuentas y resumen de las transacciones realizadas.

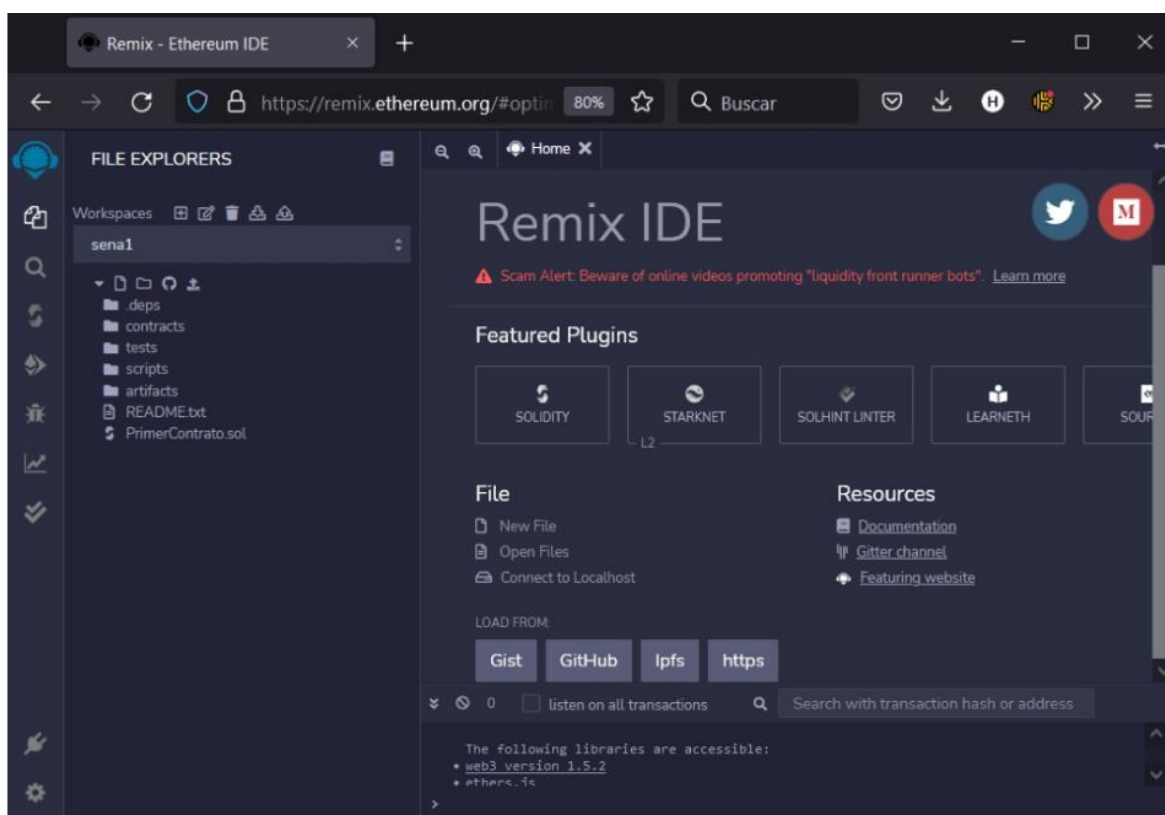
Figura 12. Pantalla principal del espacio de trabajo configurado



En esta aplicación “Ganache” se logra realizar simulaciones, pruebas, crear contratos inteligentes e incluso interconectar con otros servicios como billeteras para consultar el saldo, se invita a explorar.

En caso que el “hardware” con el que se cuenta no tenga las capacidades para que Ganache funcione, se podrá hacer uso de una aplicación web que permite probar los contratos inteligentes sin necesidad de instalaciones, esta se llama Remix Ethereum IDE y se puede acceder [aquí](#).

Figura 13. Remix Ethereum IDE



Lo anterior, presenta una interfaz general de compilador, un explorador de archivos de cada proyecto y un área de trabajo en donde se puede codificar los proyectos, es muy fácil

de usar e intuitivo, además permite crear cuentas con recursos de prueba, necesarios para el desarrollo.

Estas dos soluciones pueden ser muy útiles para desarrollar y probar los contratos inteligentes sin gastar recursos reales bajo una red de “blockchain”.

3. Principios básicos de Solidity

Solidity es un lenguaje de alto nivel, orientado a contratos y de tipo estático para implementar contratos inteligentes en la plataforma Ethereum, con el cual pueden generarse los contratos inteligentes bajo la red de Ethereum, su sintaxis es muy similar a la de lenguajes muy conocidos como C++ o Javascript.

Las siguientes son algunas características del lenguaje de programación Solidity:

1. **Lenguaje:** Solidity es un tipo de lenguaje orientado a objetos (OO) como C++ y C#.
2. **Diseño y desarrollo de aplicaciones:** fue diseñado específicamente para desarrollar aplicaciones para la red Ethereum. Por lo que sólo corre en la “blockchain” Ethereum o derivadas compatibles.
3. **Código fuente:** una ventaja que posee este lenguaje es que el código fuente de los programas que son realizados con el mismo puede ser accedido públicamente desde la “blockchain”. Incluso estando en “bytecode”, es posible descompilarlo y obtener una muestra bastante clara del código fuente original.
4. **Herencia y elementos complejos:** Solidity soporta la herencia y la herencia múltiple, dotándolo de una gran flexibilidad a la hora de programar elementos complejos.
5. **Bibliotecas:** este lenguaje también proporciona una serie de bibliotecas muy útiles que facilitan la programación de los contratos inteligentes, haciendo que el código sea reutilizable y más sencillo de mantener.

