



SafeWay

Segundo Semestre 2020

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

Marvin Martinez, Juan Landaverde, Andree Avalos

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Estructuras de Datos



Visión general

Un medio de transporte seguro en Guatemala es de vital importancia, por esto, la empresa “Guateroad” le ha encargado como desarrollador crear un proyecto que sea capaz de poder gestionar pagos, viajes, cálculo de las mejores vías para poder llegar a su destino, estar seguro que los datos enviados sean confiables y consistentes.

SafeWay es una aplicación que le permite al usuario poder viajar de un **punto A** hacia un **punto B** de la manera más rápida y eficaz, para esto, el programa debe ser capaz de soportar cargar puntos en el mapa brindado por **Google Maps** y poder calcular la mejor opción, también se debe tener la capacidad de guardar información proveniente de los viajes, así también como los conductores y usuarios, a su vez se debe tener una forma de guardar los pagos realizados por cada usuario, estos deben ser almacenados de forma segura, contando con encriptación y compresión de datos.

Objetivos

1. Que el estudiante se familiarice con el lenguaje Java
2. Que el estudiante sepa involucrarse en el ámbito de transacciones
3. Comprender y desarrollar distintos tipos de estructuras no lineales
4. Tener la capacidad de aplicar conocimientos en programación orientada a objetos
5. Familiarizarse con el manejo de lectura de archivos JSON
6. Familiarizarse con el uso de Git.
7. Uso de cifrado y compresión de datos.

Especificaciones

La aplicación será realizada en el lenguaje de programación Java, utilizando su herramienta que nos provee la posibilidad de utilizar interfaz gráfica, esta contará con varios módulos encargados para un mejor manejo y garantizando una vista de cómo se comporta la aplicación en tiempo real, los módulos se muestran a continuación.

Login

Este módulo es el encargado de dar credenciales al usuario, tendrá tres tipos de usuario.

- Normal
- Conductor
- Administrador

Usuario Normal

Este usuario podrá usar la aplicación de manera usual, podrá pedir viajes, revisar sus viajes, pagar viajes, revisar sus pagos, revisar los lugares a cual viajar.

Usuario Conductor

Este usuario únicamente podrá ver los viajes que le toca realizar y visualizar el mapa la ruta.

Usuario Administrador

Este usuario será el encargado de cargar toda la información de los archivos JSON, así también poder **generar reportes de las estructuras (Gráficas con graphviz)**.

El usuario administrador será único, y su usuario será el nombre del auxiliar:

Sección A: **marvin_martinez**

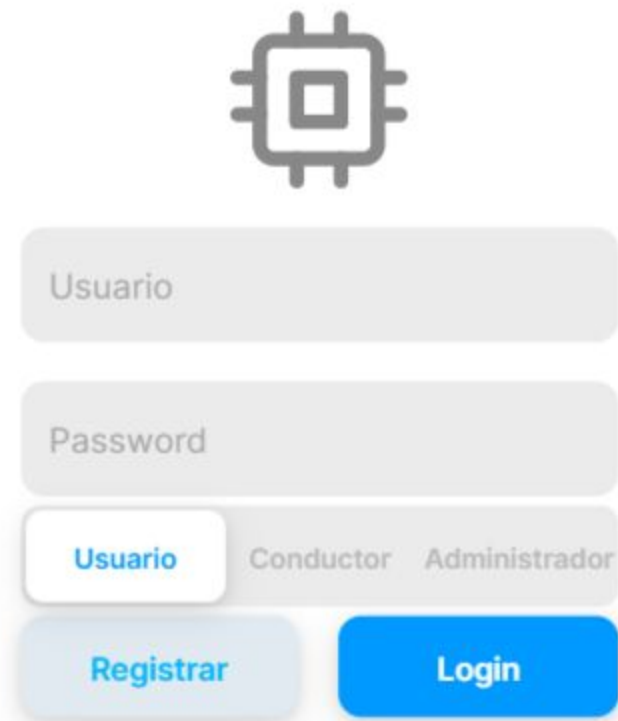
Sección B: **juan_landaverde**

Sección C: **andree_avalos**

con contraseña para todos los usuarios **"admin"**.

Cada nombre de usuario se tendrá que buscar según su rol en su árbol b, normal -> árbol b usuarios, conductor -> árbol b conductor, administrador -> único usuario.

Ejemplo de Interfaz login:



A login interface mockup featuring a central icon of a microchip. Below the icon are two input fields labeled 'Usuario' and 'Password'. Under the 'Usuario' field is a role selection menu with three options: 'Usuario' (highlighted in blue), 'Conductor', and 'Administrador'. At the bottom are two buttons: a light blue 'Registrar' button and a blue 'Login' button.

