VirtualMall [Fase 4]

Primer Semestre 2021

—

Ing. Jesus Guzman, Ing. Alvaro Hernandez, Ing. Luis Espino

Marvin Martinez, Andree Avalos, Randolph Muy, Jorge Salazar, Josué Pérez, José Véliz

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Estructuras de Datos

Visión general

A raíz de la crisis sucedida por la pandemia y la casi imposible movilización para poder realizar cualquier tipo de compras, la empresa EDDsystems le ha encargado a usted como desarrollador crear un proyecto que sea capaz de gestionar tiendas simulando el acceso a centro comercial con los comercios que este posee de manera segura y confiable.

VirtualMall será un aplicación que le permita a los clientes interactuar con las tiendas que ellos deseen de manera eficaz y segura, para esto, el sistema debe ser capaz de manejar los distintos servicios que las tiendas brindarán a sus clientes, como lo son las ventas bajo pedido y los posibles envíos a distintas locaciones, también tener la capacidad de guardar toda la información que se necesite de las tiendas, sus productos, sus servicios y sus clientes, estos deben ser almacenados de forma segura y confiable.

Objetivos

1. Que el estudiante se familiarice con el lenguaje Go
2. Que el estudiante conozca el funcionamiento de peticiones REST mediante la implementación de un servidor web en Go
3. Que el estudiante sepa envolverse en el ámbito de manejo de la memoria
4. Comprender y desarrollar distintos tipos de estructuras no lineales
5. Familiarizarse con el manejo de lectura y escritura de archivos JSON
6. Familiarizarse con el uso de Git.
7. Que el estudiante se familiarice con el lenguaje Javascript y sus frameworks.

Especificaciones

Se solicita continuar la aplicación de servidor realizada en la fase 2 agregando cambios y nuevas funcionalidades. En esta parte necesitamos páginas web que se puedan conectar al servidor, el frontend tiene que estar desarrollado en Javascript utilizando cualquier tipo de framework que el estudiante desee utilizar (recomendaciones: Angular o Reactjs), o también puede utilizarse Javascript sin frameworks. La parte del frontend también tiene que estar en el mismo repositorio que el servidor y se desea que sea intuitivo y fácil de usar para todos los tipos de usuarios.

Árbol de Merkle

En VirtualMall dado que necesitamos tener una persistencia de los datos, siendo estos confiables, no solo para nuestros clientes si no para futuras auditorias, por lo mismo se piensa en una forma de poder implementar una estructura que tenga la capacidad de contener todos estos beneficios, el funcionamiento del árbol de merkle es guardar las transacciones que se hagan dentro de la aplicación, para esto necesitamos varios pasos.

Generar una transacción

Para poder insertar en el árbol de merkle, se tiene que pasar por el proceso de tener un pedido(ir a producto, enviar a carrito) y a la hora de pagar, se tiene que simular la transacción con el banco para su futura integración con varios métodos de pago. Cuando salga exitosa la transacción este tiene que generar un nodo con los datos que crea necesario para que el hash generado sea único.

Ejemplo

id = sha256(Data)

a. Donde id es con el que se debe guardar en el árbol de merkle
b. sha256 es el algoritmo que crea el hash deseado.
c. Data: Corresponde a toda la información que generará el hash, debe incluir todos los datos de la transferencia, además que tienen que tener el dpi del usuario y la fecha de la transacción realizada.

Ejemplo de nodos hojas

id = sha256("**3425123542135,13/4/2021,700.66,...**")

id = 39b923082a194166d8d92989116bd2ed93fd4ba3c68c345d0c6d9140538886bf

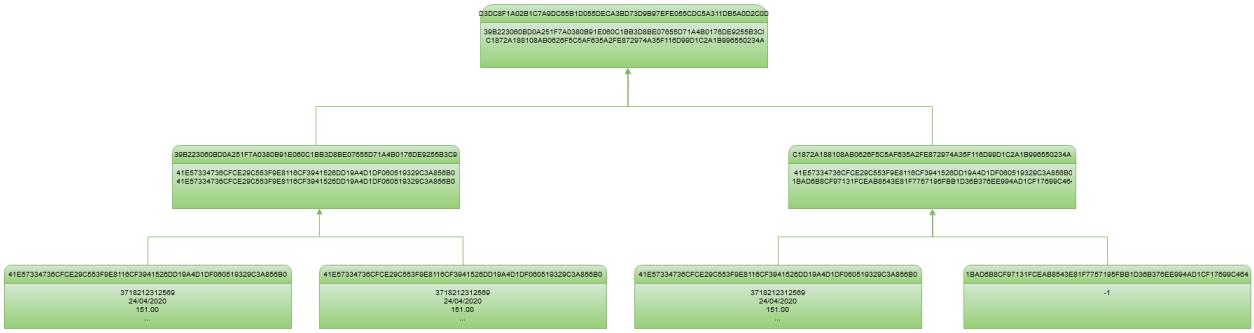
Se crea un nodo interno cuando una transacción está esperando al número mínimo para poder crear un hash (2 valores hash como mínimo para poder generar un nuevo hash)