3.1. Expresiones

Los contratos en Solidity son semejantes a las clases de lenguajes orientados a objetos. Pueden contener declaraciones del tipo de variables de estado, funciones, modificadores de función, además de eventos, structs y enums. Algunas de las sintaxis más utilizadas en estos contratos son:

Variables

Alcance de las variables

Operadores

Tipos de valor

“Array”

A continuación, se encontrará una explicación detallada de variables:

- a) **Variables de estado:** variables cuyos valores se almacenan permanentemente en un almacenamiento de contrato.

```
pragma solidity ^0.5.0;
contract SolidityTest {
    uint storedData; //Definición variable de estado
    constructor() public {
        storedData = 10; //Uso de variable de estado
    }
}
```

- b) **Variables de locales:** variables cuyos valores están presentes hasta que la función se está ejecutando.

```
pragma solidity ^0.5.0;
contract SolidityTest {
```



```

uint storedData; //Definición variable de estado
constructor() public {
    storedData = 10;

}

function getResult() public view returns(uint){
    uint a = 1; //Definición de variable local
    uint b = 2;
    uint result = a + b;
    return result; //Uso de variable local
}

}

```

- c) **Variables globales:** existen variables especiales en el espacio de nombres global utilizado para obtener información sobre la cadena de bloque.

Tabla 2. Variables globales

Nombre de variable global	Devuelve
"Blochash (uint blockNumber) returns (bytes 32)"	Hash del bloque dado: solo funciona para los bloques más recientes, excluyendo los actuales,
"block.coinbase (address payable)"	Dirección actual del minero de bloques
"block. difficulty (uint)"	Dificultad del bloque actual
"block.gaslimit (uint)"	Límite de gas del bloque de corriente
"block.number (uint)"	Número de bloque actual

Nombre de variable global	Devuelve
"block.timestamp (uint) "	Marca de tiempo de bloque actual como segundos desde la época de unix
"gasleft() returns (uint256) "	Gas restante
"msg.data (bytes calldata) "	Datos de llamada completos
"msg.sender (address payable) "	Remitente del mensaje (persona que llama actualmente)
"msg.sig (bytes4)"	Primeros cuatro bytes de los datos de llamada (identificador de función)
"msg.value (uint) "	Número de wei enviados con el mensaje
"now (uint) "	Marca de tiempo de bloque actual
"tx.gasprice (uint) "	Precio del gas de la transacción
"tx.origin (address payable) "	Remitente de la transacción

Como todo lenguaje de programación, presenta algunas recomendaciones en el momento de declarar variables, algunas de estas son:

1. No utilizar ninguna de las palabras clave reservadas de Solidity como nombre de variable.
2. Los nombres de las variables de solidez no deben comenzar con un numeral (0-9). Deben comenzar con una letra o un carácter de subrayado. Por ejemplo, 123test es un nombre de variable no válido, pero _123test es válido.
3. Los nombres de las variables distinguen entre mayúsculas y minúsculas. Por ejemplo, Nombre y nombre son dos variables diferentes.

Alcance de las variables: el alcance de las variables locales se limita a la función en la que se definen, pero las variables de estado pueden tener tres tipos de ámbitos:

1. **“Public”**: permite acceder a las variables de estado público tanto internamente como a través de mensajes. Para una variable de estado público.
2. **“Internal”**: solo permite el acceso a las variables de estado interno desde el contrato actual o el contrato derivado de él sin usar esto.
3. **“Private”**: estas variables sólo pueden ser accedidas internamente desde el contrato actual que están definidas no en el contrato derivado del mismo.

Los siguientes códigos dan claridad frente a la definición y uso de alcance de las variables, establecido anteriormente:

```
pragma solidity ^0.5.0;
contract C {
    uint public data = 30;

    uint internal iData= 10;
    function x() public returns (uint) {
        data = 3; // internal access
        return data;
    }
}

contract Caller {
    C c = new C();
    function f() public view returns (uint) {
        return c.data(); //external access
    }
}
```

```
}  
contract D is C {  
    function y() public returns (uint) {  
        iData = 3; // internal access  
        return iData;  
    }  
    function getResult() public view returns(uint){  
        uint a = 1; // local variable  
        uint b = 2;  
        uint result = a + b;  
        return storedData; //access the state variable  
    }  
}
```

Otra expresión a tener en cuenta son los Operadores. Los operadores como su nombre lo indica, permiten realizar operaciones y asignaciones de valor entre variables, por ejemplo, en la siguiente expresión $5 + 7$ es igual a 12. Aquí 5 y 7 se llaman operandos y '+' se llama el operador.

Solidity soporta diferentes tipos de operadores que se explican a continuación:

- a) **Operadores aritméticos:** Solidity admite los siguientes operadores aritméticos, Supongamos que la variable A tiene 10 y la variable B tiene 20, entonces revisemos la siguiente tabla de operadores aritméticos para ver cómo se comportan cada uno de ellos.

Tabla 3. Operadores aritméticos

Operador	Descripción	Ejemplo
+ Adición	Suma dos operandos	$(A + N) = 30$
- Resta	Resta el segundo operando del primero	$(A - B) = -10$
X Multiplicación	Multiplica ambos operandos	$(A * B) = 200$
/ División	Dividir el numerador por el denominador	$(B / A) = 2$
% Mod.	Genera el resto de una división entera	$(B \% A) = 0$
++ Incremento	Aumenta un valor entero en uno	$(A++) = 11$
-- Decremento	Disminuye un valor entero en uno	$(A--) = 9$

- b) **Operadores de comparación:** Solidity admite los siguientes operadores de comparación, Supongamos que la variable A contiene 10 y la variable B contiene 20, entonces, entonces revisemos la siguiente tabla de operadores de comparación para ver cómo se comportan cada uno de ellos.

