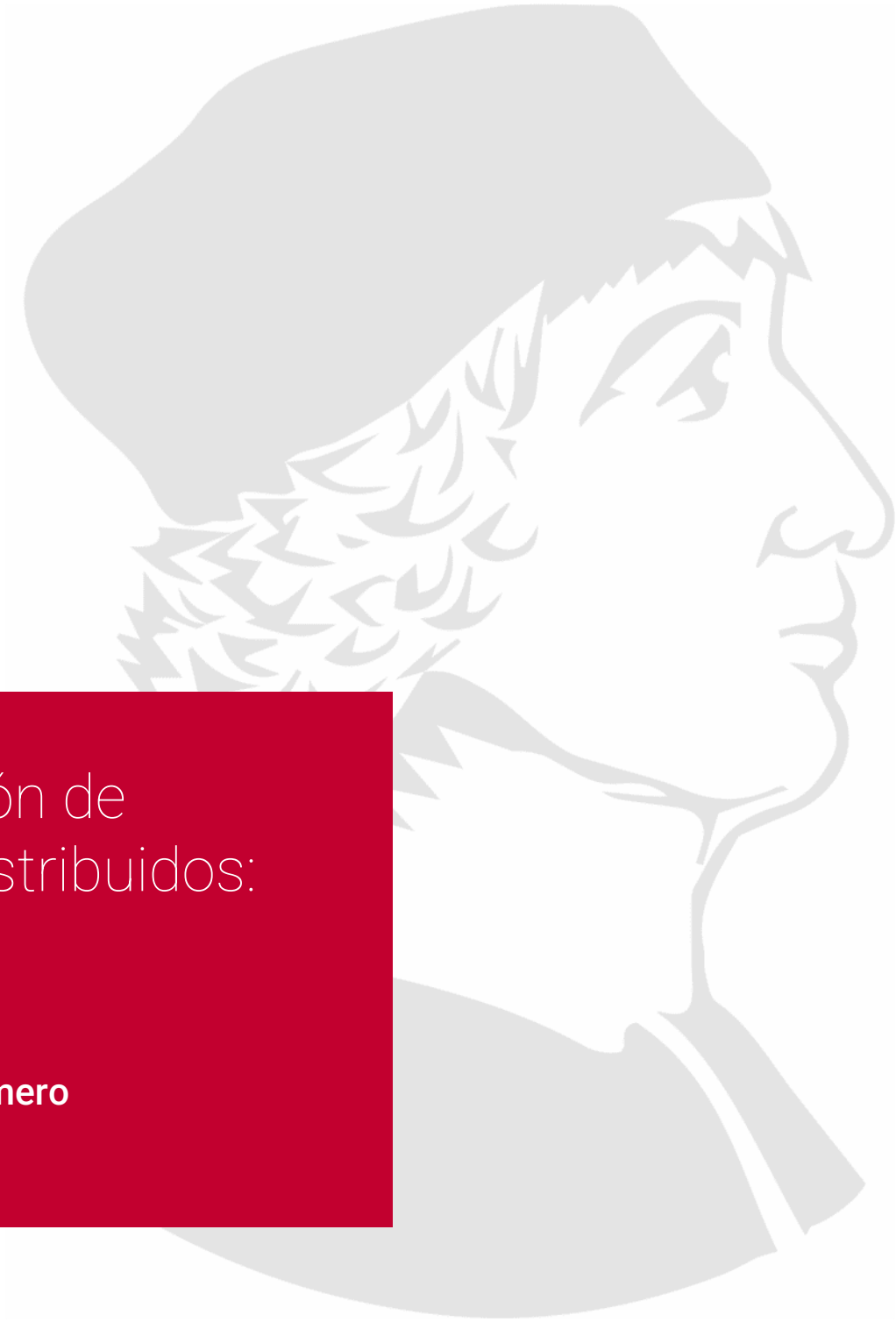


Programación de Sistemas Distribuidos: Práctica 4

Chenlei Ye
Sara Marcos
Jose Francisco Romero

UNIVERSIDAD NEBRIJA

Contenido

Introducción

En esta práctica, aprenderemos los conceptos básicos de blockchain y cómo se puede implementar una blockchain simple utilizando el lenguaje de programación Ruby. Hablaremos de las características clave de blockchain, incluyendo su estructura de datos, criptografía y consenso distribuido, y luego describiremos cómo se pueden aplicar estos conceptos en Ruby para crear una implementación básica de una cadena de bloques.