Todas contarán con una interfaz diferente y no podrán acceder a otra interfaz que no les corresponda.

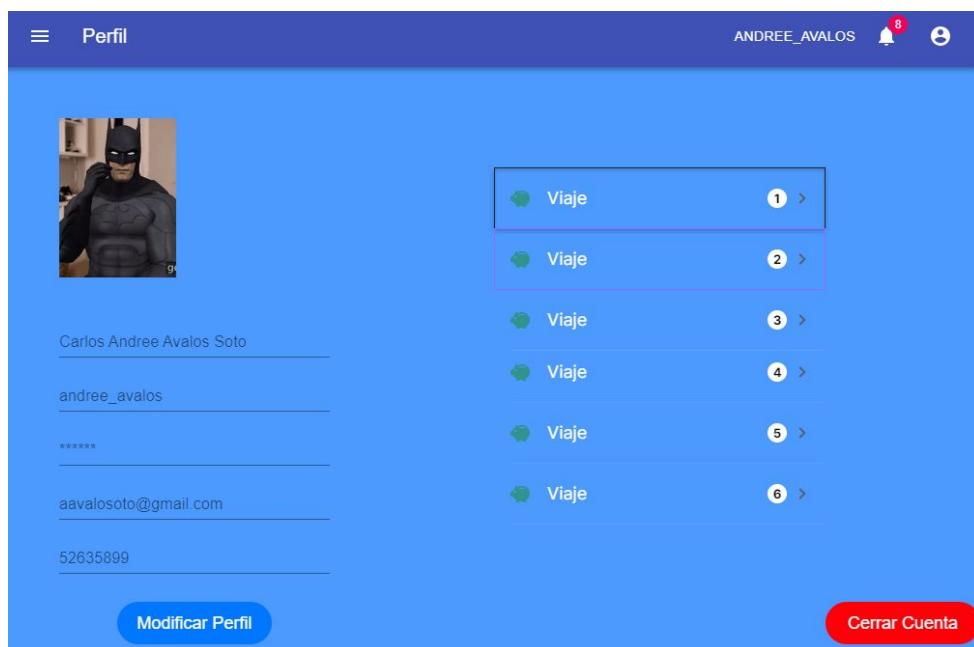
Se podrá crear un usuario normal o conductor con una opción que diga "Registrar"

Usuarios

Este módulo será el encargado de poder gestionar toda la información del usuario, tendrá la opción de visualizar su perfil, modificarlo y cerrar su cuenta.

También desde este módulo podrá visualizar los lugares que están disponibles en la aplicación para viajar y los viajes que ha realizado detalladamente.

Ejemplo de interfaz de perfil:



Para **insertar** se debe generar un **ID** el cual servirá para insertarlo en el **árbol b**, se contará con la siguiente información a guardar:

- id_usuario (**generado automáticamente**)
- nombre
- usuario
- correo
- contraseña (**Encriptado por SHA256**)
- teléfono
- latitud
- longitud

Para la visualización del **árbol B** deberá ser únicamente el **ID**

Para **eliminar** al usuario se debe eliminar por medio de su **ID**.

Notas:

- Los datos pueden ser almacenados dentro del árbol B o pueden ser apuntados con un índice hacia una estructura de las librerías de java (Ej. ArrayList).
- La longitud y latitud debe ser dinámica, para su primera posición debe ser obtenida por su ubicación y cada vez que termine un viaje la ubicación se debe actualizar.

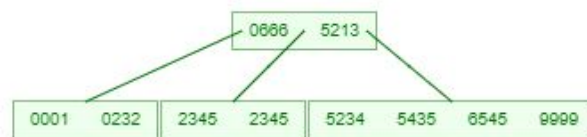
Ejemplo Árbol B:**Ejemplo de Árbol B con Estructura:**