Tabla 4. Operadores de comparación

Operador	Descripción	Ejemplo
== Igual	Comprueba si el valor de dos operandos es igual o no, en caso afirmativo, entonces la condición se convierte en verdadera.	$(A == B)$ no es verdadero
!= No es igual	Comprueba si el valor de dos operandos es igual o no, si los valores no son iguales, entonces la condición se convierte en verdadera.	$(A != B)$ es verdadero
> Mayor que	Comprueba si el valor del operando izquierdo es mayor que el valor del operando derecho, en caso afirmativo, entonces la condición se convierte en verdadera.	$(A > B)$ no es verdadero

< Menor que	Comprueba si el valor del operando izquierdo es menor que el valor del operando derecho, en caso afirmativo, entonces la condición se convierte en verdadero.	(A < B) es verdadero
>= Mayor o igual que	Comprueba si el valor del operando izquierdo es mayor o igual que el valor del operando derecho, en caso afirmativo, entonces la condición se convierte en verdadera.	(A >= B) no es verdadero
<= Menor o Igual que	Comprueba si el valor del operando izquierdo es menor o igual que el valor del operando derecho, en caso afirmativo, entonces la condición se convierte en verdadera.	(A <= B) es verdadero

- c) **Operadores de lógicos:** Solidity admite los siguientes operadores lógicos, Supongamos que la variable A contiene 10 y la variable B contiene 20, entonces, entonces revisemos la siguiente tabla de operadores lógicos para ver cómo se comportan cada uno de ellos.

Tabla 5. Operadores de lógicos

Operador	Descripción	Ejemplo
&& AND Lógico	Si ambos operandos son distintos de cero, entonces la condición se vuelve verdadera.	(A && B) es verdadero
OR Lógico	Si alguno de los dos operandos es distinto de cero, entonces la condición se convierte en verdadera.	(A B) es verdadero
! NOT Lógico	Invierte el estado lógico de su operando. Si una condición es true, el operador Lógico NOT la hará false.	!(A > B) no es verdadero

- d) **Operadores “bitwise”:** Solidity admite los siguientes operadores “bitwise”, Supongamos que la variable A contiene 2 y la variable B contiene 3, entonces,

entonces revisemos la siguiente tabla de operadores “bitwise” para ver cómo se comportan cada uno de ellos.

Tabla 6. Operadores “bitwise”

Operador	Descripción	Ejemplo
& “Bitwise” AND	Realiza una operación booleana AND en cada bit de sus argumentos enteros.	$(A \& B) = 2$
“Bitwise” OR	Realiza una operación OR booleana en cada bit de sus argumentos enteros.	$(Un \mid B) = 3$
^ “Bitwise” XOR	Realiza una operación OR exclusiva booleana en cada bit de sus argumentos enteros. Exclusivo OR significa que el operando uno es verdadero o el operando dos es verdadero, pero no ambos.	$(A \wedge B) = 1$
~ “Bitwise” Not	Es un operador unario y opera invirtiendo todos los bits en el operando.	$(\sim B) = -4$
<< desplazamiento a la izquierda	Mueve todos los bits de su primer operando a la izquierda por el número de lugares especificados en el segundo operando. Los nuevos bits se llenan de ceros. Desplazar un valor dejado por una posición equivale a multiplicarlo por 2, desplazar dos posiciones es equivalente a multiplicar por 4, y así sucesivamente.	$(A << 1) = 4$
>> desplazamiento a la derecha	Operador binario de desplazamiento a la derecha. El valor del operando izquierdo se mueve a la derecha por el número de bits especificado por el operando derecho.	$(A >> 1) = 1$
>>> desplazamiento a la derecha con cero	Este operador es como el operador >>, excepto que los bits desplazados a la izquierda son siempre cero.	$(A >>> 1) = 1$

- e) **Operadores de asignación:** Solidity admite los siguientes operadores de asignación, revisemos la siguiente tabla de operadores “bitwise” para ver cómo se comportan cada uno de ellos.

Tabla 7. Operadores de asignación

Operador	Descripción	Ejemplo
= Asignación simple	Asigna valores del operando del lado derecho al operando del lado izquierdo.	$C = A + B$ asignará el valor de $A + B$ a C
+= Agregar y asignar	Agrega el operando derecho al operando izquierdo y asigna el resultado al operando izquierdo.	$C += A$ es equivalente a $C = C + A$
-= Restar y asignar	Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo.	$C -= A$ es equivalente a $C = C - A$
*= Multiplicar y Asignar	Multiplica el operando derecho con el operando izquierdo y asigna el resultado al operando izquierdo.	$C *= A$ es equivalente a $C = C * A$
/= Dividir y asignar	Divide el operando izquierdo con el operando derecho y asigna el resultado al operando izquierdo.	$C /= A$ es equivalente a $C = C / A$
%= Módulos y Asignación	Toma el módulo usando dos operandos y asigna el resultado al operando izquierdo.	$C \% = A$ es equivalente a $C = C \% A$

Nota: La misma lógica se aplica a los operadores “Bitwise”, por lo que se volverán como $\ll=$, $\gg=$, $\&=$, $|=$ y $\^=$

- d) **Operador condicional:** el operador condicional primero evalúa una expresión para un valor verdadero o falso y, a continuación, ejecuta una de las dos instrucciones dadas en función del resultado de la evaluación.

Tabla 8. Operador condicional

Operador	Descripción y ejemplo
? : (Condicional)	¿ Si la condición es verdadera? Entonces valor X : De lo contrario valor Y

En relación a los Tipos de valor, el lenguaje de programación ofrece una amplia gama de tipos de datos con los cuales se almacena información temporalmente mientras se encuentra en ejecución un contrato inteligente, a continuación, se puede ver la tabla 2 de tipos de datos aceptados.