Blockchain es una tecnología innovadora que ha ganado gran popularidad en la última década debido a su capacidad para proporcionar un registro descentralizado y seguro de transacciones digitales. A menudo se asocia con criptomonedas como Bitcoin, pero sus aplicaciones van más allá de las finanzas, incluyendo áreas como la cadena de suministro, el registro de tierras, la votación y más. En esencia, una blockchain es una base de datos distribuida y en constante crecimiento que utiliza criptografía y consenso entre los participantes para garantizar la integridad de los datos.

Ruby es un lenguaje de programación dinámico, de alto nivel y orientado a objetos que se ha vuelto popular en la comunidad de desarrollo debido a su sintaxis clara, fácil de leer y su enfoque en la simplicidad y la productividad. Debido a su flexibilidad y características aceptables para el desarrollador, Ruby es una excelente opción para implementar una blockchain simple con fines educativos.

Ejercicios

1. Del siguiente repositorio https://github.com/apradillap/simple-blockchain-in-ruby

a) Ejecuta la blockchain.

En primer lugar, se debe instalar ruby. En nuestro caso, vamos a realizar la instalación de Ruby y modificación de los códigos en Ubuntu. Mediante el siguiente comando se haría la instalación:
"sudo apt-get install ruby-full"

```
sara@sara-vivobook:~$ sudo apt-get install ruby-full
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no
son necesarios.
  libflashrom1 libftdi1-2 libjs-highlight.js libllvm13
Utilice «sudo apt autoremove» para eliminarlos.
Se instalarán los siguientes paquetes adicionales:
  fonts-lato libgmp-dev libgmpxx4ldbl libjs-jquery libruby3.0 rake ri ruby
  ruby-dev ruby-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0
  ruby3.0-dev ruby3.0-doc rubygems-integration
Paquetes sugeridos:
  gmp-doc libgmp10-doc libmpfr-dev bundler
Se instalarán los siguientes paquetes NUEVOS:
  fonts-lato libgmp-dev libgmpxx4ldbl libjs-jquery libruby3.0 rake ri ruby
  ruby-dev ruby-full ruby-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc
  ruby3.0 ruby3.0-dev ruby3.0-doc rubygems-integration
0 actualizados, 18 nuevos se instalarán, 0 para eliminar y 8 no actualizados.
Se necesita descargar 11,0 MB de archivos.
Se utilizarán 58,8 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
```

Una vez se tiene esto, se ejecuta la blockchain mediante el comando: "ruby blockchain.rb", y se podrá empezar a interactuar con ella.

```
sara@sara-vivobook:~/Documents/web-front/practica4sd$ ruby blockchain.rb
==========================

Welcome to Simple Blockchain In Ruby !
```

En el siguiente apartado, se analiza en detalle su funcionamiento

b) Analiza cómo funciona y añade capturas de pantalla para mostrar evidencias de la compresión de la ejecución.

El código consta de 3 archivos: **block.rb, blockchain.rb** y **transaction.rb.** A continuación se analiza cada uno de ellos:

block.rb:

La clase Block es la representación de un bloque en la cadena de bloques de la blockchain. Cada bloque contiene información sobre transacciones y tiene una referencia al bloque anterior en la cadena.

A continuación, se explican los atributos y métodos de la clase Block:

- attr_reader :index, :timestamp, :transactions, :transactions_count, :previous_hash, :nonce, :hash: Establece los atributos de solo lectura para la instancia de bloque. Esto significa que no pueden ser modificados después de la creación del bloque.
- initialize(index, transactions, previous_hash): Este es el constructor de la clase Block. Recibe como argumentos el índice del bloque, las transacciones que contiene y el hash del bloque anterior en la cadena.
- compute_hash_with_proof_of_work(difficulty="00"): Este método utiliza el algoritmo de Prueba de Trabajo (PoW) para calcular el nonce y el hash del bloque. El parámetro difficulty especifica la dificultad de la PoW. El método itera sobre diferentes valores de nonce hasta encontrar un hash que comience con difficulty.
- calc_hash_with_nonce(nonce=0): Este método calcula el hash del bloque utilizando el nonce dado y los atributos del bloque.
- self.first(*transactions): Este método estático crea el primer bloque de la cadena, también conocido como bloque génesis. Recibe como argumento una lista de transacciones que se agregarán al bloque.
- self.next(previous, transactions): Este método estático crea un nuevo bloque en la cadena. Recibe como argumentos el bloque anterior en la cadena y una lista de transacciones que se agregarán al bloque.

