**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | *Blockchain*: contratos inteligentes |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501110. Implementar el sistema de seguridad de la información según modelo y estándares técnicos. | RESULTADOS DE APRENDIZAJE | 220501110-02 - Desarrollar contratos inteligentes en red de prueba. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 02 |
| NOMBRE DEL 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. |
| PALABRAS CLAVE | *Blockchain*, compilación, contrato, *Ethereum*, programa. |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 6. Ventas y servicios |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**

**Introducción**

1. **Construcción de contratos inteligentes** 
   1. Definición, características
   2. Limitaciones de contratos inteligentes
   3. Estructura del contrato
2. **Plataforma Ethereum**
   1. EVM Ethereum Virtual Machine
   2. Instalación y Configuración
3. **Principios básicos de *Solidity***
   1. Expresiones
   2. Estructuras de control
   3. Estructuras repetitivas
   4. Funciones
4. **Desplegar contratos inteligentes**
   1. Entorno de prueba y desarrollo
   2. Entorno de ejecución
5. **DESARROLLO DE CONTENIDOS:**

**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.

Vídeo

DI\_CF02\_Introducción

1. **Construcción de contratos inteligentes**

|  |  |
| --- | --- |
| Qué son los contratos inteligentes o smart contracts? | 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. |

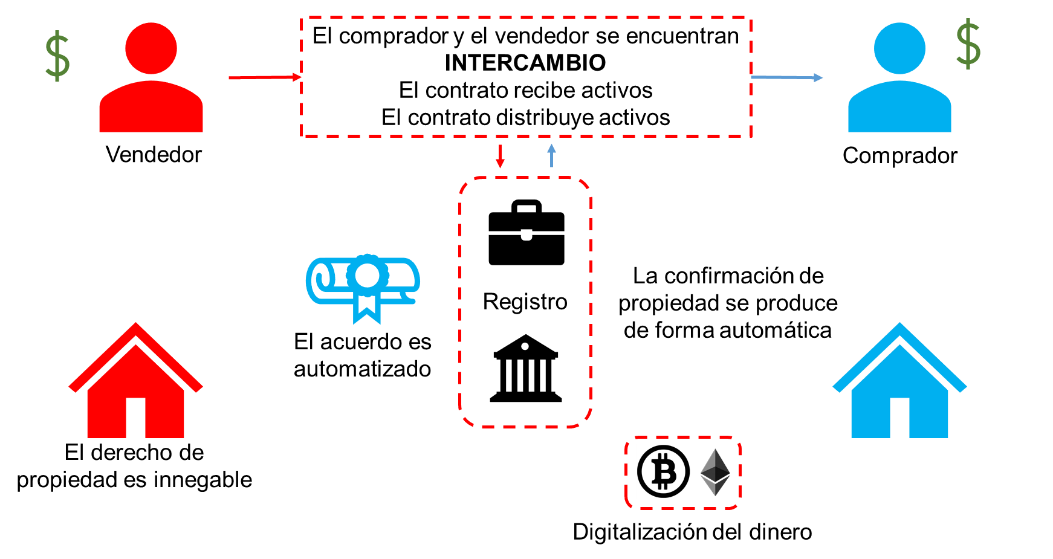
* 1. **Definición, características**

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* (ver figura 1*)*.

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*



Nota. Adaptado de Morant, A. (2018). [Los 5 Beneficios de los SMART CONTRACT. https://www.alfonsomorant.com/5-beneficios-de-los-smart-contract/](file:///C:/Users/jhon/Downloads/CF002/Los%205%20Beneficios%20de%20los%20SMART%20CONTRACT.%20https:/www.alfonsomorant.com/5-beneficios-de-los-smart-contract/)

La inclusión de estos contratos inteligentes, en diferentes aplicaciones y soluciones de comercio electrónico, basadas en *blockchain*, facilitan actividades como (ver figura 2):

**Figura 2**

*Actividades beneficiadas con contratos inteligentes*

Gobierno y administraciones públicas

Automóvil

Compañías de seguros

Banca

Salud

Mercados inmobiliarios

Transporte

Auditoría

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 (ver tabla 1):

**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 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. |

Nota. Adaptado de Shashank (2019).

**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:

Imagen Infográfica

DI\_CF02\_1.1\_Elementos\_de\_ejecución

* 1. **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 (ver figura 3):

**Figura 3**

*Limitaciones de un contrato inteligente*



En el siguiente recurso educativo, se podrá apreciar en detalle estas limitaciones generadas en los contratos inteligentes:

Imagen infográfica

DI\_CF02\_1.2\_Limitaciones\_en\_ contratos\_inteligentes

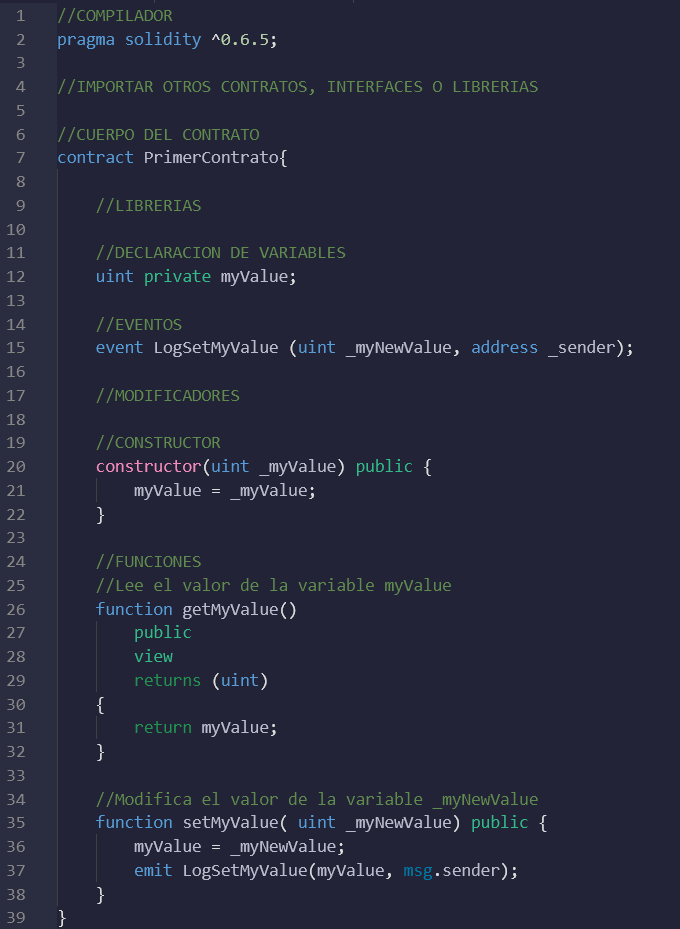
* 1. **Estructura de contrato**

|  |  |
| --- | --- |
|  | 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; en la siguiente imagen, se puede observar de forma general los apartados correspondientes de un contrato inteligente básico (ver figura 4):

**Figura 4**

*Estructura contrato inteligente básico*



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

Sliders

DI\_CF02\_1.3\_Estructura\_del\_contrato

Al igual que las funciones, los modificadores se pueden anular, en el siguiente recurso se puede observar lo correspondiente al constructor:

Pasos

DI\_CF02\_1.3\_Modificadores\_del\_constructor

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.

