Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y sistemas
Estructuras de Datos
Ingenieros:

- Ing. Luis Espino
- Ing. Jesus Guzmán
- Ing. Álvaro Hernández

Auxiliares:

- Alex Lopez
- Walter Mach
- Wilfred Perez

Proyecto 1 Fase 3

UDrawing Paper

Objetivos

Objetivo general

- Aplicar los conocimientos del curso Estructuras de Datos en el desarrollo de soluciones de software.

Objetivos específicos

- Aplicar los conocimientos adquiridos sobre estructuras de datos lineales y no lineales como matrices, árboles, tablas hash
- Implementar una aplicación de escritorio utilizando el lenguaje de programación Java
- Familiarizarse con la lectura y escritura de archivos de JSON.
- Utilizar la herramienta Graphviz para graficar estructuras de datos no lineales.
- Definir e implementar algoritmos de búsqueda, recorrido y eliminación en estructuras de datos.
- Utilizar los conceptos generales de la tecnología Blockchain.

Descripción General

La empresa Drawing Paper durante mucho tiempo únicamente ha prestado su servicio de impresión de forma presencial, con cobertura en distintos puntos del país. Es decir, el cliente llega a cada uno de los centros de impresión, solicita sus impresiones de acuerdo a las características que desea, luego espera hasta que el proceso finalice, para posteriormente retirarse.

Udrawing Paper ha decidido realizar un cambio en la forma de atención al cliente, introduciendo el servicio de solicitudes de impresión en línea y entrega a domicilio. Por lo que se le solicita a usted que ha desarrollado la aplicación principal en las fases anteriores, pueda implementar esta nueva funcionalidad, con el fin de agilizar los procesos internos de la empresa, mejor control de la información y brindar un mejor servicio a los clientes

Actualizaciones de Clientes

Debe almacenarse la siguiente información para los clientes, por lo que debe actualizar los campos que se almacenaban en la fase anterior.

- DPI
- Nombre completo
- Nombre de usuario (único dentro de la aplicación)
- Correo
- Contraseña
- Teléfono
- Dirección
- Id_Municipio

Modificaciones del inicio de sesión:

El inicio de sesión se realizará por nombre de usuario y contraseña

Mensajeros

Para cada mensajero se manejan la siguiente información:

- DPI
- Nombres
- Apellidos
- Tipo de licencia (A, B, C)
- Género
- Teléfono
- Dirección

Los mensajeros se almacenarán en una tabla hash, utilizando como llave el número de DPI

Características de la tabla hash:

- El tamaño inicial de la tabla hash sera 37 (M=37)
- La función de dispersión a utilizar es: $h(llv) = llv \bmod M$
- El porcentaje máximo de ocupación será el 75%

// cuando llegue al 75% deberá aumentar
// con el reshah vamos a utilizar la redimensionacion al primo siguiwente

//colisiones

- La política para la resolución de colisiones será la doble dispersión por medio de la función: $s(llv, i) = (llv \bmod 7 + 1) * i$

// i = la posición de la llave

Carga masiva de mensajeros:

La aplicación debe contar con la opción para realizar la carga masiva de mensajeros por medio de un archivo de texto con el siguiente formato

```
[
  {
    "dpi":"0717882466706",
    "nombres":"Devonna",
    "apellidos":"Dovey",
    "tipo_licencia":"A",
    "genero":"Female",
    "direccion":"72329 Maple Alley"
  },
  {
    "dpi":"8208175148573",
    "nombres":"Ulberto",
    "apellidos":"Goodsell",
    "tipo_licencia":"B",
    "genero":"Male",
    "direccion":"5 Northfield Hill"
  },
  {
    "dpi":"3198208768738",
    "nombres":"Burnaby",
    "apellidos":"Coviello",
    "tipo_licencia":"c",
    "genero":"Male",
    "direccion":"3278 Dixon Center"
  }
]
```

Entrega a domicilio

Carga de Lugares:

Al iniciar la aplicación, se deberá pedir la carga de un archivo el cuál contendrá los municipios que más adelante formarán los vértices en el grafo.