Así, la clase Block define la estructura y la lógica de los bloques de la cadena de bloques.

blockchain.rb:

El archivo blockchain.rb es el programa principal que crea y ejecuta la blockchain. Primero, el programa requiere los módulos de Ruby 'digest' y 'pp', así como los archivos 'block.rb' y 'transaction.rb'. Luego define una constante LEDGER vacía.

- La función create_first_block se encarga de crear el primer bloque de la blockchain, utilizando el método Block.first de la clase Block. Este bloque contiene dos transacciones de ejemplo.
- La función add_block agrega nuevos bloques a la blockchain utilizando el método Block.next de la clase Block. Cada vez que se llama a esta función, se le solicita al usuario que proporcione información sobre una o varias transacciones. Estas transacciones se pasan como argumentos al método Block.next y se crean nuevos bloques.
- La función launcher es el punto de entrada del programa. Simplemente muestra un mensaje de bienvenida y llama a create_first_block para comenzar a crear la blockchain.

El archivo blockchain.rb es el programa principal que utiliza los módulos y archivos necesarios para crear y ejecutar la blockchain. Define las funciones necesarias para agregar bloques a la cadena, utilizando los métodos de la clase Block, y el bucle de repetición que permite al usuario seguir agregando bloques de forma indefinida. También crea y utiliza la constante LEDGER para almacenar la cadena de bloques creada.

transaction.rb:

Este es un método que se encarga de recopilar información de transacciones a través de la entrada del usuario. Al llamar al método get_transactions_data, se crea un array vacío transactions_block y se establece un hash en blanco blank_transaction.

El método entra en un bucle do-while en el que el usuario es solicitado a ingresar información de transacciones una por una. La información solicitada incluye el nombre del remitente from, el nombre del destinatario to, el producto enviado what y la cantidad enviada qty.

Después de que se recopila la información de la transacción, se crea un hash con los valores ingresados y se agrega al transactions_block array.

Luego se le pregunta al usuario si desea agregar otra transacción a este bloque de transacciones. Si el usuario elige "y" se ejecuta nuevamente el método get_transactions_data, y si el usuario elige "n", el array transactions_block se devuelve y se sale del bucle.
La red de blockchain implementada en el código que se proporciona es bastante simple y consiste en una única cadena de bloques en la que cada bloque contiene una serie de transacciones.

A continuación, se muestra una captura de su funcionamiento:

```
==========================

Welcome to Simple Blockchain In Ruby !

This program was created by Anthony Amar for and educationnal purpose

Wait for the genesis (the first block of the blockchain)

.........

==========================
#<Block:0x000055b040fb4500
 @hash="0052e289c41bb951bd9ba7086669ce9c4c55201e9b5368e9f4c1486419e28e9e",
 @index=0,
 @nonce=224,
 @previous_hash="0",
 @timestamp=2023-05-02 11:04:29.351270282 +0200,
 @transactions=
  [{:from=>"Dutchgrown",
    :to=>"Vincent",
    :what=>"Tulip Bloemendaal Sunset",
    :qty=>10},
   {:from=>"Keukenhof", :to=>"Anne", :what=>"Tulip Semper Augustus", :qty=>7}],
 @transactions_count=2>
"=========================="

Enter your name for the new transaction
Transaction1

What do you want to send ?
hello!

How much quantity ?
3

Who do you want to send it to ?
Receiver

Do you want to make another transaction for this block ? (Y/n)
n
```

El programa comienza por crear el primer bloque con algunas transacciones de prueba y luego solicita al usuario que ingrese las transacciones para el siguiente bloque. Las transacciones se ingresan como un conjunto de datos que incluyen la dirección del remitente, la dirección del destinatario, la cantidad y la descripción de la transacción.

El programa utiliza la función get_transactions_data en el archivo transaction.rb para recopilar la información de las transacciones del usuario y almacenarlas en una matriz. Si el usuario desea agregar más transacciones, se llama de nuevo a la función, y se agregan más transacciones a la matriz. Si el usuario no desea agregar más transacciones, la matriz de transacciones se devuelve al programa principal.