LLAMADO A LA ACCIÓN

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 en el siguiente enlace: https://docs.soliditylang.org

1. **Plataforma Ethereum**

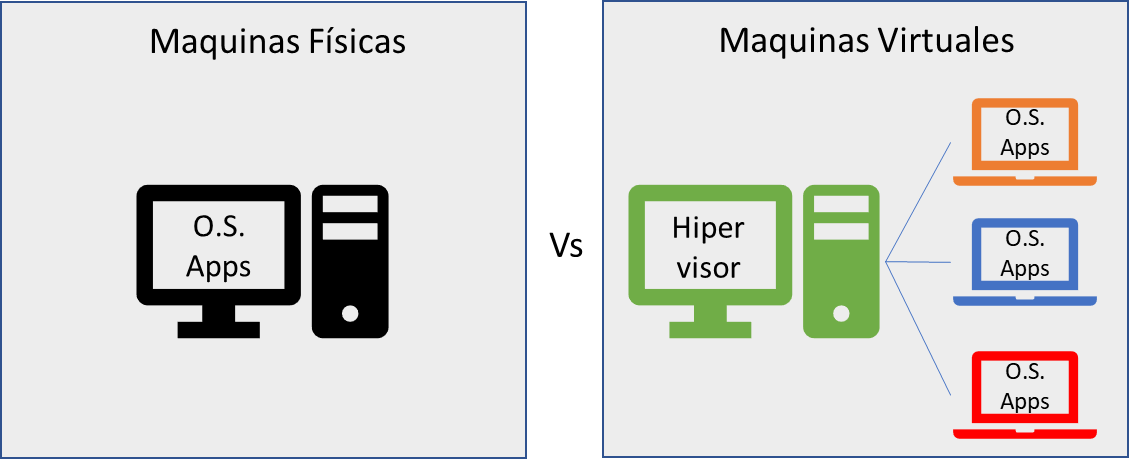
|  |  |
| --- | --- |
| Firma de la plataforma ethereum en el fondo digital video temático de finanzas y criptomonedas animación 3d para tecnología y mercado de valores financiero Foto Premium | La arquitectura de *blockchain* de *Ethereum* es una de las más grandes y utilizadas en proyectos de aplicaciones 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, revise información a continuación. |

* 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, como se puede apreciar en la siguiente figura 5:

**Figura 5**

*Máquinas Físicas vs Máquinas Virtuales*



Nota. SENA (2022).

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 :

Imagen infográfica

DI\_CF02\_2.1\_ Ethereum\_Virtual\_Machine

* 1. **Instalación y Configuración**

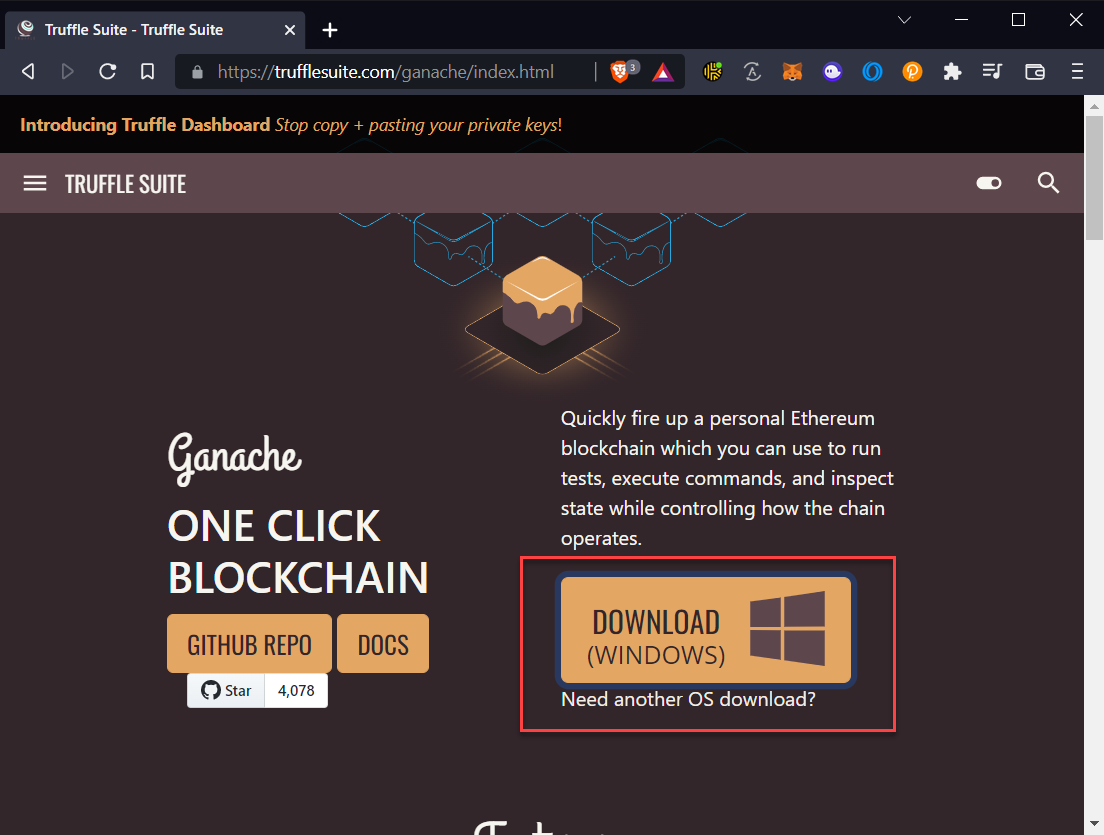
La red de *Ethereum* está abierta a la creación de nuevos nodos, además posibilita agregarse a la red de bloques, si es de nuestro interés, tan solo se debe hacer uso de sus herramientas para realizar un enlace , a continuación, se debe revisar cómo se realiza la creación de un nodo para la red *Ethereum*.

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 (ver figura 6).

**Figura 6.**

*Página oficial para descarga la aplicación Ganache*

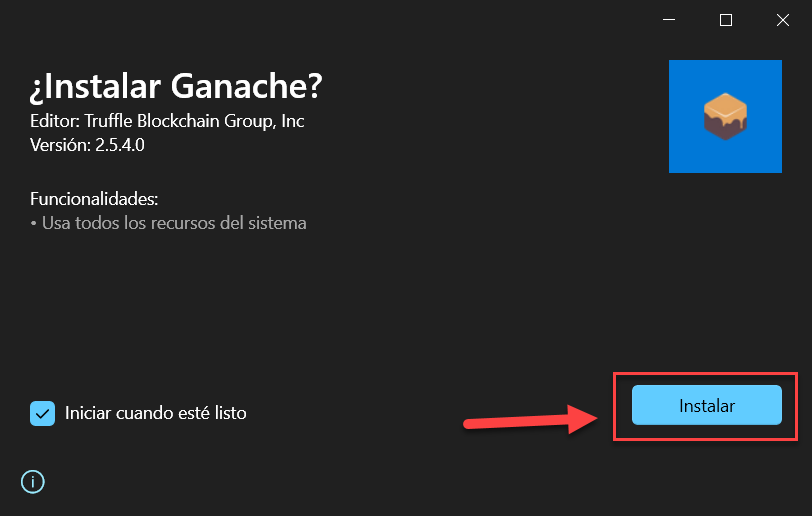


Nota. Tomado de <https://trufflesuite.com/ganache/>

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 (ver figura 7).