- id
- departamento
- nombre
- sn_sucursal (indica si existe una sucursal en el municipio)

```
"Lugares": [
  {
    "id": 17,
    "departamento": "Guatemala",
    "nombre": "Amatitlán",
    "sn_sucursal": "si",
  },
  {
    "id": 13,
    "departamento": "Guatemala",
    "nombre": "Villa Nueva",
    "sn_sucursal": "no",
  }
]
}
```

// haber si es una sucursal, esto es para ver de donde va a salir el pedido

// acá vamos a llenar nuestra lista de adyacencia

Rutas:

Luego de cargar los lugares se deberá pedir la carga de un archivo el cuál contendrá las rutas entre municipios de un departamento, en el archivo se proporcionará lo siguiente:

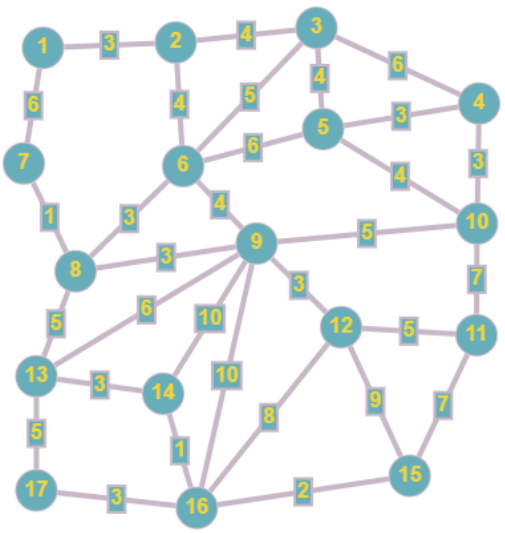
- inicio(Id del municipio)
- final (Id del municipio)
- peso (Tiempo de Ruta en minutos)

Las rutas se cargarán al iniciar la aplicación, por medio de un archivo JSON con el siguiente formato:

```
{
  "Grafo": [
    {
      "inicio": 17,
      "final": 13,
      "peso": 5
    },
    {
      "inicio": 17,
      "final": 16,
      "peso": 3
    },
    {
      "inicio": 14,
      "final": 13,
      "peso": 3
    },
    {
      "inicio": 14,
      "final": 16,
      "peso": 1
    },
    {
      "inicio": 8,
      "final": 13,
      "peso": 5
    }
  ]
}
```

Cada lugar tiene un nombre único, y el tiempo de ruta será proporcionado en minutos.

//Peso de 3, serían 3 minutos
// vamos a buscar un recorrido hacia una sucursal. que tenga el menor peso posible

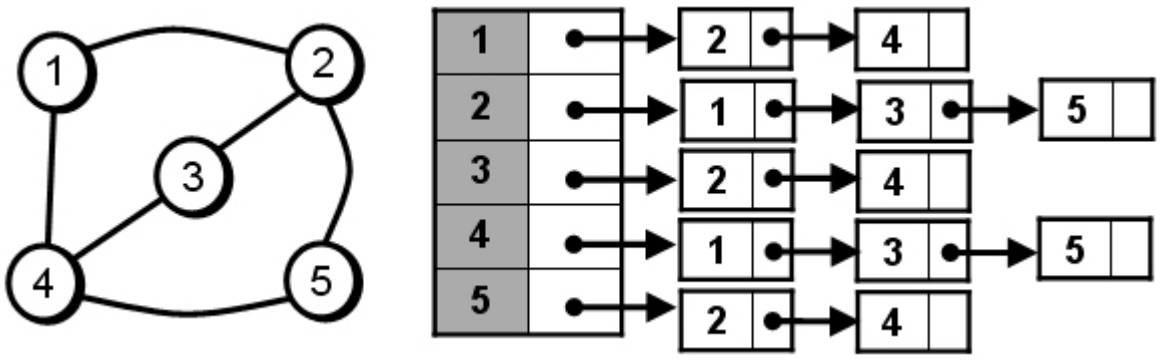


Grafo no dirigido de los municipios del departamento de Guatemala

// para el proyecto lo que podemos hacer es solo tener la carga de las imágenes, y poder mandar a imprimirlas, y que quede en el regustro lo que se hizo, para poder ir a entregarlos