Table	
PK	<u>0</u>
	Nombre
	User
	Pass



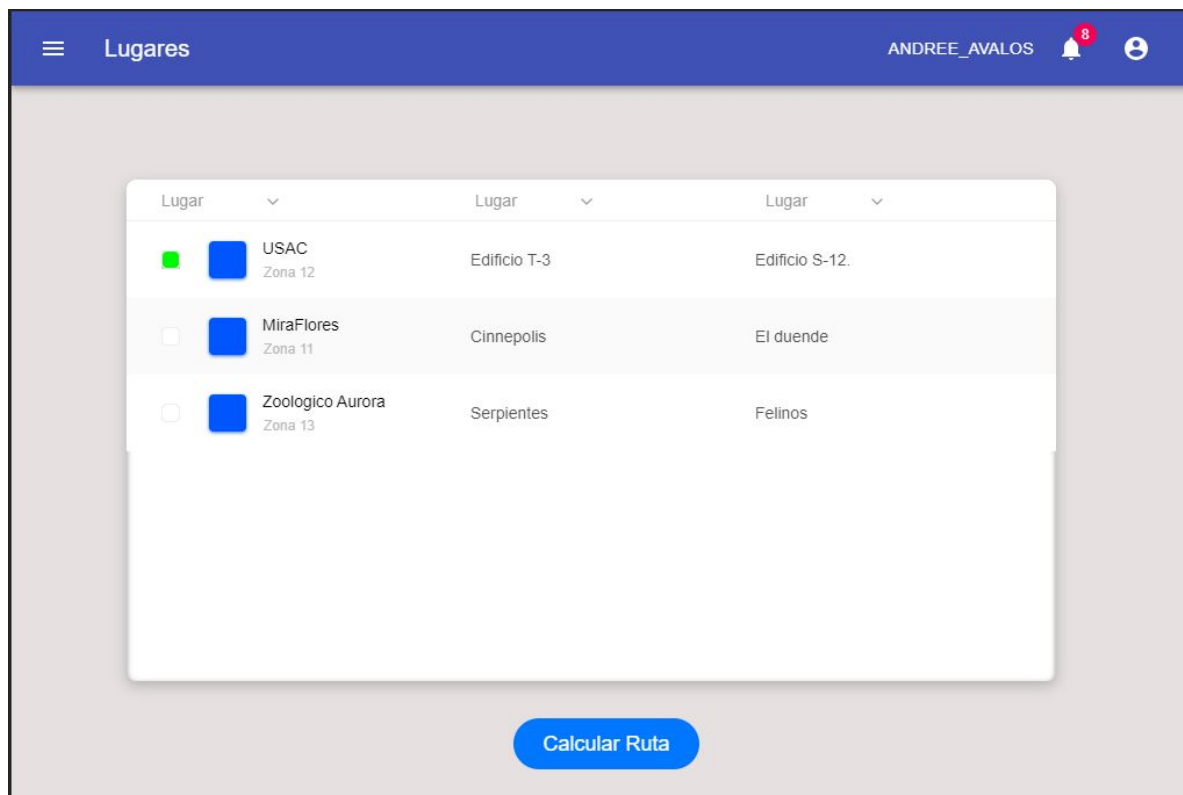
Table	
PK	<u>1</u>
	Nombre
	User
	Pass



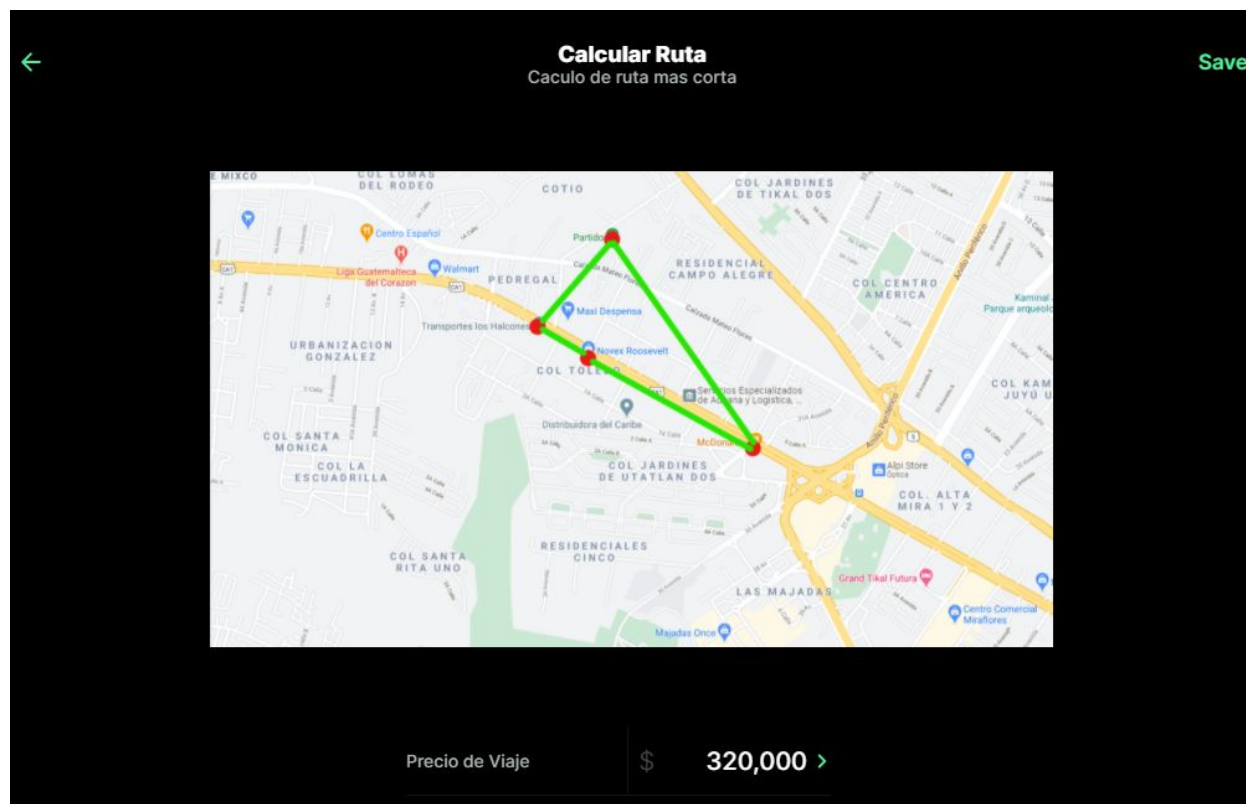
Table	
PK	<u>2</u>
	Nombre
	User
	Pass

La interfaz de los lugares debe mostrar una visualización de la **tabla hash**, esta se mostrará de forma gráfica, el estudiante puede mostrar los nodos principales de la tabla, o de manera completa, queda a elección del estudiante.

Ejemplo de Interfaz de Lugares:



Ejemplo de cálculo de ruta:



En el mapa deberá mostrar todos los puntos que existen con respecto a los lugares, **para iniciar el viaje debe comparar con su latitud y longitud (ingresados manualmente si es un usuario recién creado) con los puntos de los lugares cargados anteriormente para saber cual es el más cercano a su posición y empezar el viaje desde ese punto y a su vez actualizar la posición del usuario**, con esto se tendrá que hacer cálculo del punto de la ruta hacia el destino y dar un precio con respecto a las demás. Esta será emulada por un **grafo** y el estudiante deberá implementar el método para calcularla (ej: dijkstra).

(Sección [Viajes](#))

Transacciones

Como parte de SafeWay, la aplicación tendrá la capacidad de generar bloques cada **5 minutos (default)** con la opción de poder alterar el periodo de guardado, esto con la finalidad de tener una persistencia de datos, para esto el estudiante debe tener una estructura donde se guardará la información detallada de transacciones realizadas (**Ej: Árbol, Lista**) y limpiarse para poder dar paso a nuevas transacciones al generar un bloque. (**Estructura a utilizar para los bloques: [Block Chain](#)**)

Block Chain

En esta estructura de datos se van a almacenar todas las transacciones que se realizan durante el periodo de tiempo. El tiempo inicial es de 5 minutos, después de que pase este tiempo procederá a realizar lo siguiente.