Ejemplo de nodos Internos

id = sha256("+39b923082a194166d8d92989116bd2ed93fd4ba3c68c345d0c6d9140538886bf,4aa17d5fdf24761b54ad640691146656095ba38e5f92544c73e843db1a2f3875")

id = 113b685b12dbfa76c04bec7759a24ba66dd6a72c7c6d89159149e56567f87fe4

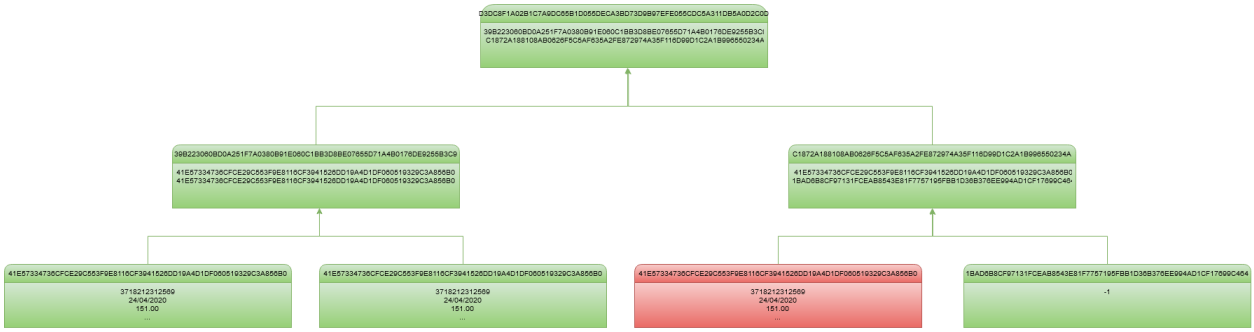
Ejemplo de árbol correcto



Comprobación de transacciones válidas, se debe mostrar de forma gráfica el arbol de merkle donde se vean que todos los hash corresponden, si por algun dado caso se modifica alguna transaccion debe mostrarse de color rojo donde se rompe la relacion.

La transacción puede ser editada, y al ser editada esta debe reemplazar el hash que tenía como nodo.

Ejemplo de árbol incorrecto



Para poder volver al estado correcto únicamente el usuario administrador puede hacer esta acción, para esto se debe tener un botón del lado del administrador que genere nuevamente los hash para que concuerden los hash y vuelva a estar en estado correcto.