// como es no dirigido, ahora debemos ingresar ambos caminos por conexion de A->B y de B -> A

Luego de realizar las cargas de los archivos se generará una lista de adyacencia la cual permitirá generar el grafo no dirigido



Ejemplo lista de adyacencia que representa el grafo no dirigido de la izquierda

Operación de Entrega:

// el usuario puede escoger el mensajero y la sucursal

Cuando un usuario genere una impresión deberá ser capaz de seleccionar un mensajero para que realice la entrega de dicha imagen y la sucursal de donde se desea obtener la impresión, el mensajero iniciara el recorrido en el grafo desde la dirección de la sucursal seleccionada, para realizar el recorrido se deberá tener en cuenta la siguiente información:

- impresión (dirección de la sucursal)
- Fecha de solicitud // get noun
- Lugar de destino(Id municipio del cliente)
- Cliente (Apuntador a árbol B) // debemos gusrdar este apuntador de alguna forma
- Mensajero (apuntador a tabla hash)
- Ruta seleccionada (Lista simplemente enlazada)
    - La aplicación debe ser capaz de determinar el mejor camino por tomar, basándose en el grafo generado con la carga masiva de rutas, una vez generado el mejor camino este deberá guardarse en una lista simplemente enlazada, cada nodo debe tener: Nombre del lugar, enlace al siguiente lugar, y tiempo de llegada.

Almacenamiento de operaciones de entrega

Debido a que es necesario tener persistencia en los datos que se almacenan en la aplicacion, para que estos sean confiables, no solo para los usuarios sino también para futuras revisiones de la empresa, es necesario implementar una estructura de datos que pueda mantener la información íntegra, el árbol de Merkle es una estructura de datos que sirve para realizar el guardado de transacciones que se realizan dentro de las aplicaciones, para lo cual se necesita seguir ciertos pasos.

Este árbol permite registrar todas las operaciones que son realizadas dentro de la plataforma, con esto se refiere a que debe de crearse un árbol para almacenar las operaciones de entrega de impresiones.

Generar una operación en el arbol de Merkle

Para poder insertar en el árbol, se debe realizar un envío de impresiones.

Cuando el proceso sea generado de manera exitosa se debe generar un nodo con todos los datos indicados en la sección **"Operación de entrega"** para generar un hash que sea único.

Ejemplo

id = funcionHash(Data)
Donde:

// función que vamos a utilizar para encriptar podemos hacerlo desde una librería

- **id** es con el que se debe guardar en el árbol de merkle
- **funcionHash:** método utilizado para encriptar la información, la función que se utilizara para la generación del hash sera sha256.
- **Data:** Corresponde a toda la información que generará el hash, debe incluir todos los datos de la operación realizada.

Ejemplo de nodos hojas

id = funcionHash("3425123542135 , 13/4/2021 , ...")

id = 39b923082a194166d8d92989116bd //resultado del hash

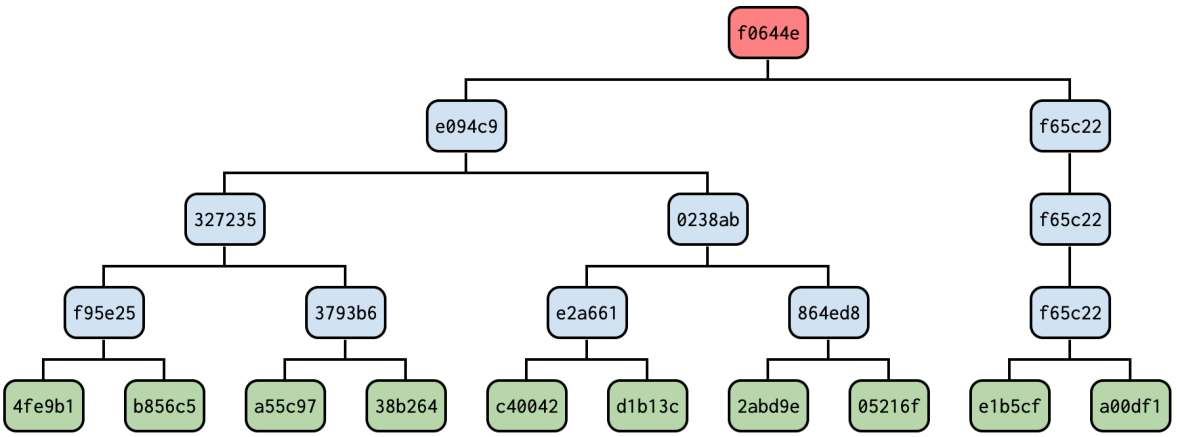
Se crea un nodo interno cuando una transacción está esperando al número mínimo para poder crear un hash (2 valores hash como mínimo para poder generar un nuevo hash).