Cada bloque en la cadena tiene un índice, un hash de bloque anterior y un hash de bloque actual generado utilizando un algoritmo de prueba de trabajo, que en este caso es una función hash SHA-256. El algoritmo de prueba de trabajo se utiliza para asegurar que se ha dedicado un esfuerzo significativo a la creación del bloque, lo que hace que sea más difícil para los atacantes modificar los bloques de la cadena.

Finalmente, el programa guarda cada bloque en una matriz llamada LEDGER, lo que permite guardar un registro de todas las transacciones que se han realizado en la cadena de bloques. En resumen, esta red simple de blockchain es un ejemplo básico de cómo funciona una cadena de bloques, donde se pueden agregar bloques de transacciones y se verifica la integridad de la cadena utilizando la función hash criptográfica y el algoritmo de prueba de trabajo.

2. ¿Qué es el bloque génesis?

El bloque génesis es el primer bloque de una cadena de bloques o blockchain. Este bloque es único, ya que no tiene ningún bloque anterior al que hacer referencia y por lo tanto, es el punto de partida de la cadena.

El bloque génesis es creado por el creador de la cadena de bloques, y contiene información esencial sobre la cadena, como la configuración inicial, los parámetros de consenso, y la identidad del creador. Además, el bloque génesis también contiene la primera transacción que se realiza en la cadena, que generalmente es la recompensa por la minería del bloque génesis.

Es importante destacar que el bloque génesis no puede ser modificado una vez que se ha creado, ya que cualquier cambio en el bloque génesis invalidaría toda la cadena de bloques que lo sigue. Por lo tanto, es crucial que la información del bloque génesis sea precisa y completa desde el principio.

El proceso de crear un bloque génesis es diferente para cada cadena de bloques, y depende del protocolo de consenso que se utilice. Algunas cadenas de bloques, como Bitcoin, utilizan un proceso de minería basado en prueba de trabajo para crear el bloque génesis, mientras que otras, como Ethereum, utilizan un proceso de generación de bloques basado en prueba de participación.

Así, se puede decir que el bloque génesis es el primer bloque de una cadena de bloques, y contiene información esencial sobre la cadena, como la configuración inicial, los parámetros de consenso, y la identidad del creador. Es fundamental que la información del bloque génesis sea precisa y completa desde el principio, ya que cualquier cambio en el bloque génesis invalidaría toda la cadena de bloques que lo sigue.

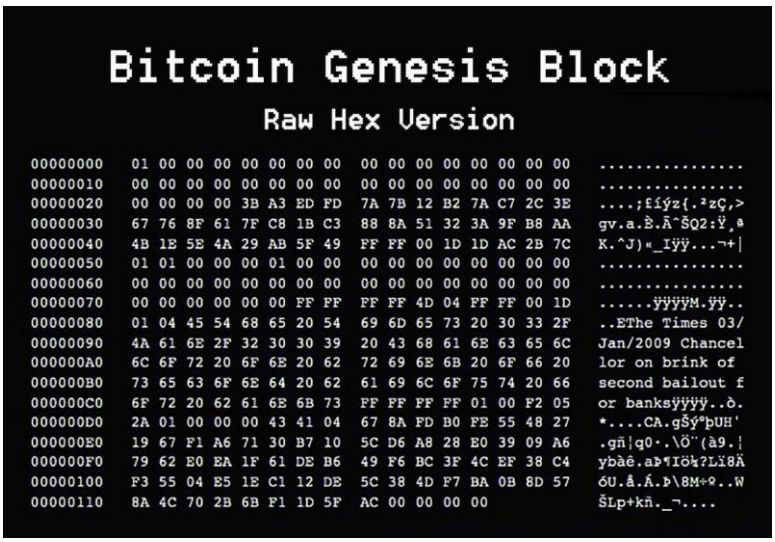


Figura 1.1: ejemplo de bloque de génesis

3. ¿Qué elementos tiene cada bloque?