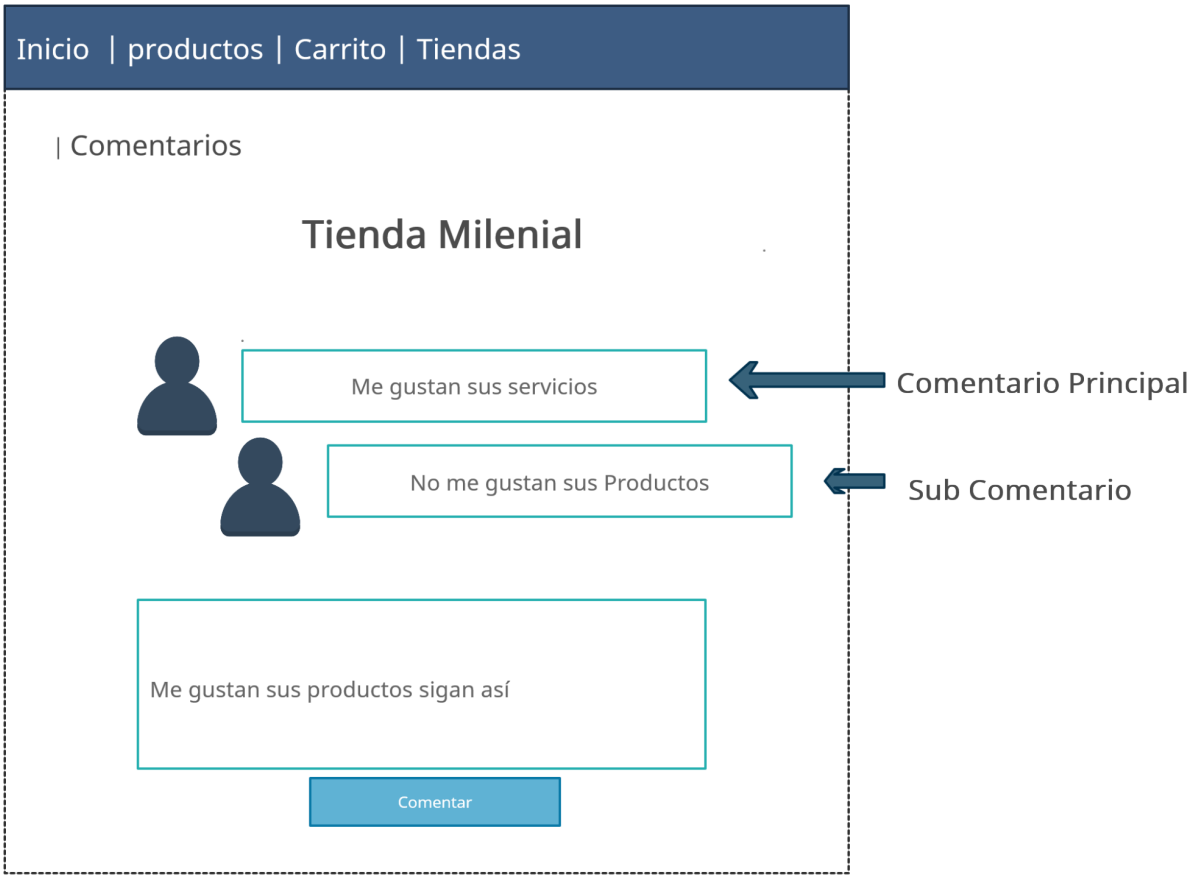
Nota: El árbol de Merkle siempre debe estar completamente lleno, de tal manera que si el número de transacciones no completa el árbol, entonces debe de llenarse el mismo con valores representativos de nulos como por ejemplo un -1.

Tabla Hash

Las tiendas que están asociadas a Virtual Mall, solicitan una nueva funcionalidad por parte de la plataforma Ecommerce, la cuál consiste en un apartado de comentarios, en donde las tiendas pueden retroalimentar la modalidad de sus servicios con respecto a las opiniones de sus clientes,por lo tanto, se solicita agregar un apartado de comentarios para las tiendas y un apartado por cada producto que las tiendas vende.

Modalidad de los comentarios

- por cada tienda debe existir una apartado para los comentarios
- Por cada producto debe existir un apartado para los comentarios
- Un comentario principal puede tener respuestas o subcomentarios
- un sub comentario puede tener otro sub comentario



Utilización de Tabla Hash

- La dirección por cada elemento de la tabla hash es el ID del usuario o DPI
- Cada elemento de la tabla hash que se identifica por el Id de usuario apunta a Nodo que contiene el comentario
- Cada nodo apunta otra tabla hash, que funciona como subcomentarios
- Por lo tanto puede existir recursividad de Tablas Hash por cada sub comentario

Para calcular las claves, la función hash a utilizar debe ser el **método de multiplicación**, el método para resolver colisiones debe ser **exploración cuadrática.** La llave a utilizar para la función hash será el **DPI del usuario**.

Función hash método de multiplicación:

$$h(k) = \lfloor m * (k*A \bmod 1) \rfloor$$

donde:

- k= llave.
- 0< A < 1 (se recomienda un numero con 4 decimanes)
- m = tamaño actual de la tabla.

El tamaño inicial de la tabla hash será de 7. Cuanto la tabla hash llegue a un **60% de uso**, se tendrá que hacer rehash de la tabla el cual será al **número primo próximo.**

Almacenamiento de información segura

Para mantener el almacenamiento de la información de manera segura y óptima se deberá guardar toda la información de las operaciones realizadas dentro de la plataforma dentro de árboles de Merkle de tal manera que se tenga un árbol de merkle que maneje todas las operaciones relacionadas sobre un tema en específico, a continuación se describen los elementos de los cuales se debe crear un árbol de Merkle.

1. Tiendas
2. Productos
3. Usuarios

La estructura de los árboles de Merkle es la misma que la descrita al inicio del enunciado, por lo cual se debe respetar ese orden.

Por último, es importante saber que las sumarizaciones finales de los árboles de Merkle permitirán la creación de los bloques de Blockchain ya que así se tendrá un manejo de la información seguro, óptimo e integral.