Ejemplo de nodos internos (se toman los hash de los nodos hoja para generar el hash del nodo interno)

id=funcionHash("39b923082a194166d8d92989116bd, aa17d5fdf24761b54ad640691146656095b")

id = 113b685b12dbfa76c04bec7759a24ba66dd

Ejemplo de salida de un árbol Merkle



Notas:

- Las cadenas de caracteres utilizadas en la ejemplificación del proceso de construcción del árbol no son cadenas válidas correspondientes a algún método de cifrado existente, por lo que el árbol mostrado es solo una representación de la estructura de datos.
- El árbol de merkle se caracteriza por ser un árbol que siempre se encuentra completamente lleno, así que cuando el número de transacciones no complete el árbol, este debe de llenarse con valores representativos como nulos, por ejemplo **-1**.

Para la comprobación de las transacciones válidas, se debe mostrar de forma gráfica el árbol para poder visualizar que todos los hash correspondan, para ello **se deben leer los bloques de blockchain** (descritos en la siguiente sección) y luego validar los hashes de merkle y los hashes de la cadena de bloques, si por algún caso se modifica alguno de los procesos o asignaciones debe mostrarse de diferente color donde se rompe la relación.

// Esto para encontrar errores

Blockchain

La aplicación va a funcionar utilizando los conceptos de blockchain. Esta estructura almacena las operaciones de entrega realizadas dentro de la aplicación.

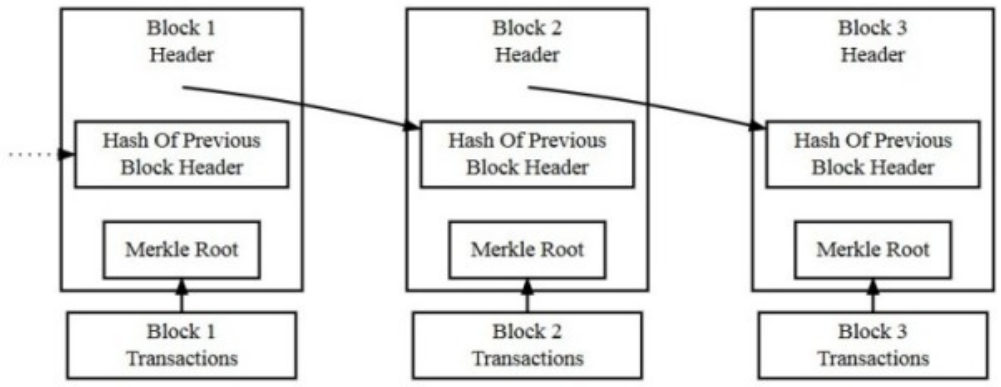
Almacenamiento de información en árboles Merkle:

Inicialmente se estableció que se debe guardar la información de todas las entregas realizadas dentro de la plataforma en los árboles merkle para validar la consistencia de la información, por lo cual esto se mantendrá en memoria mientras transcurre el tiempo de la creación del nuevo bloque, en el momento que se cumple el tiempo el bloque se escribirá, los arboles de merkle deben reiniciarse, esto para poder guardar la nueva informacion que se incluirá en el nuevo bloqeue.

Si no existieron operaciones realizadas desde la creación del último bloque, el nuevo bloque se debe generar pero los datos de las operaciones deberán estar vacías.