Cada bloque de una cadena de bloques contiene diferentes elementos, pero hay algunos que son comunes en la mayoría de las implementaciones de blockchain. Elementos más comunes en un bloque de cadena de bloques:

1. Identificador del bloque: es un número único que identifica al bloque dentro de la cadena.
2. Hash del bloque anterior: es el hash del bloque inmediatamente anterior en la cadena. Esta es la forma en que los bloques están enlazados entre sí para formar una cadena.
3. Timestamp o sello temporal: es la marca de tiempo que indica cuándo se creó el bloque.
4. Datos: son los datos específicos del bloque. Una lista de transacciones que se registran en el bloque.
5. Una prueba de trabajo (nonce): un número aleatorio que se agrega al cálculo del hash para garantizar que el proceso de minería sea difícil y que se requiera una cantidad significativa de energía computacional para crear nuevos bloques en la cadena.
6. Hash del bloque: es el hash generado a partir de todos los elementos anteriores del bloque. Es único para cada bloque y actúa como una huella digital que garantiza la integridad del bloque y su posición en la cadena.

Además de estos elementos, algunos bloques pueden incluir otros datos o campos específicos de la cadena de bloques, como la dificultad de minería, el tamaño del bloque, la recompensa de minería, entre otros.

Cada elemento en un bloque es importante para garantizar la integridad y seguridad de la cadena de bloques. Por ejemplo, el hash del bloque anterior garantiza que ningún bloque anterior pueda ser modificado sin afectar la integridad de la cadena. El timestamp asegura que los bloques se creen en un orden cronológico preciso, y la inclusión de un nonce en el proceso de minería garantiza que los bloques sean creados de manera aleatoria y no puedan ser predecibles.

4. ¿Qué hace el método def compute_hash_with_proof_of_work? ¿Qué significa Proof of work?

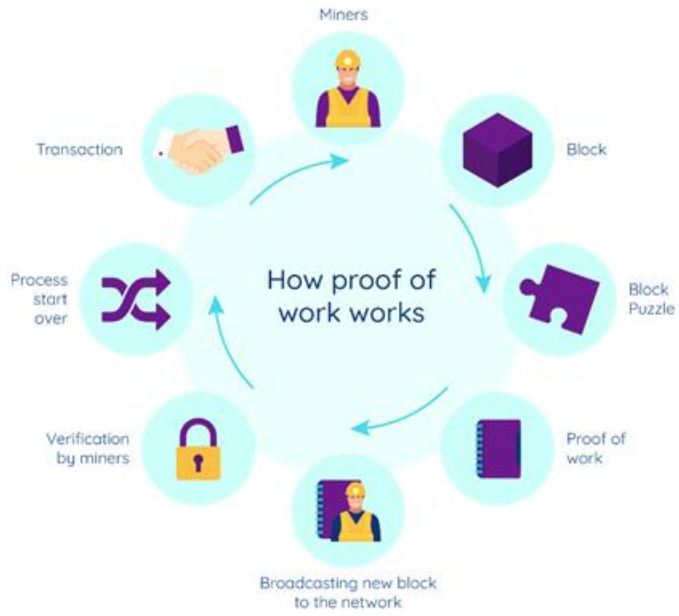
El método compute_hash_with_proof_of_work es responsable de calcular y encontrar el hash del bloque actual junto con una prueba de trabajo (Proof of Work). Este método implementa un algoritmo de prueba de trabajo básico para encontrar un número aleatorio llamado nonce que, cuando se concatena con el contenido del bloque actual y se aplica la función hash SHA256, produce un hash que cumple con un requisito específico, conocido como dificultad.

La dificultad se refiere a la cantidad de ceros requeridos al comienzo del hash calculado. Cuanto mayor sea la dificultad, más difícil será encontrar un hash válido y, por lo tanto, más trabajo deberá hacer el algoritmo de prueba de trabajo. Este proceso se repite iterativamente hasta que se encuentra un hash que cumple con los requisitos de dificultad.

Por otra parte, Proof of Work es un protocolo de consenso utilizado en blockchain que se utiliza para confirmar transacciones y agregar nuevos bloques a la cadena de bloques. La idea es que los mineros en la red realicen trabajo computacional para validar las transacciones y producir nuevos bloques.

Al hacer que este trabajo sea difícil y costoso, se hace más seguro el proceso de validación. El objetivo es garantizar la seguridad y la integridad de la cadena de bloques al evitar el doble gasto y la manipulación de la cadena.

Es comúnmente utilizada en sistemas de blockchain como Bitcoin y Ethereum.