**Figura 7**

*Lanzamiento de la instalación de Ganache*



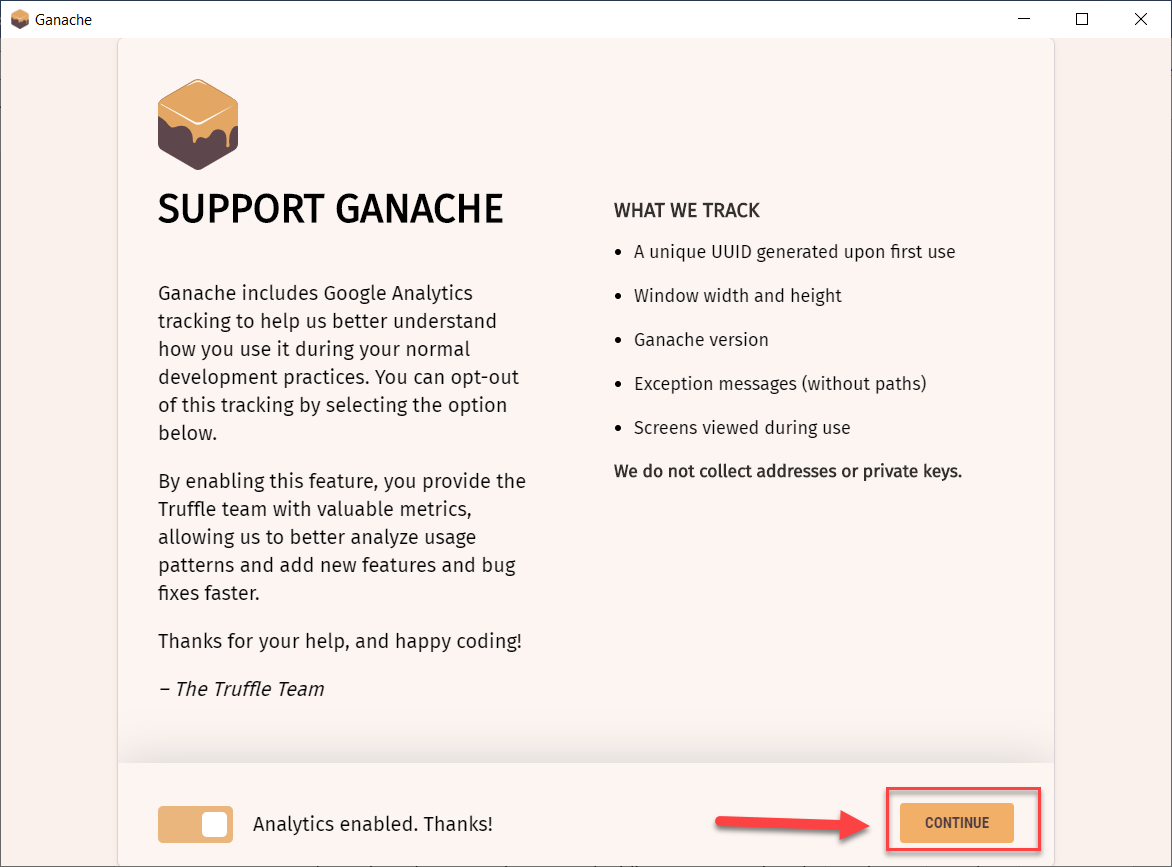
Nota. Tomado de <https://trufflesuite.com/ganache/>

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 8**

*Primer inicio de la aplicación Ganache*



Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

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*”, como se muestra en la siguiente figura 9:

**Figura 9**

*Selección del tipo de workspace a crear*

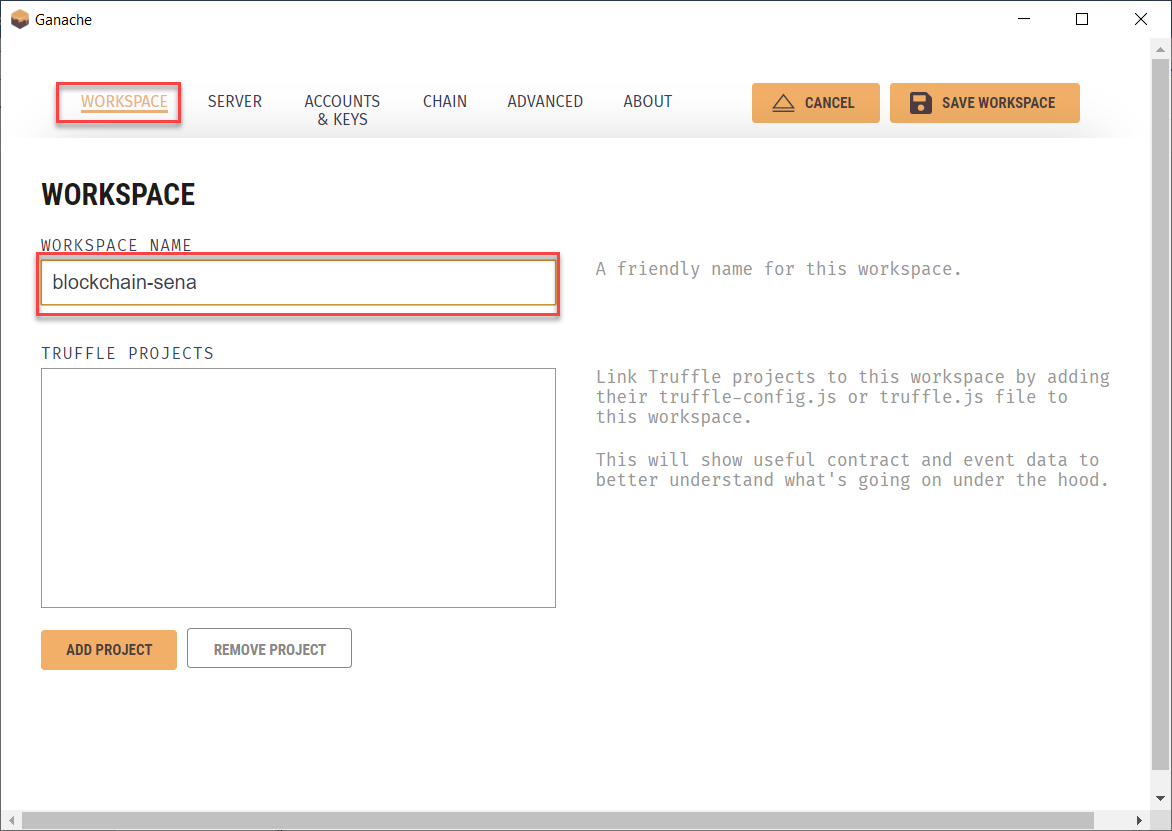


Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

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, como se muestra en la siguiente figura 10:

**Figura 10**

*Ingreso del nombre del espacio de trabajo*



Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

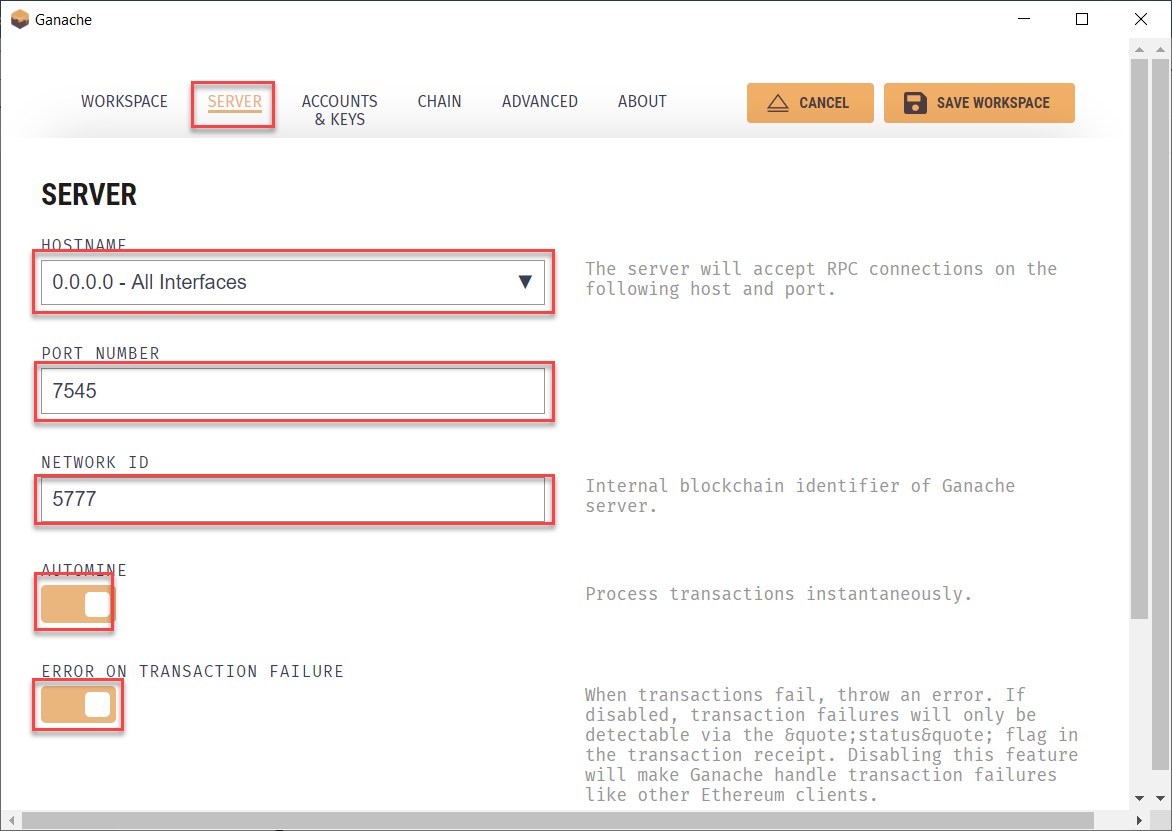
En la pestaña “Server” se puede personalizar el servicio para la conexión a la red blockchain, en la cual es posible configurar:

* **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.

Tal como se presenta en la siguiente figura 10.

**Figura 10**

*Ingreso del nombre del espacio de trabajo*



Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

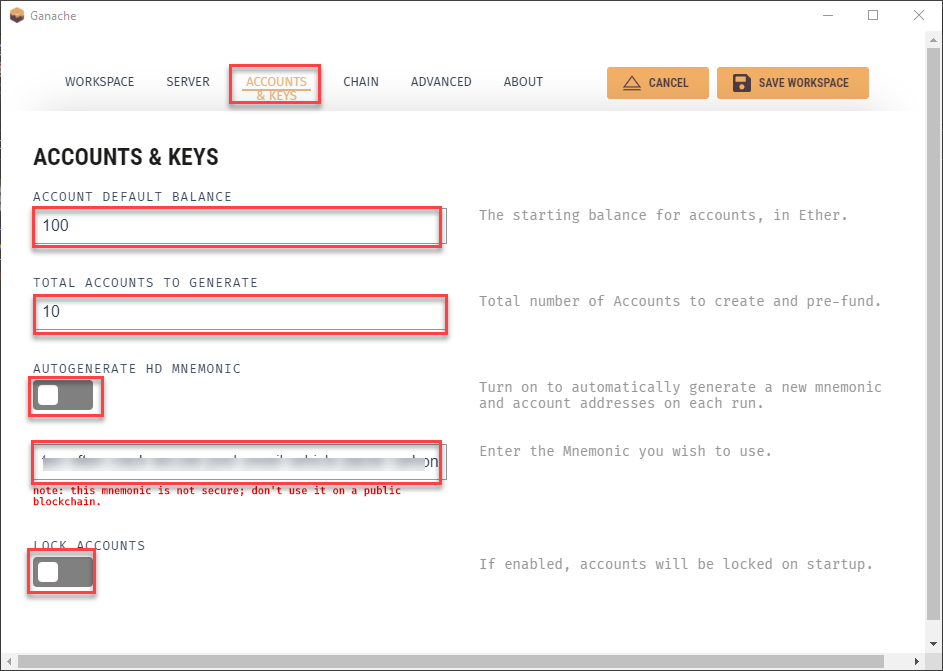
En la pestaña de cuentas y claves, se pueden configurar las cuentas que serán creadas en la red, y se establecen los siguientes parámetros, como se indica a continuación:

* Account Default Balance. Balance inicial en Ether de cada una de las cuentas
* Total, *Accounts to Generate*. Cantidad de cuentas a crear
* *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.
* *Lock Accoounts. C*onfigura si las cuentas iniciarán en estado bloqueado*.*

En la siguiente imagen se muestra las cuentas requeridas para crear en la red, una vez se han tenido en cuenta los criterios indicados:

**Figura 12**

*Gestión de las cuentas para crear en la red*



Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

En la pestaña “*Chain*”, se puede configurar el *hardfork* de la red, es decir sobre la versión base, así como los gastos por comisión *Gas*:

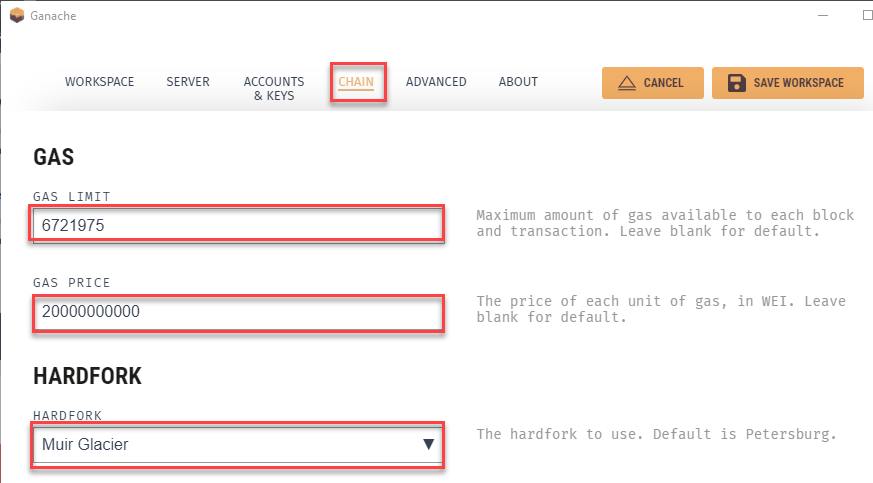
Botones

DI\_CF02\_2.2\_Configurar\_hardfork

Esta configuración, se verá reflejada en el proceso que está llevando a cabo como se muestra en la figura a continuación:

**Figura 13**

*Configuración de la cadena y valor del Gas de comisión*



Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

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

***Output Logs to File***

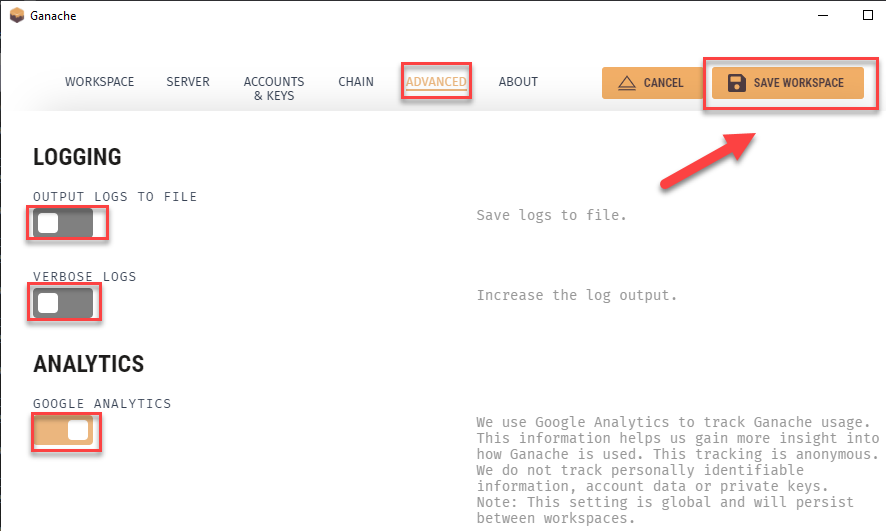
***Verbose Logs***

***Google Analytics***

Finalmente, cuando se haya realizado y revisado la configuración, se puede hacer clic en el botón “*Save Workspace*”, para iniciar el espacio de trabajo. Tal como se muestra en la siguiente figura 14.