Estructuras a utilizar en el Blockchain

- **Blockchain:** Esta es una estructura de bloques, los cuales se comportan como una lista enlazada simple, en donde cada uno de los nodos (bloques) se ingresan por delante.



Simplified Bitcoin Block Chain

// La prueba de trabajo

Prueba de Trabajo

Es el proceso por el cual se encuentra un hash que cumpla con la condición de tener un prefijo de n ceros, donde n corresponde a un número entero, este puede ser cambiado por el administrador y por defecto tendrá el valor de 4. Para ello se debe de iterar un entero denominado **NONCE** hasta encontrar un hash válido para el bloque.

// El algoritmo que vamos a resolver es encontrar un hash válido, con 4 ceros al iniScio del has anterior, debemos ir iterando e iterando hasta enconcontrar un has de este tipo

Operaciones de Blockchain

Nuevo Bloque: Pasado el tiempo de configuración en la aplicación se genera un nuevo bloque que almacena las sumarización del árbol de merkle anteriormente descrito. El tiempo en que un nuevo bloque se genera está determinado por un valor m, el cual es un número entero representando la cantidad de minutos, este puede modificarse y por defecto tendrá el valor de 3 (minutos).

// cada 3 min se genera un nuevo bloque, y el bloque contiene info del árbol de mercle actualizado, guardado cda 3 min

Bloque
El bloque se define de la siguiente manera:

- **INDEX:** representa el número de bloque el bloque génesis tendrá valor de index 0 , los bloques posteriores deberán tener valores 1, 2, 3, 4 ... etc.
- **TIMESTAMP (fecha y hora de creación):** Es la fecha y hora exacta en la que se creó el bloque**.** Debe de tener el siguiente formato: (DD-MM-YY-::HH:MM:SS).
- **DATA:** Contendrá cada una de las entregas realizadas por los clientes los que serán almacenados en este apartado.
- **NONCE:** Será el número entero que se debe iterar de uno en uno hasta encontrar un hash que cumpla con la prueba de trabado, es decir que contenga como prefijo un numero de 0s configurado por el adminsitrador.

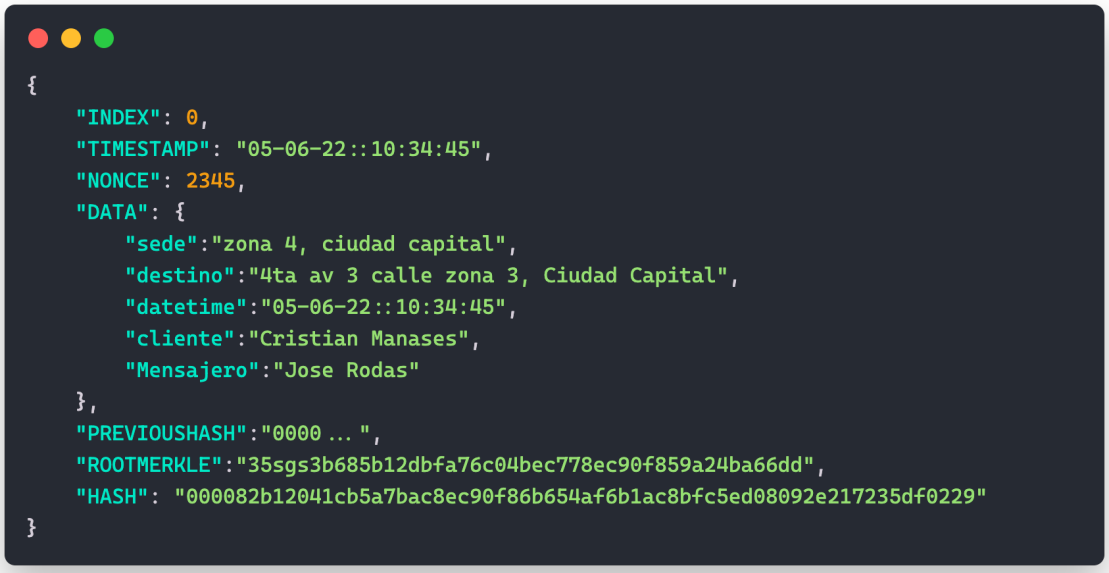
// volvemos a calcular el hash del anterior y comparamos