1.2: Funcionamiento del Proof of Work

5. Haz un fork de la práctica. ¿Te atreves a realizar algún cambio? Se valorarán más las propuestas ingeniosas.

Primero hay que hacer el fork del repositorio.

1. Accedemos al repositorio de GitHub:https://github.com/apradillap/simple-blockchain-in-ruby
2. Hacemos clic en "Fork" en la esquina superior derecha de la página. Esto creará una copia del repositorio en nuestra cuenta de GitHub.
3. Una vez completado el proceso de fork, buscamos el repositorio "simple-blockchain-in-ruby".
4. Hacemos clic en clonar y copiamos la URL del repositorio en el portapapeles.
5. Clonamos el repositorio donde queramos de nuestro ordenador con la url.

A partir de aquí hemos realizado algunos cambios que incluyen la incorporación de un menú para permitir al usuario visualizar y manejar de manera diferente las acciones que puede realizar. También se ha implementado una gestión de errores para manejar situaciones en las que algo no funciona según lo esperado.

Se ha implementado la posibilidad de ver los bloques y las transacciones en consola. Además, se ha añadido una funcionalidad para guardar las transacciones en un archivo de texto, lo que permite al usuario ver el historial de transacciones. La clase Block no ha sido modificada.

Estos cambios proporcionan al usuario una experiencia más completa y amigable al interactuar con la cadena de bloques en Ruby.

Menú:

```
\Practica4\simple-blockchain-in-ruby> ruby blockchain.rb
#<Block:0x00000234d4ceb598
 @hash="0052d0c7666ab73fb6479be8336c2c1dcb9279600c023332ed4ba4dcea266231",
 @index=0,
 @nonce=232,
 @previous_hash="0",
 @timestamp=2023-05-07 13:58:10.4692614 +0200,
 @transactions=
  [{:from=>"Dutchgrown", :to=>"Vincent", :what=>"Tulip Bloemendaal Sunset", :qty=>10},
   {:from=>"Keukenhof", :to=>"Anne", :what=>"Tulip Semper Augustus", :qty=>7}],
 @transactions_count=2>
"============================"
Select an option:
1. Add a transaction
2. List all blocks
3. List all transactions
4. Save transactions to file
5. Exit
```

El usuario debe ingresar un número para elegir una opción del menú. Después de que el usuario elige una opción, el código usa una declaración de "case" para realizar una acción correspondiente a la opción seleccionada.

Si el usuario ingresa un número que no coincide con ninguna de las opciones, se llama a la función "invalid_choice" para manejar la entrada inválida.

Después de que se realiza una acción, el código llama a la función "menu" nuevamente para presentar al usuario el menú de opciones. Esto permite al usuario interactuar con la cadena de bloques de manera continua hasta que decida salir del programa.

```ruby
def menu
    puts "Select an option:"
    puts "1. Add a transaction"
    puts "2. List all blocks"
    puts "3. List all transactions"
    puts "4. Save transactions to file"
    puts "5. Exit"

    choice = gets.chomp.to_i

    case choice
    when 1
      add_transaction
    when 2
      list_all_blocks
    when 3
      list_all_transactions
    when 4
      save_transactions_to_file
    when 5
      exit_program
    else
      invalid_choice
    end

    menu
  end
```

Además, también para que funcione este menú hemos añadido algunas funciones como: "list_all_blocks", "add_transaction", "list_all_transactions", "save_transactions_to_file", "exit_program", "invalid_choice".

Estas 3 funciones están definidas en la clase blockchain:

```ruby
def list_all_blocks
  LEDGER.each do |block|
    pp block
    puts "============================"
  end
end

def exit_program
  exit
end

def invalid_choice
  puts "Invalid choice! Please try again."
end
```

En primer lugar, el método list_all_blocks itera a través de la matriz LEDGER que contiene todos los bloques creados en la cadena de bloques. Dentro del bloque do, se muestra el contenido del bloque en la pantalla usando el método pp (pretty printer) y se imprime una línea para separar los bloques visualmente.

El método exit_program simplemente llama a la función exit para terminar la ejecución del programa.