Nuevo bloque

Pasado el tiempo de configuración de la aplicación se genera un nuevo bloque que almacena todo tipo de transacciones, crear un usuario, hacer un viaje, pagar, crear un nuevo lugar, etc. Cada bloque tendrá 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

Son 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.

PreviosHash

Es el hash del bloque anterior, nos permite validar que la cadena de 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 el método de encriptación de *SHA256*, utilizando las propiedades de índice, Fecha, Data, PreviosHash y Nonce. Todos estos campos tienen que ir como cadenas concatenadas sin espacios en blanco.

SHA256(Indice+Fecha+PreviosHash+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. Para esto la aplicación utiliza un algoritmo de **compresión** dado por la librería **Deflater**.

Posteriormente se procederá a **cifrar** los datos, para esto se utilizará un algoritmo **cifrado simétrico de bloques** obtenido de la librería **javax.crypto**.

Notas:

- Los bloques deben guardarse en la carpeta "bloques", de no existir se debe crear.
- El nombre de los bloques debe ser la hora y fecha para evitar que reemplace bloques y se pierda data.
- Si no hay nuevas transacciones en la estructura, entonces no se genera un bloque, esto para evitar que existan bloques vacíos sin información.
- Solo el usuario administrador debe tener acceso a esta interfaz.

Ejemplo:

 12.36.02 21-05-2018	 21/05/2018 12:36	 Documento de tex	 0 KB
---	--	--	--

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 procede a descifrar cada uno de los bloques generados (**Librería: Javax.Crypto**), luego hace la descompresión (**Librería: Inflater**) de los bloques y procede a cargar las estructuras.

Nota: De no encontrar la carpeta de “bloques” debe proceder a generarla.