Blockchain

En esta estructura de datos se van a guardar todas las transacciones que se encuentran en todas las estructuras de datos que ya se han creado con anterioridad, esto se realizará cada cierto periodo de tiempo que puede variar dependiendo de las configuraciones realizadas.

Inicialmente el tiempo será de 5 minutos, por lo cual ya pasado este tiempo se debe realizar lo siguiente:

Nuevo bloque:

Pasado el tiempo de configuración de la aplicación se genera un nuevo bloque que almacena las sumarizaciones de los árboles de Merkle anteriormente descritos y si no es posible entonces todo tipo de transacciones tales como: realizar un pedido, agregar producto, crear un usuario, agregar una tienda, etc., además cada bloque creado debe presentar la siguiente estructura:

- **Índice**: El número de correlativo del bloque, el bloque inicial tendrá el valor de 0.
- **Fecha**: Almacena el momento exacto en el que se genera el bloque, debe de tener el siguiente formato (DD-MM-YY::HH:MM:SS).
- **Data**: Deben ser todas la sumarizaciones de los árboles de Merkle ya creados con anterioridad, si no se cuenta con los árboles de Merkle entonces deben ser todos los datos que se realizaron durante el periodo de utilización de este bloque, creaciones, eliminaciones o actualizaciones.
- **Nonce:** Es un número entero que se debe iterar de uno en uno hasta encontrar un hash válido, es decir un hash que inicie con cuatro ceros.
- **PreviousHash:** Es el hash del bloque anterior, nos permite validar que la cadena del bloque no está alterada. Para el primer bloque el hash anterior debe ser 0000.
- **Hash:** Es el hash del bloque actual.

Crear el Hash:

Para crear el hash de este bloque se va a hacer uso de SHA256, utilizando las propiedades de índice, Fecha, Data, PreviousHash y Nonce.

Todos estos campos tienen que ir como cadenas concatenadas sin espacios en blanco.

SHA256(Indice + Fecha + PreviousHash + Data + Nonce)

Para que el hash cumpla con la condición de que contenga cuatro ceros al inicio se va a hacer uso de la prueba de trabajo o minar un bloque.

Prueba de trabajo

Es el proceso por el cual se encuentra un hash que cumpla con la condición de tener un prefijo de cuatro ceros. Para ello se debe de iterar el campo Nonce hasta encontrar un hash válido para el bloque.

Guardar bloque

Luego de terminar el proceso de crear un nuevo bloque, se procede a almacenar el archivo en la carpeta de bloques.

Para generar el bloque el estudiante debe generar un archivo donde guardará la información necesaria para que al ser leída el proyecto siga teniendo los mismos datos que al cerrarse.

Estos bloques no son legibles para las personas que quieren o tienen intención de ver o manipular los datos.

Configuración de bloques

El administrador puede cambiar el lapso de tiempo entre cada bloque, el tiempo va a estar siempre en minutos. Este tiempo inicia después de iniciar la aplicación y luego de que carguen los bloques anteriores.

Arranque de la aplicación

Al iniciar la aplicación se procede a descifrar cada uno de los bloques generados, luego se hace la descompresión de los bloques y se procede a cargar las estructuras.

Cierre de Aplicación

La aplicación debe tener la capacidad que antes de cerrarse, pueda guardar la última instancia de las transacciones.

Ejemplo

Notas:

1. Los bloques deben guardarse en la carpeta “bloques”, de no existir se debe crear.
2. El nombre de los bloques debe ser manejado por correlativos para evitar que reemplace bloques y se pierda información.
3. Si no hay nuevas transacciones en la estructura,entonces no se genera un bloque, esto para evitar que existan bloques vacíos sin información.
4. Solo el usuario administrador debe tener acceso a esta interfaz.

Consideraciones

I. Lenguaje a utilizar será **Go y Javascript**

II. **Puerto** por defecto **3000** para el lenguaje **Go,** y para **Javascript** es **libre.**

III. Las estructuras serán realizadas por el estudiante.

IV. IDE a utilizar **Goland** o **Visual Studio Code.**

V. La entrega será por medio de la plataforma de **UEDI.**

VI. El estudiante debe tener un repositorio privado en github con el nombre **EDD_VirtualMall_#Carné** y agregar a su tutor como colaborador al repositorio del proyecto (Cada tutor les hará llegar su usuario)

VII. Será calificado del último commit realizado antes de la fecha de entrega.

VIII. Las copias totales o parciales **serán penalizadas con nota de 0 puntos** y reportadas ante la escuela de ciencias y sistemas.

IX. Todos los reportes deben ser generados con graphviz.

X. **Fecha de entrega: 08 de mayo de 2021 antes de las 23:59 horas.**