Por último, el método invalid_choice imprime un mensaje en la pantalla cuando el usuario ha ingresado una opción de menú no válida.

```ruby
def add_transaction
    latest_block = LEDGER.last
    latest_block_hash = latest_block.hash
    new_transactions = get_transactions_data
    new_block = Block.next(latest_block, new_transactions)
    LEDGER << new_block
    puts "New block created with #{new_block.transactions.size} transactions."
  end


def list_all_transactions
  LEDGER.each do |block|
    block.transactions.each do |transaction|
      puts "From: #{transaction[:from]}, To: #{transaction[:to]}, What: #{transaction[:what]}, Qty: #{transaction[:qty]}"
    end
  end
end

def save_transactions_to_file
    puts "Enter file name to save transactions to (without .txt extension): "
    filename = gets.chomp
    File.open("#{filename}.txt", "w") do |file|
      LEDGER.each do |block|
        block.transactions.each do |transaction|
          file.puts "From: #{transaction[:from]}, To: #{transaction[:to]}, What: #{transaction[:what]}, Qty: #{transaction[:qty]}"
        end
      end
    end
    puts "Transactions saved to #{filename}.txt"
  end
```

La función add_transaction toma la última entrada de la cadena, calcula su hash y agrega nuevas transacciones a un nuevo bloque utilizando el método next de la clase Block. El nuevo bloque se agrega a la cadena, y se imprime un mensaje que indica cuántas transacciones se agregaron al nuevo bloque.

La función list_all_transactions recorre todos los bloques de la cadena e imprime la información de cada transacción en cada bloque.

La función save_transactions_to_file solicita al usuario un nombre de archivo, recorre todos los bloques de la cadena y escribe la información de cada transacción en el archivo especificado. Cada transacción se escribe en una nueva línea. Al final, se imprime un mensaje que indica que las transacciones se han guardado en el archivo especificado.

Estas últimas 3 funciones están definidas en transaction.rb, donde también se ha modificado get_transactions_data que ahora incluye validaciones para asegurarse de que los valores ingresados sean válidos. Por ejemplo, se agregó la validación valid_number? para asegurarse de que el valor ingresado para la cantidad sea un número válido. También se agregaron ciclos while para asegurarse de que los campos obligatorios no estén vacíos y se añadieron mensajes de error más descriptivos para ayudar al usuario a ingresar valores válidos.

```ruby
def get_transactions_data
    def valid_number?(input)
      input.match?(/^\d+(\.\d+)?$/)
    end

    transactions_block ||= []
    blank_transaction = Hash[from: "", to: "", what: "", qty: ""]
    loop do
      puts ""
      puts "Enter your name for the new transaction"
      from = gets.chomp
      while from.empty?
        puts "This value cannot be empty"
        print "Enter the name for the transaction: "
        from = gets.chomp
      end

      puts ""
      puts "What do you want to send ?"
      what = gets.chomp
      while what.empty?
        puts "This value cannot be empty."
        print "Enter what you want to send: "
        what = gets.chomp
      end

      puts ""
      puts "How much quantity ?"
      qty = gets.chomp
      while qty.empty? || !valid_number?(qty)
        if qty.empty?
          puts "This value cannot be empty."
        else
          puts "This value has to be a number."
        end
        print "Enter a number "
        qty = gets.chomp
      end

      puts ""
      puts "Who do you want to send it to ?"
      to = gets.chomp
      while to.empty?
        puts "This value cannot be empty."
        print "Enter the destination: "
        to = gets.chomp
      end

      transaction = Hash[from: "#{from}", to: "#{to}", what: "#{what}", qty: "#{qty}"]
      transactions_block << transaction

      puts ""
      puts "Do you want to make another transaction for this block ? (Y/n)"
      new_transaction = gets.chomp.downcase

      if new_transaction == "n"
        return transactions_block
      end
    end
end
```

Funcionamiento completo del sistema con las modificaciones implementadas:

Al pulsar el número 1, introducimos los parámetros:

```
Select an option:
1. Add a transaction
2. List all blocks
3. List all transactions
4. Save transactions to file
5. Exit
1

Enter your name for the new transaction
Jose

What do you want to send ?
euros

How much quantity ?
500

Who do you want to send it to ?
Chenlei

Do you want to make another transaction for this block ? (Y/n)
n
New block created with 1 transactions.
```