Cierre de Aplicación

La aplicación debe tener la capacidad que antes de cerrarse, pueda guardar la última instancia de las transacciones. Para esto, cuenta con lo siguientes lineamientos:


Deberá guardar la información de las transacciones (**Ej: [Block Chain](#)**), para esto la aplicación utiliza un algoritmo de **compresión** dado por la librería **Deflater**.

Posteriormente se procederá a **cifrar** los datos, para esto se utilizará un algoritmo **cifrado simétrico de bloques** obtenido de la librería **javax.crypto**.

Notas:

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

Ejemplo:

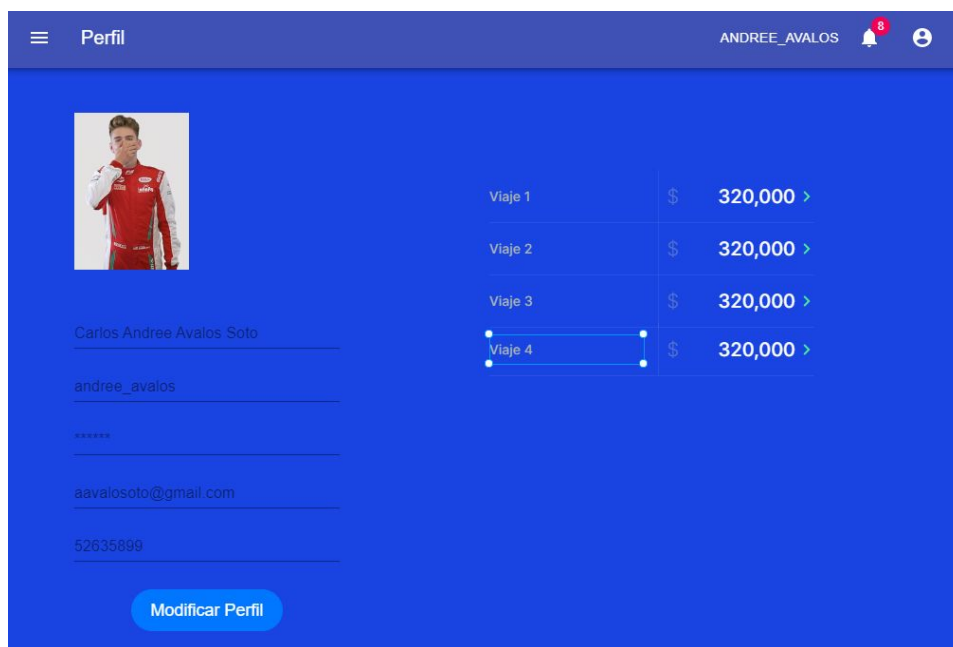
 12.36.02 21-05-2018	 21/05/2018 12:36	 Documento de tex	 0 KB
---	--	--	--

Conductores

Este módulo será el encargado de poder gestionar toda la información del usuario, tendrá la opción de visualizar su perfil, modificarlo y cerrar su cuenta.

También tendrá la posibilidad de visualizar los viajes que ha realizado y el costo.

Ejemplo de Interfaz de conductores:



Para **insertar** se debe generar un **ID** el cual servirá para insertarlo en el **árbol b**, se contará con la siguiente información a guardar:

- id_conductor (**generado automáticamente**)
- nombre
- usuario
- disponibilidad
- contraseña (**Encriptado por SHA256**)
- teléfono
- latitud
- longitud

Para la visualización del **árbol B** deberá ser únicamente el **ID**.

Para **eliminar** debe ser por medio de su **ID**.

Notas:

- Los datos pueden ser almacenados dentro del árbol B o pueden ser apuntados con un índice hacia una estructura de las librerías de java (Ej. ArrayList).
- La latitud y longitud serán dinámicos y serán cargados por aparte y tomarán la latitud y longitud del id de los lugares.
- La disponibilidad es un estado dinámico que cambia cuando se está en un viaje.

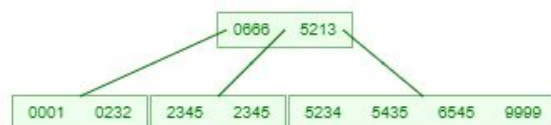
Ejemplo Árbol B:**Ejemplo de Árbol B con Estructura:**

Table	
PK	<u>0</u>
	Nombre
	User
	Pass



Table	
PK	<u>1</u>
	Nombre
	User
	Pass



Table	
PK	<u>2</u>
	Nombre
	User
	Pass

Viajes

Este módulo será el encargado de gestionar los viajes requeridos por los usuarios.

El caso de uso inicia cuando el usuario ingresa a su pantalla y solicita un viaje de cierto punto A al punto B, los puntos estarán ingresados en un combo box, de los que existen en la aplicación

Nota: El usuario no puede ingresar un punto nuevo solo puede utilizar los definidos inicialmente por medio del json de carga de lugares.