**Figura 14**

*Configuración avanzada de la red y finalización de la configuración*

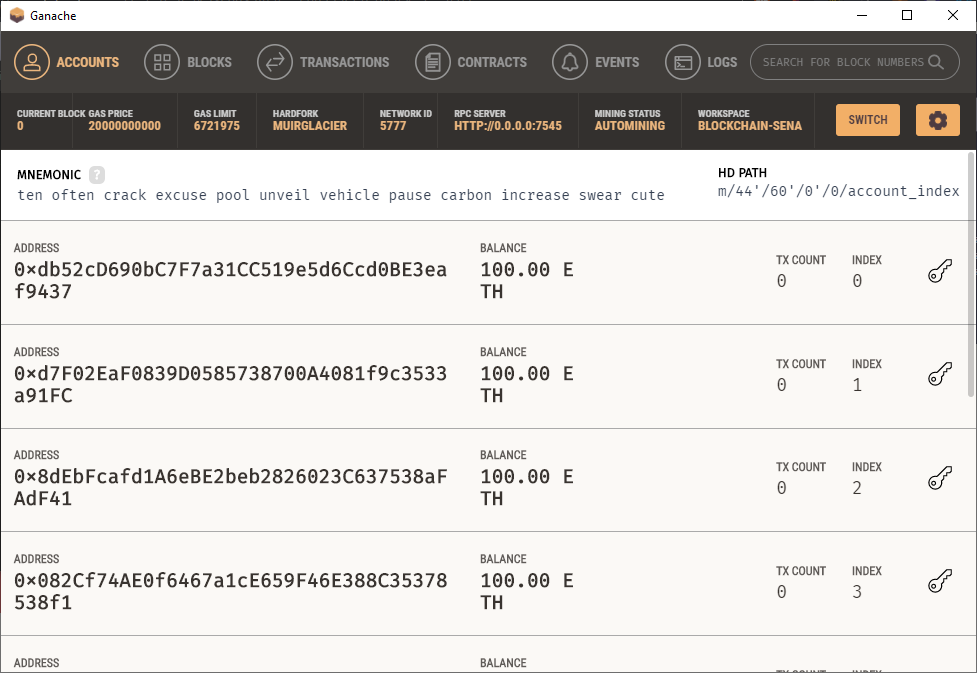


Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

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, como se puede apreciar en la siguiente figura 15.

**Figura 15**

*Pantalla principal del espacio de trabajo configurado*



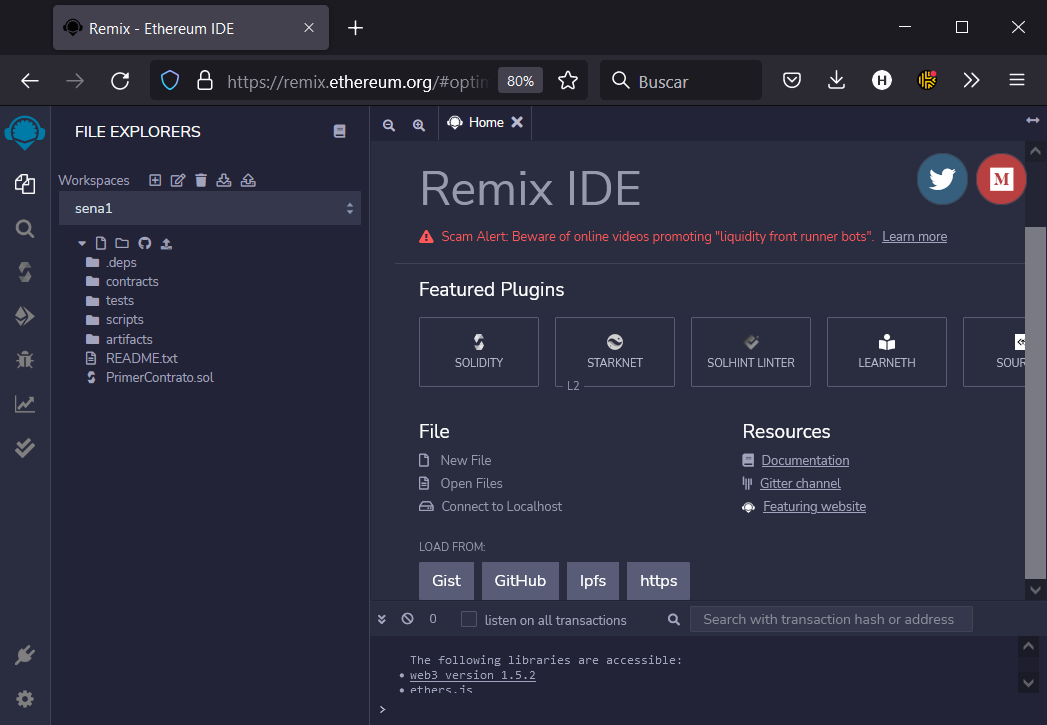
Nota. Adaptado de <https://trufflesuite.com/docs/ganache/quickstart/>

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 desde la url: <https://remix.ethereum.org>

**Figura 16**

*Remix Ethereum IDE*



Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org/)

Este IDE web, 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*.

1. **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.* | Presencia Universitaria |

Las siguientes son algunas características del lenguaje de programación *Solidity*:

Infografía

DI\_CF02\_3\_Solidity

* 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 (ver figura 17):

**Figura 17**

*Expresiones más usadas en Solidity*

**Variables**

**Alcance de las variables**

**Operadores**

**Tipos de valor**

***Array***

A continuación, se encontrará una explicación detallada de cada una de estas expresiones, se invita a revisar el siguiente recurso educativo, en donde se explica lo correspondiente a las **Variables**:

Sliders

DI\_CF02\_3.1\_Expresiones

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

* No utilizar ninguna de las palabras clave reservadas de *Solidity* como nombre de variable.
* 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.
* 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.

Tarjetas flip

DI\_CF02\_3.1\_Alcance\_de\_las\_variables

La siguiente imagen da claridad frente a la definición y uso de alcance de las variables, establecido anteriormente (ver figura 18):

**Figura 18**

*Definición y uso de alcance de las variables*

|  |
| --- |
| Definición y uso de alcance de las variables |
| 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 *ipos* de operadores que se explican a continuación:

Pestañas

DI\_CF02\_3.1\_Operadores

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 2**

*Tipos de datos.*

|  |  |  |
| --- | --- | --- |
| Tipo | Tipo | Valores |
| 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. |

Nota. SENA (2022).

Adicional, 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); |

Nota. SENA (2022).