- **PREVIOUSHASH:** Es el hash del bloque previo, este es necesario para validar que la cadena de bloques no esté corrupta. En caso del bloque génesis, el hash anterior debe de ser 0000.
- **ROOTMERKLE**: En este bloque se almacena el nodo padre del árbol de Merkle.Este árbol de Merkle se forma con los datos del campo DATA, que son las operaciones de entrega
- **HASH (bloque actual):** El hash que protege que la data no se ha comprometido, el hash deberá generarse aplicando la función SHA256 a las propiedades: INDEX, TIMESTAMP, PREVIOUSHASH, ROOTMERKLE y NONCE **todas estas propiedades como cadenas concatenadas sin espacios en blanco ni saltos de línea**. Es decir SHA256(INDEX+TIMESTAMP+PREVIOUSHASH+ROOTMERKLE+NONCE). Para considerar el hash como válido este debe de tener un prefijo de cuatro ceros. Es decir que un hash valido sería el siguiente: 000082b12041cb5a7bac8ec90f86b654af6b1ac8bfc5ed08092e217235df0229

// cuando se genere un nuevo bloque se debe vaidar que el anterior sí esté bien

Definición de bloques

Cada uno de los bloques tendrá la siguiente estructura

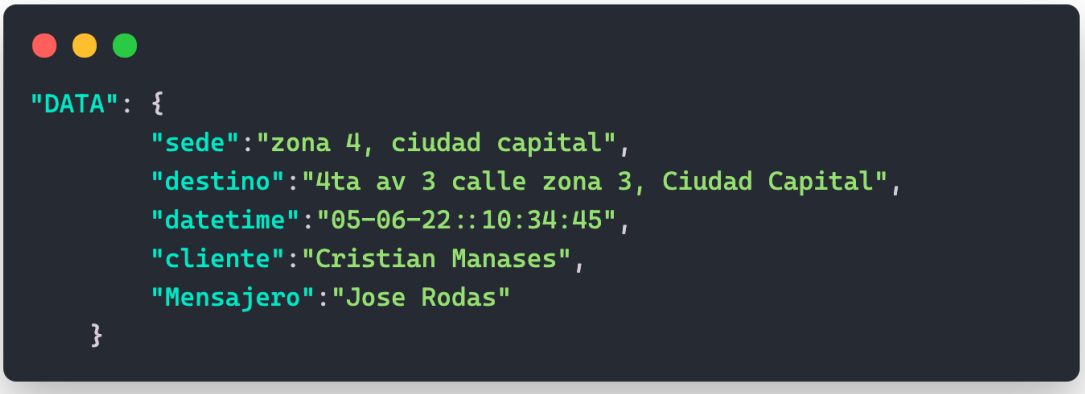


```
{
    "INDEX": 0,
    "TIMESTAMP": "05-06-22::10:34:45",
    "NONCE": 2345,
    "DATA": {
        "sede":"zona 4, ciudad capital",
        "destino":"4ta av 3 calle zona 3, Ciudad Capital",
        "datetime":"05-06-22::10:34:45",
        "cliente":"Cristian Manases",
        "Mensajero":"Jose Rodas"
    },
    "PREVIOUSHASH":"0000 ... ",
    "ROOTMERKLE":"35sgs3b685b12dbfa76c04bec778ec90f859a24ba66dd",
    "HASH": "000082b12041cb5a7bac8ec90f86b654af6b1ac8bfc5ed08092e217235df0229"
}
```

En la sección DATA se colocarán los datos de las operaciones de entrega que se ejecutaron en el nodo que creó el bloque.

Operación Entrega

La operación representará una solicitud de entrega, la definicion sera la siguiente:



```
"DATA": {
        "sede":"zona 4, ciudad capital",
        "destino":"4ta av 3 calle zona 3, Ciudad Capital",
        "datetime":"05-06-22::10:34:45",
        "cliente":"Cristian Manases",
        "Mensajero":"Jose Rodas"
    }
```

Guardar bloque

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

Al momento de iniciar el programa se deben leer los bloques que ya existan.