En ese momento el sistema debe estimar que conductor se encuentra más cercano al punto inicial del viaje (esto lo debe calcular mediante el método de grafo que el estudiante haya seleccionado para hacer el cálculo de pago) y mostrarle al usuario quien será su conductor.

Notas:

- **El sistema debe de tener en todo momento registrado la posición actual (latitud y longitud), tanto de los conductores como de los usuarios.**
- **La disponibilidad del conductor debe cambiar al tomar un viaje.**
- **Si el conductor está ocupado debe seguir buscando el más cerca y libre**

Cuando el viaje sea aceptado por el conductor el sistema debe de calcular el monto del mismo mediante el método del grafo ya descrito anteriormente.

Para **insertar** se debe generar un **ID** el cual servirá para registrarlo en el **árbol b**, se contará con la siguiente información a guardar:

- **id_viaje (generado automáticamente)**
- **nombre inicio (hash calculado)**
- **nombre fin (hash calculado)**
- **fecha**

Para la visualización del **árbol B** deberá ser únicamente el **ID**.

Para **eliminar** debe ser por medio de su **ID**.

Nota: Los datos pueden ser almacenados dentro del árbol B o pueden ser apuntados con un índice hacia una estructura de las librerías de java (Ej. ArrayList).

Ejemplo de interfaz de viaje:

The screenshot shows a web interface for managing travel records. At the top, there is a header bar with a hamburger menu icon, the title 'Viajes', the user name 'ANDRE', a notification bell icon with a red '8' badge, and a profile icon. The main content area has a blue background and contains four rows of form fields, each with a label on the left, a right-pointing arrow, and a dropdown menu on the right. The fields are: 'Usuario' with 'Juan Lopez', 'Conductor' with 'Conductor 1', 'Punto de origen' with 'Edificio T3', and 'Punto destino' with 'Miafines'. At the bottom of the form is a blue button labeled 'Guardar'.

Facturas

Para este módulo se tendrá que registrar todas las transacciones que existen en la aplicación.

Para **insertar** se debe generar un **ID** el cual servirá para insertarlo en el **árbol b**, se contará con la siguiente información a guardar:

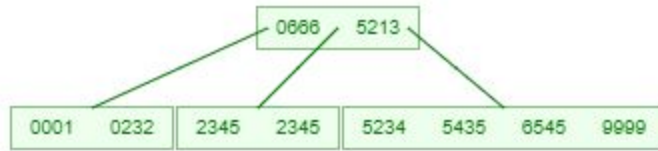
- id_pago (**generado automáticamente**)
- id_usuario (**Árbol B usuarios**)
- id_conductor (**Árbol B conductor**)
- id_viaje (**Árbol B viaje**)
- fecha
- Monto de viaje

No se puede eliminar ninguna transacción de este árbol.

El árbol b de pago será únicamente visualizado con el usuario administrador y será generado con graphviz.

Nota: Los datos pueden ser almacenados dentro del árbol B o pueden ser apuntados con un índice hacia una estructura de las librerías de java (Ej. ArrayList)

Ejemplo Árbol B:



Ejemplo de Árbol B con Estructura:

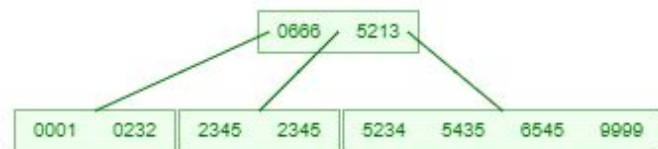


Table	
PK	<u>0</u>
	Nombre
	User
	Pass



Table	
PK	<u>1</u>
	Nombre
	User
	Pass



Table	
PK	<u>2</u>
	Nombre
	User
	Pass

Ejemplo de interfaz de pago:

← **Pago**

Usuario 1

Conductor 1

21/09/2020

Precio de Viaje \$ 320,000 >

Pagar

Catálogo de Lugares

Para este módulo se debe contar con una **tabla hash** la cual debe tener la capacidad poder guardar los lugares clave, a su misma puede guardar los lugares internos como por ejemplo:

- USAC
 - S-12
 - T-3
 - T-4
 - M-3
- Miraflores
 - Cinépolis
 - Zara
 - El museo
 - El duende
 - Max

La tabla hash va a ser con **direccionamiento abierto** y para la resolución de colisiones se va a utilizar una exploración cuadrática.

Se va a utilizar la clave como parámetro de la función hash para obtener la posición a la que corresponde en la estructura. Para el cálculo del hash se debe utilizar el nombre de la categoría y obtener el ascii.