Por último se encuentra ***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*, se debe 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 se desea declarar una matriz de tamaño dinámico en *Solidity*, se debe 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 se requiere acceder a una posición para leer o escribir un dato, se puede acceder invocando la posición que se desea trabajar, así:

|  |
| --- |
| Lectura de una posición de un array |
| uint salary = balance[2]; |

* 1. **Estructuras de control**

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

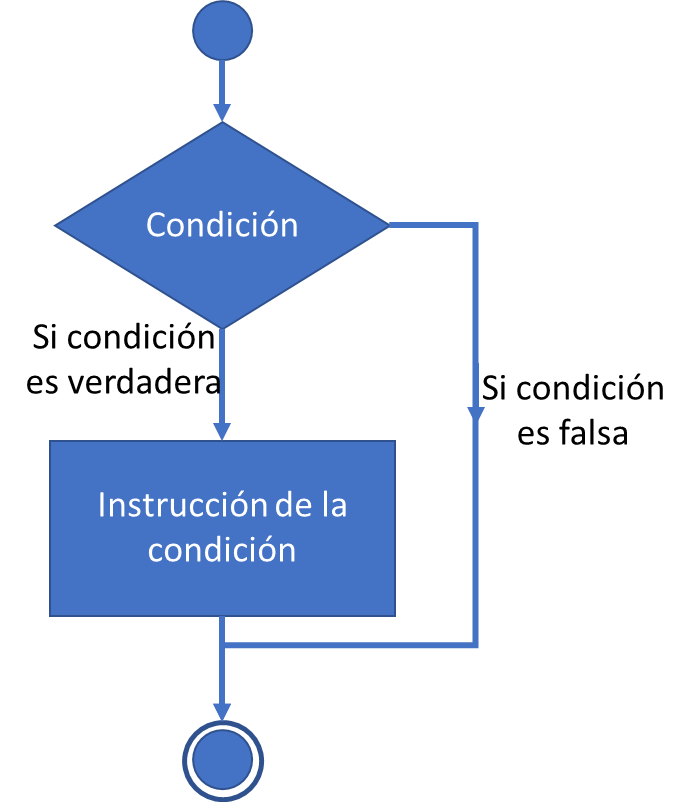
***If-else***: esta estructura de control permite tomar decisiones a partir del cumplimiento de condiciones específicas y con comportamiento definido, presenta las siguientes formas de declararse (ver figura 19):

Contenido colapsable

DI\_CF02\_3.2\_Estructuras\_de\_control

**Figura 19**

*Instrucción if else*



* 1. **Estructuras repetitivas**

Las estructuras repetitivas o también conocidas como bucles, permiten repetir cierto número de veces una acción, mientras se cumpla una condición limitante, dentro de las estructuras repetitivas utilizadas en *S* *olidity* se tienen:

Pestañas

DI-CF02-3.3\_Estructuras\_repetitivas

Con el uso de las estructuras de control se consigue programar la lógica de negocio de un contrato inteligente, de acuerdo con las necesidades y el comportamiento deseado, a continuación, se observa cómo se implementaría en código *Solidity*:

|  |
| --- |
| Uso de estructuras de control de solidity en un contrato inteligente |
| 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);  }  } |

LLAMADO A LA ACCIÓN

Es así como las estructuras repetitivas, pueden ayudar a implementar controles, validaciones y establecer el flujo del proceso y de la información en la construcción de un contrato inteligente, para conocer más sobre la sintaxis del lenguaje el sitio oficial de la documentación de solidity: https://solidity-es.readthedocs.io/es/latest/index.html

* 1. **Funciones**

|  |  |
| --- | --- |
| Funciones en solidity - Crypto Habanero | Una función es un fragmento del 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.    En el siguiente recurso se explican los pasos requeridos para llevar a cabo este proceso: |

Una vez se defina la función, la invoca 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 dicha función.

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

|  |
| --- |
| Ejemplo de definición y uso de una función en solidity |
| 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 se puede implementar una estructura modular de operaciones dentro del contrato inteligente que permita optimizar el código y aprovechar los recursos generados.

LLAMADO A LA ACCIÓN

Si se desea profundizar más en los conceptos revisados anteriormente, se invita a consultar el siguiente enlace, en donde se podrá informar acerca de cómo generar funciones en la documentación oficial del lenguaje de programación:

https://docs.soliditylang.org/en/v0.8.12/contracts.html#functions

1. **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 con esto, desarrollar un contrato inteligente necesitaría muchos recursos dada las pruebas y validaciones que se tienen que hacer antes de desplegar en una red oficial o de producción, para lograrlo, anteriormente se había presentado e instalado la plataforma de desarrollo denominada *Ganache*, sobre la cual se puede hacer las pruebas correspondientes antes de realizar su despliegue en una red oficial.

Con el fin de presentar el proceso de desarrollo y despliegue de un contrato inteligente, se parte de un contrato sencillo con una funcionalidad básica de sumar un valor a una variable declarada internamente, permitiendo consultar el valor final almacenado.

|  |
| --- |
| 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);  }  } |

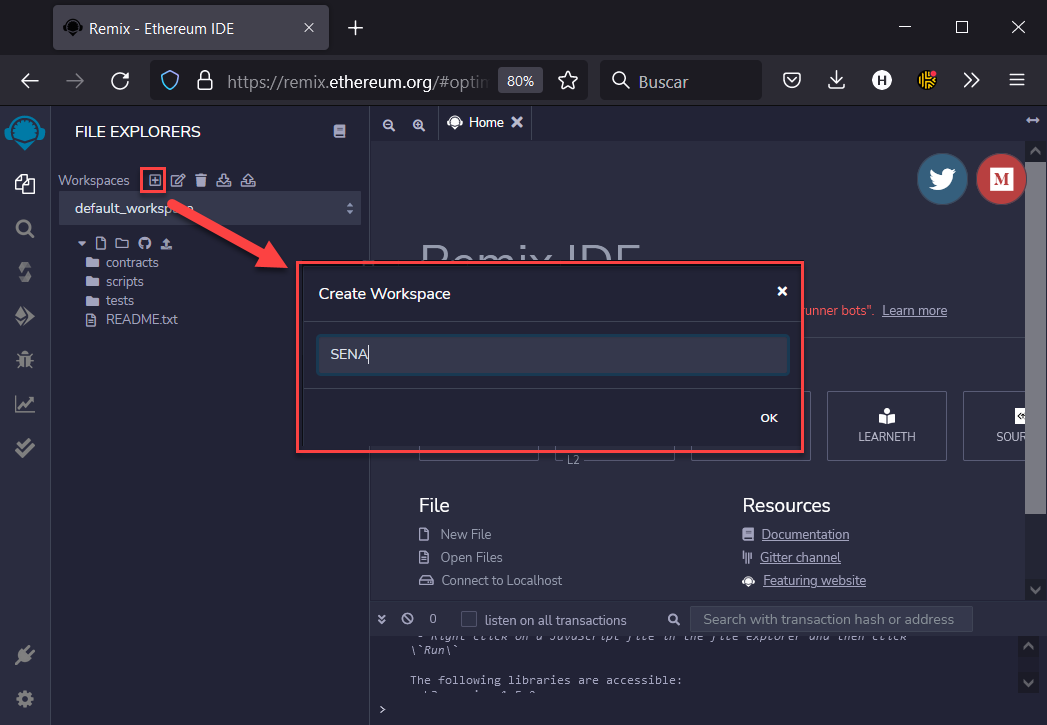
A continuación, se revisa el proceso para desplegarlo en la red de pruebas y desarrollo, así como en una red de producción.

* 1. **Entorno de prueba y desarrollo**

Para realizar la programación y teste del contrato inteligente en una red de pruebas, se puede hacer uso de la plataforma *Ganache* o también del IDE *Remix* de *Ethereum*, para lo cual se debe dirigir a la URL: <https://remix.ethereum.org>, en donde se podrá crear un *workspace* o espacio de trabajo para cada proyecto, por ejemplo, en la siguiente figura 20 se muestra cómo se crea un *workspace* denominado “SENA”.