Tabla 9. Operador condicional

Operador	Descripción	Ejemplo
Booleano	bool	verdadero/falso.
Entero	int/uint	Enteros con y sin signo de diferentes tamaños.
Entero	int8 a int256	Firmado int de 8 bits a 256 bits. int256 es lo mismo que int.
Entero	uint8 a uint256	Int sin firmar de 8 bits a 256 bits. uint256 es lo mismo que uint.
Números de puntos fijos	fijo/no fijo	Números de punto fijos firmados y sin firmar de diferentes tamaños.
Números de puntos fijos	fixedMxN	Número de punto fijo firmado donde M representa el número de bits tomados por tipo y N representa los puntos decimales. M debe ser divisible por 8 y va de 8 a 256. N puede ser de 0 a 80. fixed es igual que fixed128x18.
Números de puntos fijos	ufixedMxN	Número de punto fijo sin signo donde M representa el número de bits tomados por tipo y N representa los puntos decimales. M debe ser divisible por 8 y va de 8 a 256. N puede ser de 0 a 80. ufixed es lo mismo que ufixed128x18.

Adicionalmente existe un tipo predefinido tipo “address”, el cual permite almacenar una cadena de 20 bytes de longitud que representa una dirección de Ethereum:

Declaración y uso de una variable de tipo “address”

```
address x = 0x212;  
  
address myAddress = this;  
  
if (x.balance < 10 && myAddress.balance >= 10) x.transfer(10);
```

“Array”: Los arreglos al igual que en los demás lenguajes de programación, son estructuras de datos, que almacenan una colección de elementos del mismo tipo.

Para la declaración de un array en solidity, debemos especificar el tamaño es decir la cantidad de elementos que contendrá y el tipo de dato, así

```
type arrayName [ arraySize ];
```

Por ejemplo, para declarar una matriz de 10 elementos denominada balance de tipo uint

Declaración de un array de tipo uint de 10 posiciones

```
uint balance[10];
```

Si deseamos declarar una matriz de tamaño dinámico en Solidity, debemos de especificar el tipo de elementos de la siguiente manera:

Declaración de un array de tamaño dinámico

```
uint[] memory a = new uint[](7);
```

Ahora bien, si requerimos acceder a una posición para leer o escribir un dato, podemos acceder invocando la posición que deseamos trabajar, así:

Lectura de una posición de un array

```
uint salary = balance[2];
```

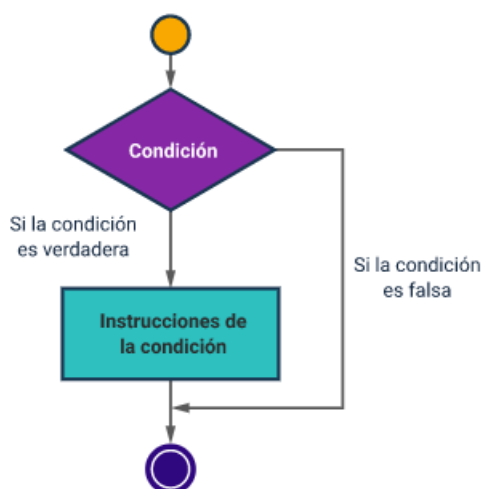
3.2. Estructuras de control

Las estructuras de control dentro de un lenguaje de programación, nos permiten establecer la toma de decisiones a partir del cumplimiento de condiciones, entre las estructuras de control podemos encontrar:

If-else: esta estructura de control permite tomar decisiones a partir del cumplimiento de condiciones específicas y con comportamiento definido, presenta las siguientes formas de declararse:

- a) **Instrucción if:** esta es la instrucción de control fundamental que permite al lenguaje de programación tomar decisiones y ejecutar instrucciones condicionalmente.
- b) **If ... else:** la instrucción 'if... else' es la siguiente forma de controlar la instrucción de control y que permite al lenguaje de programación ejecutar instrucciones de una manera más controlada.
- c) **If ... else if... :** esta declaración es una forma avanzada de si... de lo contrario, permite al lenguaje de programación tomar una decisión correcta a partir de varias condiciones.

Figura 14. Instrucción if else



3.3. Estructuras repetitivas

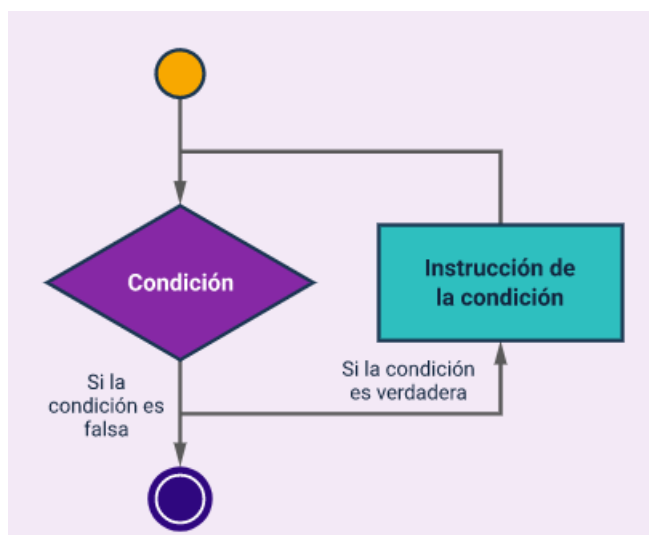
Las estructuras repetitivas o también conocidas como bucles, nos permiten repetir cierto número de veces una acción, mientras se cumpla una condición limitante, dentro de las estructuras repetitivas utilizadas en solidity tenemos:

“For”: esta estructura de control iterativa, nos permite ejecutar de manera repetitiva un bloque de instrucciones, conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo, y su declaración está compuesta por los siguientes 3 elementos:

1. **La inicialización de bucle**: donde inicializamos nuestro contador a un valor inicial. La instrucción de inicialización se ejecuta antes de que comience el bucle.
2. **La declaración de prueba**: que probará si una condición dada es verdadera o no. Si la condición es verdadera, entonces se ejecutará el código dado dentro del bucle, de lo contrario el control saldrá del bucle.
3. **La instrucción de iteración**: en la que puede aumentar o disminuir el contador.

Estos elementos se agregan en una línea al inicio de bucle separados por punto y coma.

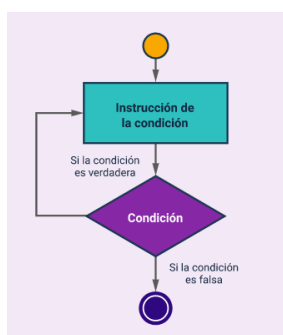
Figura 15. Ciclo “For”



While: esta también es una estructura cíclica, que nos permite ejecutar sentencias de manera repetitiva sin necesidad de tener un valor inicial o final, esta estructura no depende de valores numéricos, sino de valores booleanos, es decir continúa con su ciclo mientras una condición sea verdadera, en caso contrario terminará, a continuación, vamos a ver el flujo de la estructura while.

Do: El ciclo do... while trabaja de manera similar al ciclo while con excepción que la comprobación de condición ocurre al final del bucle. Esto significa que el bucle siempre se ejecutará al menos una vez, incluso si la condición es falsa.

Figura 16. Ciclo Do



Con el uso de las estructuras de control podemos programar la lógica de negocio de un contrato inteligente, de acuerdo a nuestras necesidades y comportamiento deseado, a continuación, podemos observar cómo se implementaría en código solidity:

```
pragma solidity ^0.5.0;

contract SolidityTest {
    uint storedData;

    constructor() public{
        storedData = 10;
    }

    function getResult() public view returns(string memory){
        uint a = 1;
        uint b = 2;
        uint result = a + b;
        return integerToString(result);
    }

    function integerToString(uint _i) internal pure
    returns (string memory) {

        if (_i == 0) {
            return "0";
        }
        uint j = _i;
        uint len;

        while (true) {
            len++;
        }
    }
}
```

```
        j /= 10;
        if(j==0){
            break;
        }
    }
    bytes memory bstr = new bytes(len);
    uint k = len - 1;

    while (_i != 0) {
        bstr[k--] = byte(uint8(48 + _i % 10));
        _i /= 10;
    }
    return string(bstr);
}
}
```

Es así como nos pueden ayudar a implementar controles, validaciones y establecer el flujo del proceso y de la información en la construcción de un contrato inteligente.

3.4. Funciones

Una función son fragmentos de código reutilizable al que se puede llamar en cualquier parte del programa. Esto elimina la necesidad de escribir el mismo código una y otra vez, facilitando su modularidad.

El primer paso es realizar la definición, la cual se realiza haciendo uso de la palabra clave “function”, seguida del nombre el cual debe ser único dentro del programa, posteriormente vendrán los parámetros los cuales son opcionales y finalmente el bloque de instrucciones.

Una vez se defina la función, podemos invocarla desde algún lugar del programa, tan solo con citar el nombre de la función y cargar los parámetros en caso de ser necesarios y hayan sido implementados en la función.

A continuación, podemos ver un ejemplo de una declaración de una función que suma dos números y retorna el resultado, el cual puede ser recibido y asignado en una variable que debe tener el mismo tipo de dato que la función:

```
pragma solidity ^0.5.0;

contract Test {
    function sumaNumeros(uint A, uint B) public view returns(uint){
        uint result = (A + B);
        return result;
    }

    uint numeroUno = 5;
    uint numeroDos = 3;
    uint suma = 0;

    suma = sumaNumeros(numeroUno, numeroDos);
}
```

La devolución de datos en una función es opcional, es decir no es necesario implementarlo si no se requiere, además solidity permite crear funciones que retornen más de un valor, aunque esto debe de estar declarado en la función inicial.

Ejemplo de definición y uso de una función que retorna 2 valores en solidity:

```
pragma solidity ^0.5.0;

contract Test {
    function operaciones () public view returns(uint product, uint sum){
        uint a = 1;
        uint b = 2;
        product = a * b;
        sum = a + b;
    }
}
```



```
}
```

Es así como podemos implementar una estructura modular de operaciones dentro de nuestro contrato inteligente que nos permita optimizar nuestro código y aprovechar los recursos generados, podemos conocer más sobre como generar funciones en la documentación [oficial del lenguaje de programación](#).

4. Desplegar Contratos Inteligentes

Los contratos inteligentes van a funcionar sobre redes de “blockchain” como las de Ethereum, para todos los casos, cualquier consulta, envío de información o transacción va a requerir del uso de la red y de los gas o comisiones que ello conlleva, de acuerdo a esto, desarrollar un contrato inteligente necesitaría muchos recursos dado las pruebas y validaciones que tenemos que hacer antes de desplegar en una red oficial o de producción, para ello anteriormente habíamos presentado e instalado la plataforma de desarrollo denominada Ganache, sobre la cual podremos hacer las pruebas correspondientes antes de realizar su despliegue en una red oficial.

Para presentar el proceso de desarrollo y despliegue de un contrato inteligente vamos a partir de un contrato sencillo con una funcionalidad básica de sumar un valor a una variable declarada internamente, permitiendo consultar el valor final almacenado, a continuación, un modelo de contrato inteligente básico para pruebas:

```
//COMPILADOR
pragma solidity ^0.6.5;

//IMPORTAR OTROS CONTRATOS, INTERFACES O LIBRERIAS

//CUERPO DEL CONTRATO
contract PrimerContrato{

//LIBRERIAS
```

```
//DECLARACION DE VARIABLES
uint private myValue;

//EVENTOS
//Registra los eventos del programa
event LogSetMyValue (uint _myNewValue, address _sender);

//MODIFICADORES

//CONSTRUCTOR
constructor(uint _myValue) public {
    myValue = _myValue;
}

//FUNCIONES
//Lee el valor de la variable myValue
function getMyValue() public view returns (uint) {
    return myValue;
}

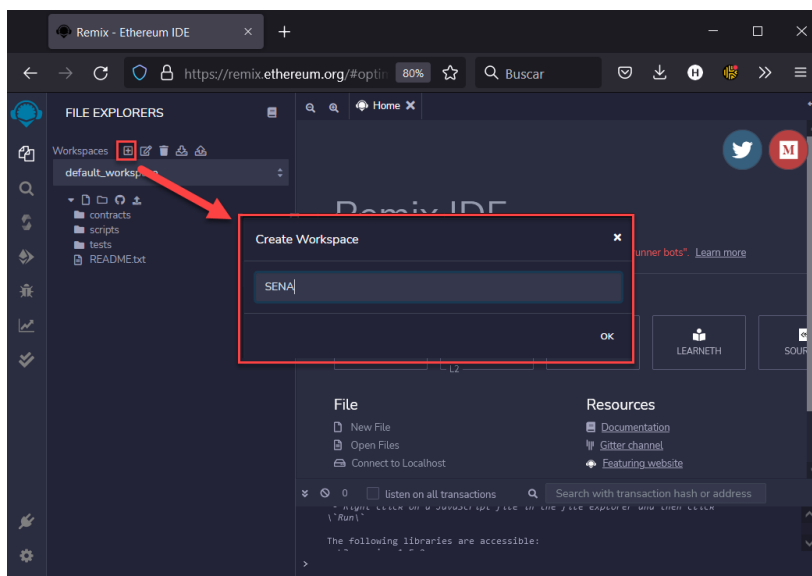
//Modifica el valor de la variable _myNewValue
function setMyValue( uint _myNewValue) public {
    myValue = _myNewValue;
    emit LogSetMyValue(myValue, msg.sender);
}
}
```

4.1. Entorno de prueba y desarrollo

Para realizar la programación y teste de nuestro contrato inteligente en una red de pruebas, podemos hacer uso de la plataforma Ganache o también del IDE Remix de

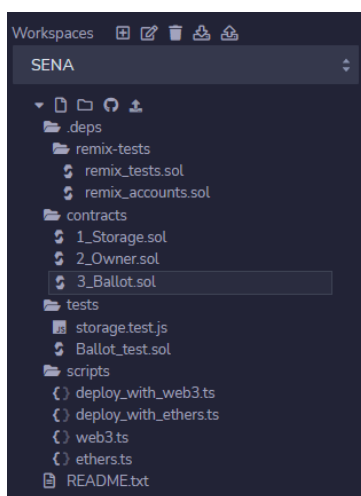
Ethereum, para lo cual nos dirigimos [aquí](#), en donde podremos crear un “workspace” o espacio de trabajo para cada proyecto, por ejemplo, en la siguiente figura vamos a crear un “workspace” denominado “SENA”.

Figura 17. Creación de Workspace en Remix Ethereum IDE



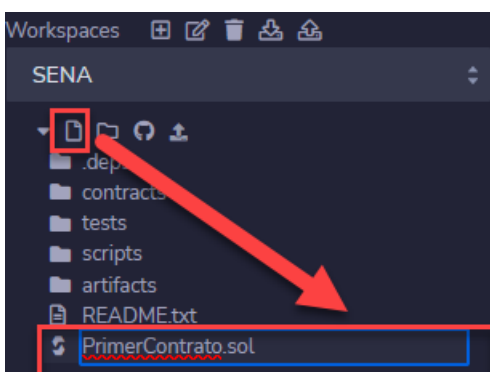
Al crear este “workspace” nos generara una estructura de archivos base como se muestra en la siguiente figura, pero no es necesario seguirla o utilizarla:

Figura 18. Estructura de archivos de un “Workspace”



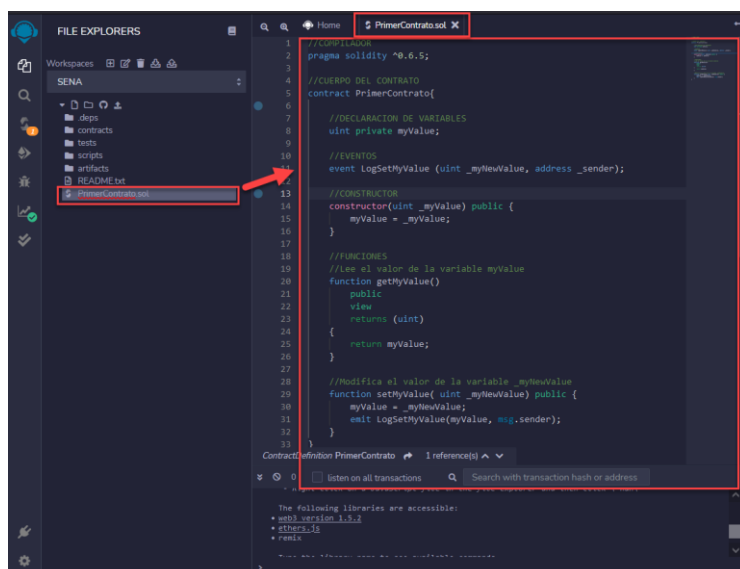
A continuación, procedemos a crear nuestro primer archivo del contrato, para nuestro caso será el archivo denominado “PrimerContrato.sol”:

Figura 19. Creación de un archivo en el “Workspace”



Ahora bien, dentro del archivo que hemos creado, podemos codificar nuestro contrato inteligente, en este caso vamos a pegar nuestro contrato de prueba:

Figura 20. Codificación de un contrato inteligente



Una vez hallamos codificado nuestro contrato inteligente, podemos ir a la opción de Compilación y seleccionar el entorno de compilación con el cual vamos a ejecutar nuestro contrato inteligente, entre la información a configurar encontramos:

1. **Versión del compilador:** entre las cuales podemos seleccionar versiones no muy recientes, como tampoco muy antiguas, esto debe ir en relación con la versión solicitada en la cabecera del contrato inteligente.
2. **Lenguaje:** seleccionar Solidity.
3. **Seleccionar la opción “Auto compile”:** esta opción permite compilar el Código mientras se escribe.

Existen más opciones, pero no son relevantes para el ejercicio, así que podemos hacer “click” en el botón de compilación, para validar el código.

Figura 21. Opciones de compilación de un contrato inteligente

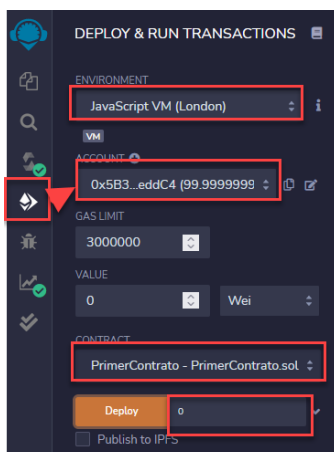


Una vez realizada la compilación, si no presentamos problemas, podemos proceder a realizar el despliegue del contrato inteligente en nuestra red de pruebas, para ello, procedemos a hacer “click” en el botón de Compilación de Solidity, en donde podremos seleccionar algunos elementos necesarios para realizar el despliegue del contrato inteligente:

1. **“Enviroment” /Entorno:** entorno de ejecución, dado que estamos haciendo pruebas, podremos hacer uso de la opción “Javascript VM” en cualquiera de sus versiones, esto nos lleva a trabajar sobre redes de prueba y que podemos utilizar recursos no reales.
2. **“Account” /Cuenta:** este entorno nos proporciona una serie de cuentas, con recursos disponibles, pero este recurso será únicamente de prueba y no será útil en una red de “blockchain” real, podemos escoger cualquiera de las cuentas que cuente con saldo.
3. **“Contract” /Contrato:** en esta opción vamos a seleccionar el archivo del contrato inteligente que acabamos de crear, para proceder con la compilación.
4. **Inicialización de variables:** para este caso, vamos a inicializar la variable que utilices en cero “0”.

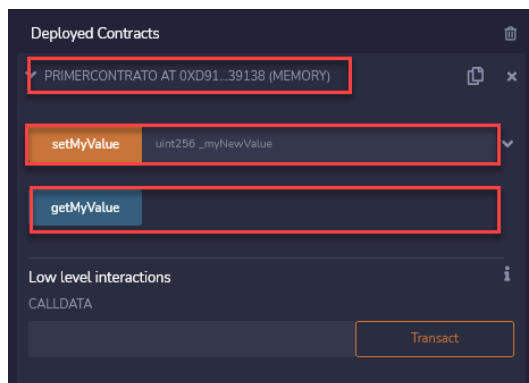
Finalmente podemos hacer “click” en el botón “Deploy” con el cual desplegamos el contrato seleccionado, y será visible dentro de la red:

Figura 22. Despliegue del contrato sobre la red



En la siguiente figura podemos visualizar el contrato inteligente desplegado, con una dirección visible a través de la red, y con los métodos codificados en nuestro ejemplo.

Figura 23. Consulta del contrato inteligente desplegado



Como podemos darnos cuenta, hacer uso de estas herramientas nos permiten codificar, validar y testear nuestros contratos inteligentes sin gastar recurso real, permitiendo corregir errores antes de llevarlos a una red de producción de “blockchain”.

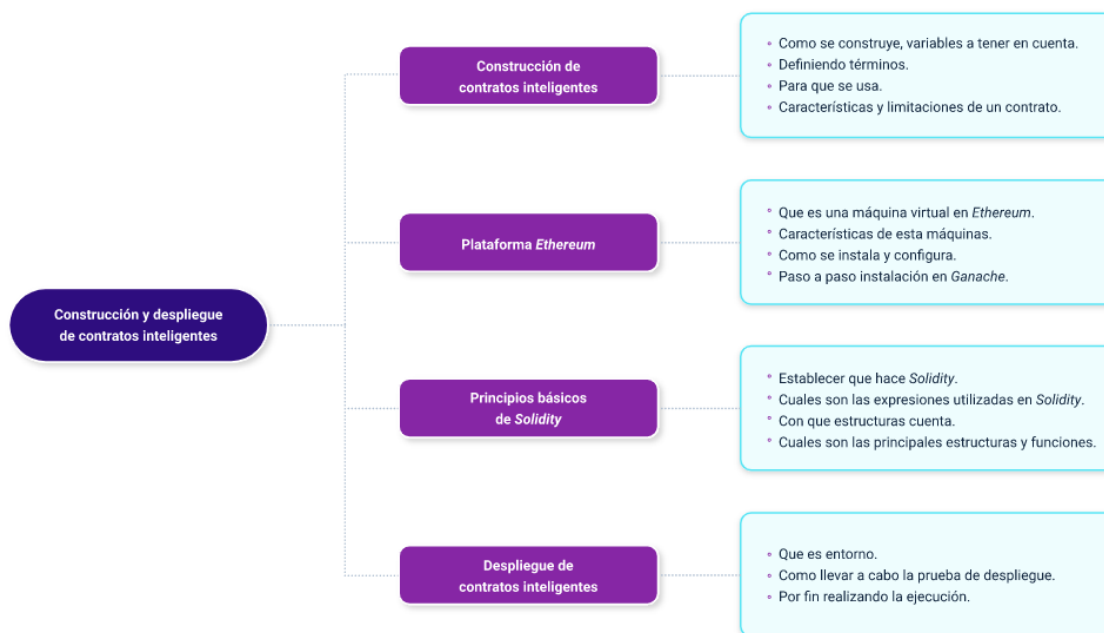
4.2. Entorno de ejecución

Así como se realiza el despliegue de un contrato inteligente en una red de pruebas, también se puede hacer uso de una de producción, estas redes trabajan con la de “blockchain” e interactúan con otros contratos inteligentes y DApps para ofrecer y consumir información, obviamente esto conlleva a que cada transacción haga uso de sus redes y requerirá de una comisión o Gas para poder llevar a cabo la transacción.