Ejemplo:

Miraflores->1044

Luego de obtener el ascii se debe calcular la posición de nuestra tabla hash, para eso se va a hacer uso del **método de multiplicación**, el cual nos retorna la posición a la que pertenece el elemento.

```
1.  $R * x$ 
2.  $d = R * x - \text{ParteEntera}(R * x)$ 
3.  $h(x) = \text{ParteEntera}(m * d)$ 
```

Para hacer uso de este método la constante va a ser el número de su carnet. Ejemplo:

0.carnet -----> R=0.201602520

Ejemplo:

Miraflores-->1044

$x=1044$

$R=0.201602520$

m =tamaño de la tabla (inicialmente es 10)

1. $0.201602520 * 1044$ ----->210.4730309
2. $d=210.4730309-210$ -----> $d=0.4730309$
3. $h(1044)= 0.4730309 * 10$
4. $h(1044)=4$

Si en mi tabla hash ya está un elemento en esa posición se procede a realizar el método cuadrático.

Nota:

para que no se realice ningún elemento fuera del tamaño de la tabla se va a volver a hacer el hash con el nuevo valor.

$h(1044)=4$

$p = p + i * i$

En donde p es la posición que nos retorna el primer hash. La variable `i` inicia en 1 y se irá incrementando hasta que encuentre una posición vacía.

p=4

i=1

p=4+1*1----->5

x=5

m=10

R=0.201602520

1. $0.201602520 * 5 \rightarrow 1.0080126$
2. $d = 1.008 - 1 \rightarrow d = 0.0080126$
3. $h(5) = 0.0080126 * 10$
4. $h(5) = 0$

¿Está ocupado?

Si

Se incrementa i en uno y se vuelve a calcular el nuevo valor de p y el nuevo valor del hash. Resumiendo quedaría así la ubicación para un nuevo elemento.

```

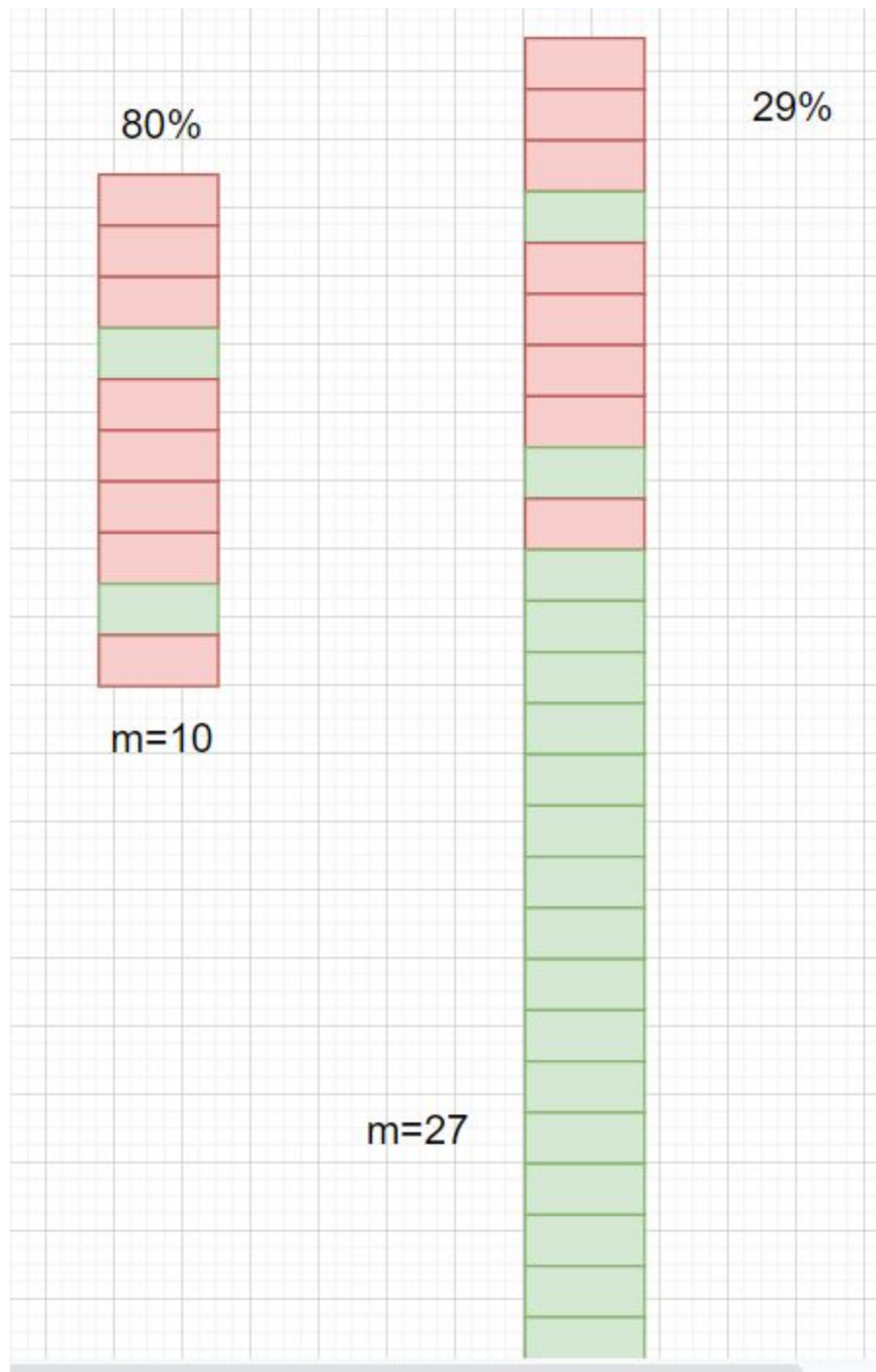
int direccion(String clave){
    int i=0,p=0;
    long d=asccii(clave);
    p=calcularMulltiplicativo(d);
    while(tabla[p]!=null){
        i++;
        p=p+i*i;
        p=calcularMulltiplicativo(p);
    }
    return p;
}