**Figura 20**

*Creación de Workspace en Remix Ethereum IDE*

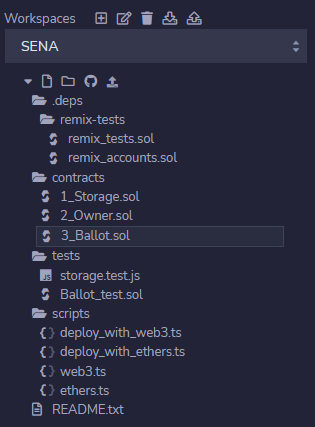


Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

Al crear este *workspace* genera una estructura de archivos base como se muestra en la siguiente figura 21, pero no es necesario seguirla o utilizarla:

**Figura 21**

*Estructura de archivos de un Workspace*

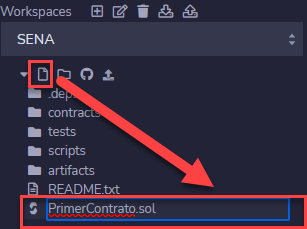


Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

A continuación, se procede a crear el primer archivo del contrato, para el caso del ejemplo será el archivo denominado “PrimerContrato.sol” (ver figura 22).

**Figura 22**

**Creación de un archivo en el *Workspace***

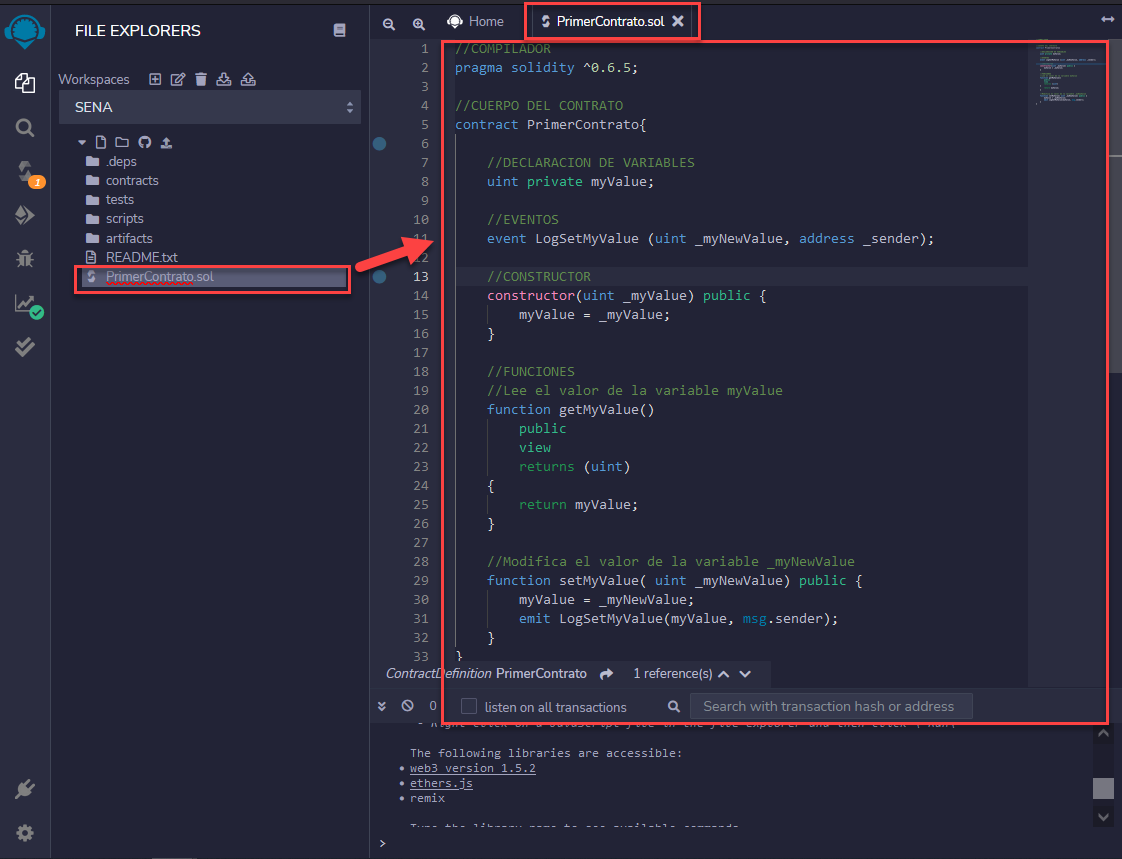


Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

Ahora bien, dentro del archivo que se ha creado, se puede codificar el contrato inteligente, en este caso se va a pegar un contrato de prueba (ver figura 23).

**Figura 23**

*Codificación de un contrato inteligente.*



Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

Una vez se haya codificado el contrato inteligente, se puede ir a la opción de Compilación y seleccionar el entorno de compilación con el cual se va a ejecutar el contrato inteligente, entre la información a configurar se encuentra:

**Auto compile**

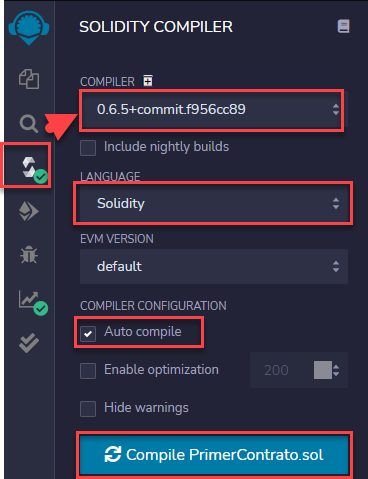
**Lenguaje**

**Versión del compilador**

Existen más opciones, pero no son relevantes para el ejercicio, así que se puede hacer clic en el botón de compilación, para validar el código (ver figura 24).

**Figura 24**

*Opciones de compilación de un contrato inteligente.*



Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

Una vez realizada la compilación, si no se presentan problemas, se puede proceder a realizar el despliegue del contrato inteligente en nuestra red de pruebas, para ello, se procede a hacer clic en el botón de Compilación de *Solidity*, en donde se pueden seleccionar algunos elementos necesarios para realizar el despliegue del contrato inteligente:

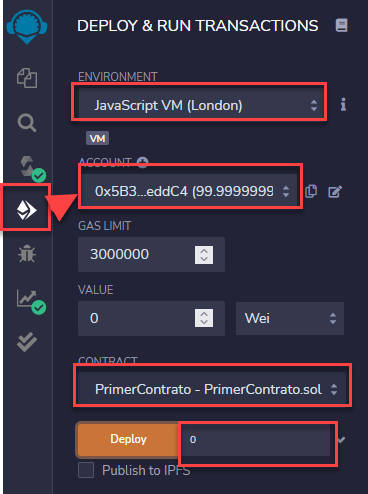
Imagen Infográfica

DI-CF02-4.1\_Despliegue\_contrato\_inteligente

Finalmente se puede hacer clic en el botón “*Deploy*” con el cual se desplega el contrato seleccionado, y será visible dentro de la red (ver figura 25).

**Figura 25**

*Despliegue del contrato sobre la red.*

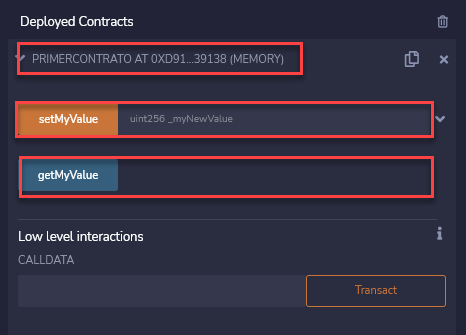


Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

En la siguiente figura 26 se visualiza el contrato inteligente desplegado, con una dirección visible a través de la red, y con los métodos codificados en el ejemplo.