Nombre de la carpeta de bloques: Bloques

Ruta de la carpeta de bloques: C:\udrawing\blockchain\

Nombres de los bloques: INDEX_TIMESTAMP.json donde INDEX Y TIMESTAMP son atributos definidos en el bloque.

Configuración de bloques:

El tiempo por defecto de la creación de bloques será de 3 minutos.

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

También podrá crear un bloque inmediatamente sin necesidad de esperar el tiempo establecido.

Arranque de la aplicación:

// Implementar un contador de tiempo, con hilos

Al iniciar la aplicación se procede a leer cada uno de los bloques generados, luego se procede a cargar las estructuras de datos.

Cierre de la aplicación:

// aplicaciones paralelas

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

Notas:

de genenerar nuevos bloques y de leer los bloques ya existentes

1. Solo el usuario administrador debe tener acceso a la interfaz de configuración de blockchain.

Reportes

// como el tiempo de genreación de bloque y otras cosillas que se pueden modificar

Gráficos:

- Tabla de dispersión de mensajeros el reporte debe mostrar todos los mensajeros que están registrados en las distintas sedes de la empresa.
- Grafo de rutas
- Lista de adyacencia
- Nodos de la red // cantidad de veces que se abre la ap
- Blockchain que representa el log de las operaciones de entrada realizadas dentro de la aplicación.

Datos de la empresa:

// podemos ir recorriendo y guardando en una lista ordenada

- Top viajes: 10 entregas con mas distancia.
- Top clientes: 10 clientes con mayor cantidad de solicitudes de entrega.
- Top Mensajeros: 10 mensajeros con la mayor cantidad de entregas.
- [redacted]

Compresion y descompresion de la información

Dentro de la aplicación se podrán escribir archivos con información específica, estos archivos deben ser comprimidos o descomprimidos utilizando el método de Huffman.

// estos reportes se van a poder guardar, comprimir y descomprimir

La información será la siguiente:

- Copias de seguridad de los datos almacenados en cualquiera de las estructuras.
- Reportes generados dentro de la aplicación.

Los archivos deberán tener la extensión .edd

Protección de los datos

Para asegurar la información de los clientes, es necesario implementar buenas prácticas de seguridad, para ello es necesario proteger la información importante o sensible de cada uno de ellos.

Al momento de realizar las acciones de: **carga de información de clientes y registro de clientes** se debera utilizar en el campo de contraseña **Bcrypt la cual es** una función de hashing de passwords

Restricciones

Las estructuras deben ser desarrolladas por los estudiantes sin el uso de ninguna biblioteca o estructura predefinida en el lenguaje a utilizar.

- Se calificará desde el ejecutable, el cual estará dentro de su repositorio y dentro del archivo en UEDI.
- Se validará que los reportes estén correctos ya que únicamente se calificarán las estructuras graficadas, sin importar que exista el código de la implementación.
- Se deberán realizar un mínimo de 3 commit coherentes por semana para validar la continuidad de su flujo de trabajo. Se penalizará con el 25% de la nota final si no se cumple este requisito.

Observaciones

- Lenguaje de programación a utilizar: JAVA
- Sistema Operativo: Libre
- IDE: Libre.
- Herramienta para desarrollo de reportes gráficos: **Graphviz.**
- Las gráficas deben mostrarse dentro de la aplicación, no buscarse en carpetas ajenas.
- Durante la calificación se harán preguntas para validar que el estudiante realizó el proyecto, de no responder correctamente anulará la nota obtenida en la o las secciones en la que se aplique tal concepto.
- Cada estudiante deberá utilizar el repositorio de la fase anterior separando por carpetas cada fase del proyecto, verificar que su auxiliar esté agregado como colaborador para poder analizar su progreso.
- Apartado de entrega en la plataforma UEDI:
- Fecha y hora de entrega: 5 de mayo a las 23:59 horas.
- Las copias serán penalizadas con una nota de 0 y castigadas según lo indique el reglamento.

Entregables

- Manual de Usuario
- Manual Técnico
- Link a repositorio
- Código fuente
- Ejecutable