```

No

Esa es la nueva posición para ese elemento.

Para optimizar el espacio de la aplicación la tabla hash se va a manejar con un porcentaje de llenado y un tamaño inicial. El tamaño de la tabla inicialmente va a ser de 10 lugares, si se llega a ocupar el 75% o más de su tamaño la estructura procede a redimensionarse de tal manera que crece hasta que su contenido abarque menor o igual a 30% de su tamaño actual.



Cada nodo hash debe tener un atributo llamado latitud, y longitud que servirán para marcarlos en el mapa. Un ejemplo de los nodos:

- id_lugar
- nombre
- latitud
- longitud

Nota:

El metodo de multiplicacion se encuentra en el libro de Estructuras de datos en java-Luis Joyanes

Carga de Archivos

La aplicación debe tener la capacidad de cargar archivos, para esto el administrador debe poder ingresar un archivo en formato JSON, a continuación hacemos mención del **formato** que se utilizara.

Usuarios

Los campos a ingresar en el archivo json de usuarios serán los siguientes:

- id_usuario
- nombre
- usuario
- correo
- password
- teléfono
- rol

Nota: Rol indica si es "Conductor" o "Normal".

Ej: [Archivo Usuarios](#)

Localidades

Los campos a ingresar en el archivo json de localidades son los siguientes:

- id_conductor
- id_lugar
- disponibilidad

Ej: [Archivo Localidad](#)

Lugares

Los campos a ingresar en el archivo json de lugares son los siguientes:

- id_lugar
- categoría
- nombre
- latitud
- longitud

Ej: [Archivo Lugares](#)

Conexiones

Archivo donde se incluirán las conexiones de los lugares junto con su peso.

Los campos a ingresar en el archivo json de lugares son los siguientes:

- inicio (id_lugar)
- final (id_lugar)
- peso
- unidad
- precio
- moneda

Ej: [Archivo Conexiones](#)

Localidades Iniciales de Usuarios Normales

Este archivo contendrá todas las localidades iniciales de los usuarios normales, con el fin de que todos los usuarios tengan un punto inicial.

Los campos a ingresar en el archivo json de localidades son los siguientes:

- id_usuario
- id_lugar

Ej: [Localidades de Usuarios Normales](#)

Nota: Los datos de estos archivos son solo ejemplos del formato del archivo JSON, no son archivos de entrada.

Consideraciones

- I. Lenguaje a utilizar será Java
- II. **El árbol será de orden 5.**
- III. Las estructuras serán realizadas por el estudiante.
- IV. No se permite el uso de librerías para la implementación de la tabla hash.
- V. Ide a utilizar libre
- VI. La entrega será por medio de la plataforma de **UEDI**
- VII. El estudiante debe tener un repositorio privado en github y agregar a su tutor como contribuidor al repositorio del proyecto (Cada tutor les hará llegar su usuario)
- VIII. Será calificado del último commit realizado antes de la fecha de entrega.
- IX. Las copias totales o parciales serán penalizadas con nota de 0 puntos y reportadas ante la escuela de ciencias y sistemas.
- X. Todos los reportes deben ser generados con graphviz.
- XI. **Fecha de entrega: 4 de Noviembre de 2020 antes de las 23:59 horas.**