Al pulsar el número 2 aparecen los bloques:

```
#<Block:0x00000235c9f6b598
 @hash="003f8ffdb294adbbf0fc054272b69753b5e7d4df5e40fb45778bcba5de40580c",
 @index=0,
 @nonce=28,
 @previous_hash="0",
 @timestamp=2023-05-07 15:42:50.0380334 +0200,
 @transactions=
  [{:from=>"Dutchgrown", :to=>"Vincent", :what=>"Tulip Bloemendaal Sunset", :qty=>10},
   {:from=>"Keukenhof", :to=>"Anne", :what=>"Tulip Semper Augustus", :qty=>7}],
 @transactions_count=2>
==========================
#<Block:0x00000235c997e8d8
 @hash="00c63cdaa2ef4550ce40edab55f50dbe3c66227adc7de8bf91e20821ecbd7ebb",
 @index=1,
 @nonce=514,
 @previous_hash="003f8ffdb294adbbf0fc054272b69753b5e7d4df5e40fb45778bcba5de40580c",
 @timestamp=2023-05-07 15:43:15.937218 +0200,
 @transactions=[{:from=>"Jose", :to=>"Chenlei", :what=>"euros", :qty=>"500"}],
 @transactions_count=1>
==========================
Select an option:
1. Add a transaction
2. List all blocks
3. List all transactions
4. Save transactions to file
5. Exit
```

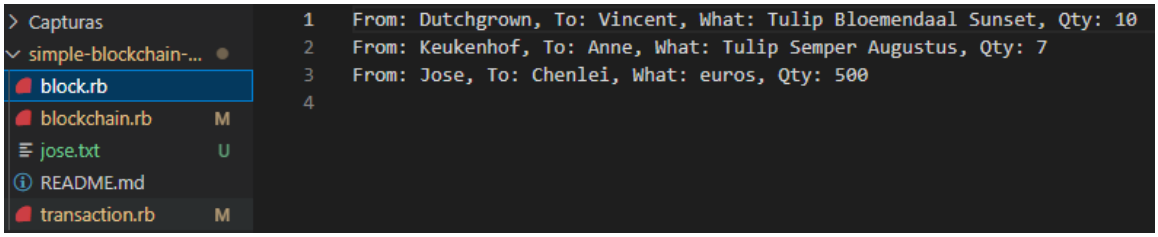
Al pulsar el número 3 podemos ver las transacciones:

```
3
From: Dutchgrown, To: Vincent, What: Tulip Bloemendaal Sunset, Qty: 10
From: Keukenhof, To: Anne, What: Tulip Semper Augustus, Qty: 7
From: Jose, To: Chenlei, What: euros, Qty: 500
Select an option:
1. Add a transaction
2. List all blocks
3. List all transactions
4. Save transactions to file
5. Exit
```

Al pulsar el número 4 guardamos las transacciones en el archivo de texto, y al pulsar el 5 terminamos:

```
4
Enter file name to save transactions to (without .txt extension):
jose
Transactions saved to jose.txt
Select an option:
1. Add a transaction
2. List all blocks
3. List all transactions
4. Save transactions to file
5. Exit
5
```

Aquí podemos observar el archivo txt:



El fork realizado con las modificaciones se puede encontrar en el siguiente respositorio:
https://github.com/saramarcoss/simple-blockchain-in-ruby

Conclusión

A lo largo de esta práctica hemos implementado una cadena de bloques con Ruby, donde hemos aprendido los conceptos básicos de la tecnología blockchain. Al implementar una cadena de bloques desde cero, se aprende cómo funcionan los bloques, cómo se agregan a la cadena, cómo se asegura la integridad de la cadena.

Se puede comprender cómo se implementan los algoritmos de prueba de trabajo y cómo se aplican a la minería de bloques. Esto es esencial para entender cómo se garantiza la seguridad y la inmutabilidad de la cadena de bloques.
Estos conceptos, pueden ser aplicados a problemas del mundo real y para diseñar soluciones que aprovechen el poder de la tecnología blockchain.

Bibliografía

Instalación de Ruby -> https://www.ruby-lang.org/es/
Información útil
https://www.youtube.com/watch?v=SSo_EIwHSd4
https://blog.bitnovo.com/que-es-un-bloque-genesis/
https://academy.bit2me.com/que-es-bloque-genesis/
https://www.itdo.com/blog/que-son-los-bloques-en-la-tecnologia-blockchain/
https://www.ibm.com/topics/blockchain
https://www.investopedia.com/terms/b/blockchain.asp