Es importante tener presente que la tecnología “blockchain” es una tecnología en desarrollo y día a día están surgiendo cambios que pueden llevar a que los procedimientos cambien o los gastos por comisiones cambien su costo, por lo que los invitamos a estar estudiando la tecnología y explorar las nuevas funcionalidades que estas soluciones pueden brindar a los entornos empresariales.

Síntesis

La construcción y despliegue de un contrato inteligente, es un proceso relativamente fácil de llevar a cabo, se debe hacer uso de las herramientas dispuestas para este propósito como Ethereum y Solidity, y tener en cuenta que el seguimiento y la aplicación del paso a paso establecido es lo que dará éxito en la tarea. A continuación, se presentan los principales aspectos relacionados con este interesante tema:



Material complementario

Tema	Referencia APA del Material	Tipo de material	Enlace del Recurso o Archivo del documento material
Construcción contratos inteligentes	Solidity. (2022). <i>Introducción a los contratos inteligentes</i> .	Página web	https://docs.soliditylang.org/en/v0.8.20/
Instalación y Configuración	Remix IDE. (2022). <i>Welcome to Remix's documentation</i> .	Manual electrónico	https://remix-ide.readthedocs.io/en/latest/
Instalación y Configuración	Ganache. (2022). <i>Ganache documentation</i> .	Manual electrónico	https://trufflesuite.com/docs/ganache/

Glosario

Compilador: programa que traduce un lenguaje de programación a un lenguaje comprensible por una máquina.

Constante: palabra que permite almacenar información de manera permanente mientras se ejecute un programa, sin posibilidad de actualizarlo.

“Deploy”: opción que permite desplegar un contrato inteligente dentro de una red de blockchain, para obtener su dirección de identificación.

Función: conjunto de código de instrucciones de un lenguaje de programación, que puede ser invocado desde otro lugar del mismo proyecto, algunas funciones pueden regresar valores o no.

Variable: es una palabra que representa un objeto y que su valor puede variar de acuerdo al tipo de información que recibe.

“Workspace”: es un espacio de trabajo en donde se almacenarán los archivos de un proyecto.

Referencias bibliográficas

Arroyo Guardes, D. Díaz Vico, J. & Hernández Encinas, L. (2019). Blockchain. Editorial CSIC Consejo Superior de Investigaciones Científicas. <https://elibro-net.bdigital.sena.edu.co/es/ereader/senavirtual/111431>

Fuentes Blanco, E. A. (2022). Contratos inteligentes: un análisis teórico desde la autonomía privada en el ordenamiento jurídico colombiano. 1. Editorial Unimagdalena. <https://elibro-net.bdigital.sena.edu.co/es/ereader/senavirtual/214513>

Ganache. (2022). *Ganache documentation*. <https://trufflesuite.com/docs/ganache>

LogRocker (2021) *Writing smart contracts with Solidity* <https://blog.logrocket.com/writing-smart-contracts-solidity>

MINTIC. (2022) *Guía de referencia de "blockchain" para la adopción e implementación de proyectos en el estado colombiano*. https://gobiernodigital.mintic.gov.co/692/articles-161810_Ley_2052_2020.pdf

Remix IDE. (2022). *Welcome to Remix's documentation*. <https://remix-ide.readthedocs.io/en/latest>

Shashank (2019). *What are Smart Contracts? A Beginner's Guide To Smart Contracts*. <https://www.edureka.co/blog/smart-contracts/>

Solidity (2022), *Solidity Documentation*. <https://solidity-es.readthedocs.io/es>

Créditos

Nombre	Cargo	Regional y Centro de Formación
Claudia Patricia Aristizábal Gutiérrez	Líder del equipo	Dirección General
Liliana Morales	Liliana Victoria Morales Gualdron	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Hernando José Peña Hidalgo	Experto Temático	Regional Cauca, Centro de teleinformática y producción industrial
María Inés Machado López	Diseñador instruccional	Regional Norte de Santander. Centro de la Industria, la Empresa y los Servicios
Alix Cecilia Chinchilla Rueda	Metodóloga para la formación virtual	Regional Distrito Capital. Centro de Diseño y Metrología
Rafael Neftalí Lizcano Reyes	Responsable del equipo de desarrollo curricular	Regional Santander. Centro Industrial del Diseño y la Manufactura
Jhon Jairo Rodríguez Pérez	Corrector de estilo	Regional Distrito Capital - Centro de Diseño y Metrología
Alix Cecilia Chinchilla Rueda	Metodóloga para la formación virtual	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Eulises Orduz Amezquita	Diseñador web	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Diego Fernando Velasco Güiza	Desarrollador Fullstack	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Ernesto Navarro Jaimes	Animador y Productor Multimedia	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Lina Marcela Pérez Manchego	Validación y vinculación en plataforma LMS	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital

Castaño Pérez Leyson Fabian	Validación y vinculación en plataforma LMS	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capita
Coca Salazar Carolina	Evaluador para Contenidos Inclusivos y Accesibles	Regional Distrito Capital- Centro de Gestión de Mercados, Logística y Tecnologías de la Información.