**Figura 2**6

*Consulta del contrato inteligente desplegado*



Nota. Tomado de [Remix - Ethereum IDE](https://remix.ethereum.org)

Como se observa, hacer uso de estas herramientas permite codificar, validar y testear los contratos inteligentes sin gastar recurso real, permitiendo corregir errores antes de llevarlos a una red de producción de *blockchain*.

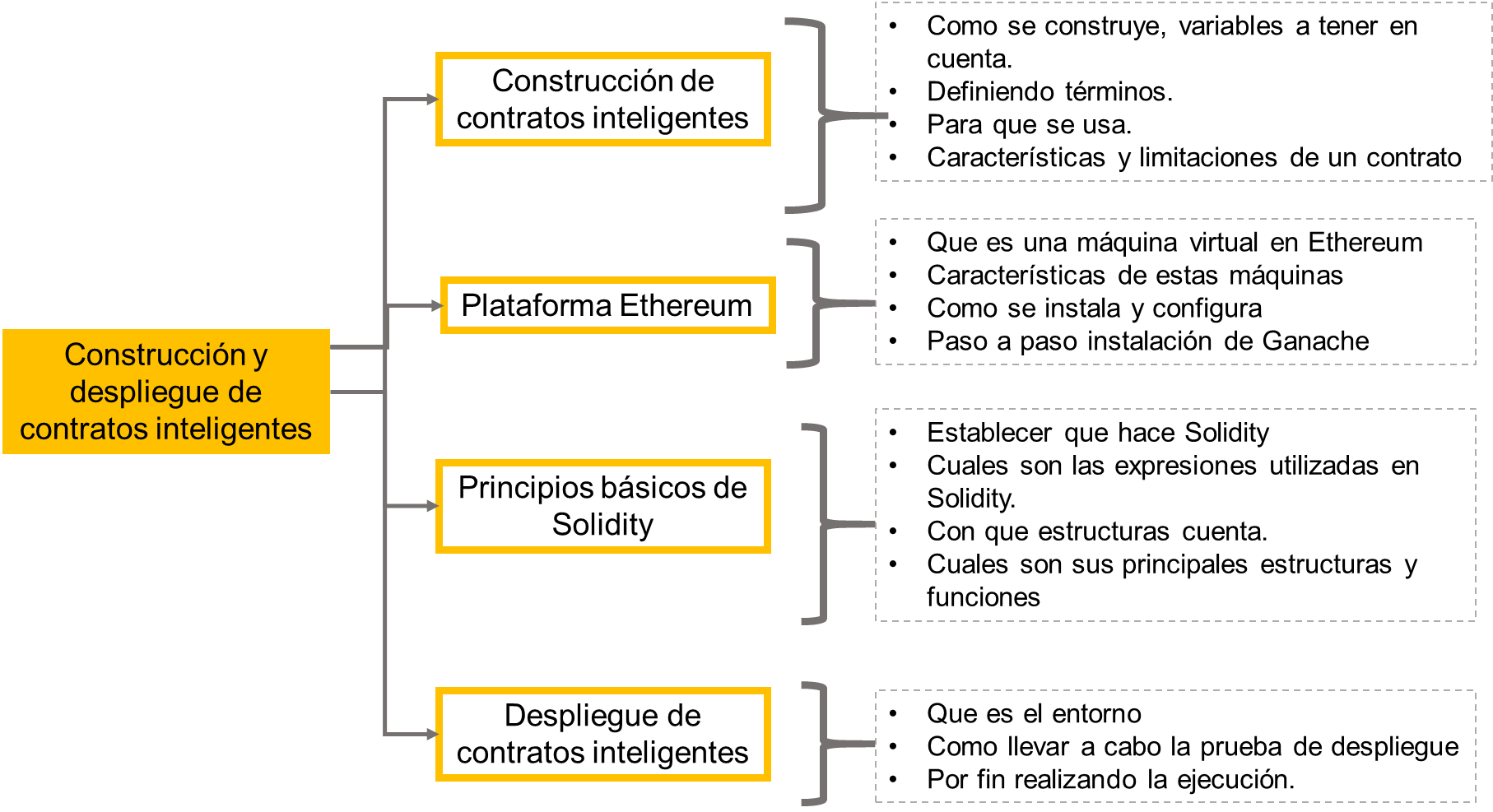
* 1. **Entorno de ejecución**

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 al actualizarse esta tecnología y explorar las nuevas funcionalidades estas soluciones se pueden implementar en los entornos empresariales.

1. **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:

****

1. **ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Relacionemos términos y conceptos |
| Objetivo de la actividad | Identificar conceptos clave para la construcción de los contratos inteligentes, teniendo en cuenta la plataforma *Ethereum* y *Solidity*. |
| Tipo de actividad sugerida | Relación de columnas y conceptos |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | Anexo Actividad didáctica 1. |

1. **MATERIAL COMPLEMENTARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tema** | **Referencia APA del Material** | **Tipo de material**  **(Video, capítulo de libro, artículo, otro)** | **Enlace del Recurso o**  **Archivo del documento o material** |
| Construcción contratos inteligentes | Solitidy. (2022). Introducción a los contratos inteligentes (<https://docs.soliditylang.org> | Página web | <https://docs.soliditylang.org> |
| Construcción contratos inteligentes | Solitidy (2022), Solidity Documentation. <https://solidity-es.readthedocs.io/es> | Manual electrónico | <https://solidity-es.readthedocs.io/es> |
| Instalación y Configuración | Remix IDE. (2022). Welcome to Remix’s documentation. <https://remix-ide.readthedocs.io/en/latest> | Manual electrónico | <https://remix-ide.readthedocs.io/en/latest> |
| Instalación y Configuración | Ganache. (2022). Ganache documentation. <https://trufflesuite.com/docs/ganache> | Manual electrónico | <https://trufflesuite.com/docs/ganache> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| **TÉRMINO** | **SIGNIFICADO** |
| *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. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Arroyo Guardeño, 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>

Morant, A. (2018). [Los 5 Beneficios de los SMART CONTRACT. https://www.alfonsomorant.com/5-beneficios-de-los-smart-contract/](file:///C:/Users/jhon/Downloads/CF002/Los%205%20Beneficios%20de%20los%20SMART%20CONTRACT.%20https:/www.alfonsomorant.com/5-beneficios-de-los-smart-contract/)

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>

Tudela, L (2019) Arquitectura blockchain para la securización de dispositivos iot mediante smart contracts. <http://castor.det.uvigo.es:8080/xmlui/handle/123456789/345>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Nombre** | | **Cargo** | | **Dependencia**  ***(Para el SENA indicar Regional y Centro de Formación)*** | | **Fecha** | |
| **Autor (es)** | | Hernando José Peña Hidalgo | | Experto Temático | | Regional Cauca, Centro de teleinformática y producción industrial. | | Mayo 2022 | |
| María Inés Machado López | | Diseñador instruccional | | Regional Norte de Santander. Centro de la Industria, la Empresa y los Servicios. | | Mayo 2022 | |
| Alix Cecilia Chinchilla Rueda | | Asesor Metodológico | | Regional Distrito Capital. Centro de Diseño y Metrología. | | Mayo 2022 | |
| Rafael Neftalí Lizcano Reyes | | Responsable del equipo de desarrollo curricular | | Regional Santander. Centro Industrial del Diseño y la Manufactura. | | Mayo 2022 | |
|  | | Jhon Jairo Rodríguez Pérez | | Corrector de estilo | | Regional Distrito Capital - Centro de Diseño y Metrología. | | Junio 2022 | |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |