

Proyecto → Fase 3

SMART CLASS



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

Randolph Muy, Jorge Salazar, Josué Pérez, José Véliz

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Estructuras de Datos



Índice

Índice	1
Visión General	2
Objetivos	2
Especificaciones	3
Front End	3
Login	3
Usuario Administrador	4
Usuario Estudiante	4
Registro	5
Vistas de la aplicación web	6
Apuntes - Tabla Hash	8
Red de Estudio - Grafos	9
Cifrado	11
Cifrado Simétrico	11
Información de Estudiante	12
Árbol de Merkle	12
Blockchain	15
Consideraciones	18



Visión General

A raíz de la situación por la cual atraviesa la sociedad, la digitalización ha sido un total apoyo a casi todas las actividades que se realizan, por lo cual se le solicita a usted como desarrollador, la creación de una aplicación de apoyo a la universidad haciendo uso de los conocimientos y capacidades que ha adquirido hasta el punto en el cual se encuentra en la carrera.

Smart class será una plataforma diseñada para el apoyo a los estudiantes universitarios a organizar de mejor manera las actividades que realizan en su día a día, esto para poder manejar de manera ordenada las mismas, lo cual facilitará la visualización y administración de estas actividades, esta aplicación será capaz de almacenar no solamente las tareas sino también los estudiantes que forman parte de la misma, todos estos datos deben ser analizados y almacenados de manera segura y confiable.

Objetivos

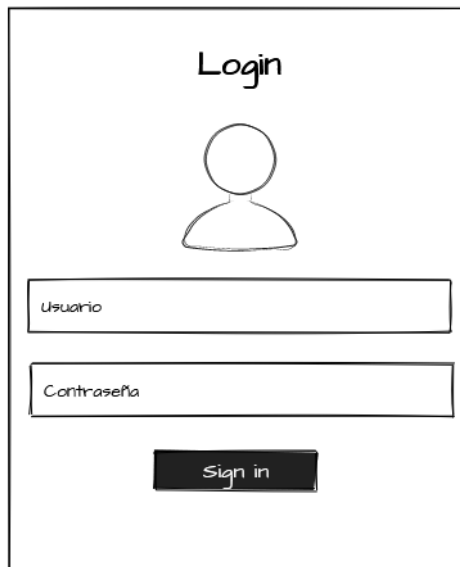
- Que el estudiante se familiarice con los lenguajes de programación Python y Javascript.
- Que el estudiante sepa involucrarse en el ámbito del manejo de la memoria.
- Familiarizarse con el uso de Git.
- Que el estudiante se familiarice con el manejo de lectura de archivos.
- Que el estudiante se familiarice con el uso de frameworks de desarrollo web.
- Comprender el uso de estructuras de datos no lineales.

Especificaciones

En esta fase se solicita crear una aplicación web mediante algún **framework de Javascript**(como por ejemplo **Angular**, ReactJs o VueJs) o en su defecto solamente Javascript, esta aplicación podrá incluir las funciones básicas que se describirán a continuación y además realizará peticiones a un servidor desarrollado en Python para poder obtener la información que mostrará en la aplicación. La implementación de esta aplicación web permitirá generar una mayor facilidad de uso al usuario trasladando todas las peticiones realizadas en la fase 2 a un entorno mucho más amigable.

Front End

Login

A diagram of a login form. At the top, the word "Login" is centered. Below it is a simple line-art icon of a person's head and shoulders. Under the icon are two rectangular input fields. The first field is labeled "Usuario" and the second is labeled "Contraseña". Below these two fields is a dark rectangular button with the text "Sign in" in white.

La plataforma contará con una página de inicio en donde se solicitará al usuario que ingrese sus credenciales para poder acceder a las funcionalidades de la aplicación, hay que tener en cuenta que hay dos tipos de usuarios, los cuales son: **estudiantes y administrador**. Se recomienda que el **usuario administrador esté de manera quemada o de manera default dentro de las configuraciones de la aplicación**, debido a que cada tipo de usuario conlleva una configuración y una serie de vistas distintas. El login contará con un campo de **usuario** (Se ingresa carnet) y **contraseña** para dar ingreso al estudiante, en el caso de que el usuario sea un **administrador**, el usuario y la contraseña son **admin, admin** respectivamente.

Usuario Administrador

Este usuario tiene permisos para gestionar la configuración de la aplicación. Por lo tanto se requieren vistas que den funcionalidad a los siguientes permisos.

- Carga masiva de estudiantes
 - Establecer el password maestro para descriptar y encriptar los datos
- Carga masiva de cursos de pensum
- Carga masiva de cursos del estudiantes
- Carga masiva de los apuntes de estudiantes
- Manejo de los bloques de blockchain
 - Alterar árbol de merkle
 - Corregir árbol de merkle
- Reporte de todos los apuntes de estudiantes
- Reporte de todos los estudiantes registrados
- Reporte del grafo de prerequisites de los cursos del pensum
- Reporte del encriptado del árbol de merkle

Usuario Estudiante

Este usuario es quien consume las estructuras de datos, es decir que haciendo uso de las mismas, administra la información sobre el control de los cursos y tareas. Se solicitan vista que den funcionalidad a los siguientes permisos para este usuario.

- Realizar apuntes
- Asignarse cursos del pensum
- Reporte de todos los cursos previos que deben ser cursados para poder asignarse el curso que el estudiante ingrese por medio del código del curso.
- Reporte de cursos asignados
- Crud de tareas

Registro



The sketch shows a registration form with the title "Registro" at the top center. On the left side, there is a square placeholder for a profile picture, containing a black silhouette of a person's head and shoulders. To the right of the profile picture, there are seven text input fields stacked vertically, each with a label inside: "carnet", "DPI", "nombre", "carrera", "correo", "password", and "edad". At the bottom right of the form, there are two buttons: "Cancelar" and "Registrar".

Esta funcionalidad registra a nuevos estudiantes dentro del sistema, de esta manera el estudiante crea una cuenta para poder ingresar al sistema y poder hacer gestiones de cursos y de tareas.

Nota:

La vista de registro es solo para estudiantes, debido a que el administrador está configurado por defecto, por lo tanto los campos que se solicitan para realizar un registro son los siguientes:

- carnet
- DPI
- nombre
- carrera
- correo
- password
- edad

Vistas de la aplicación web

- Vista creación apunte

Esta vista es para estudiantes y permite a un estudiante registrar un nuevo apunte.

Smart Class

Inicio Apuntes Tareas Red de cursos cerrar sesión

Nombre Usuario

Ver Apuntes Nuevo Apunte

Carnet: 200000000

Titulo: Apunte 1

Contenido de la nota o apunte...

Guardar

- Vista de apuntes

Esta vista permite ver a un estudiante todos sus apuntes y seleccionar al que quiere acceder.

Smart Class

Inicio Apuntes Tareas Red de cursos cerrar sesión

Nombre Usuario

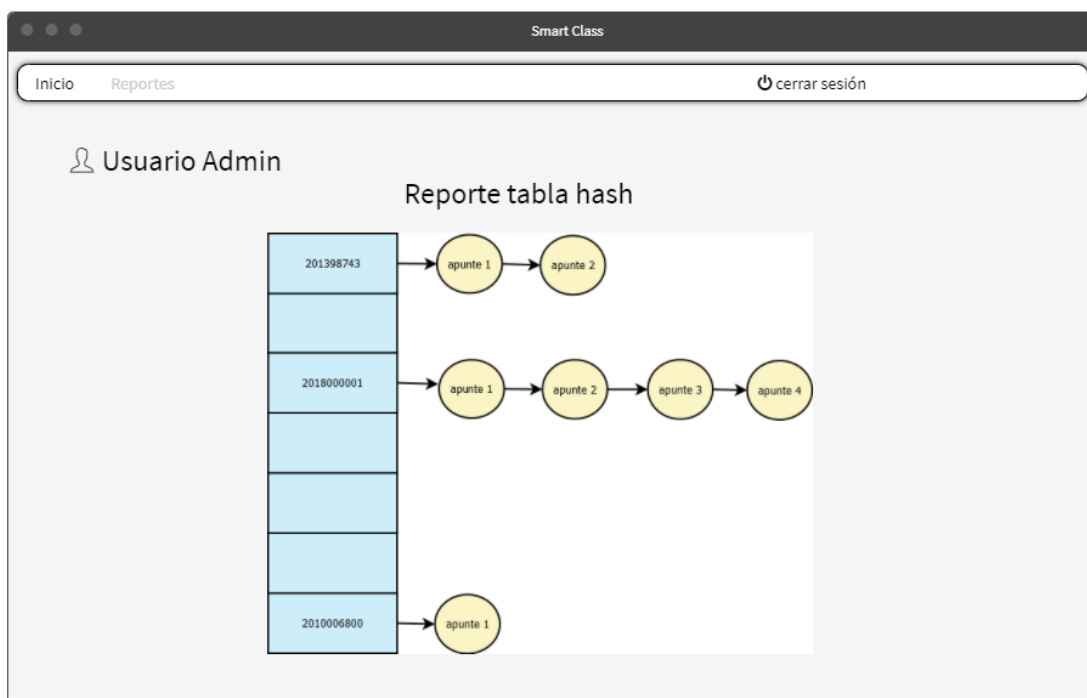
Ver Apuntes Nuevo Apunte

1	Apunte 1	Ver
2	Apunte 2	Ver
3	Apunte 3	Ver

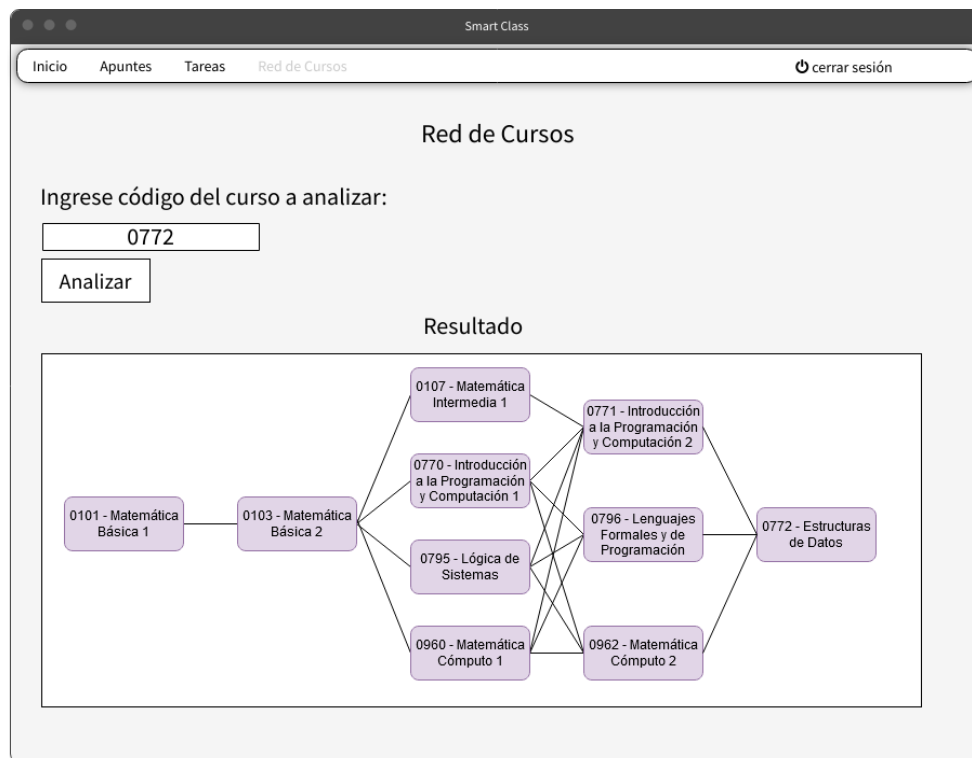


- **Vista Admin tabla Hash**

Solo el Administrador tiene acceso a esta vista, y muestra el reporte de la tabla hash de apuntes.



- **Vista de red de estudio**

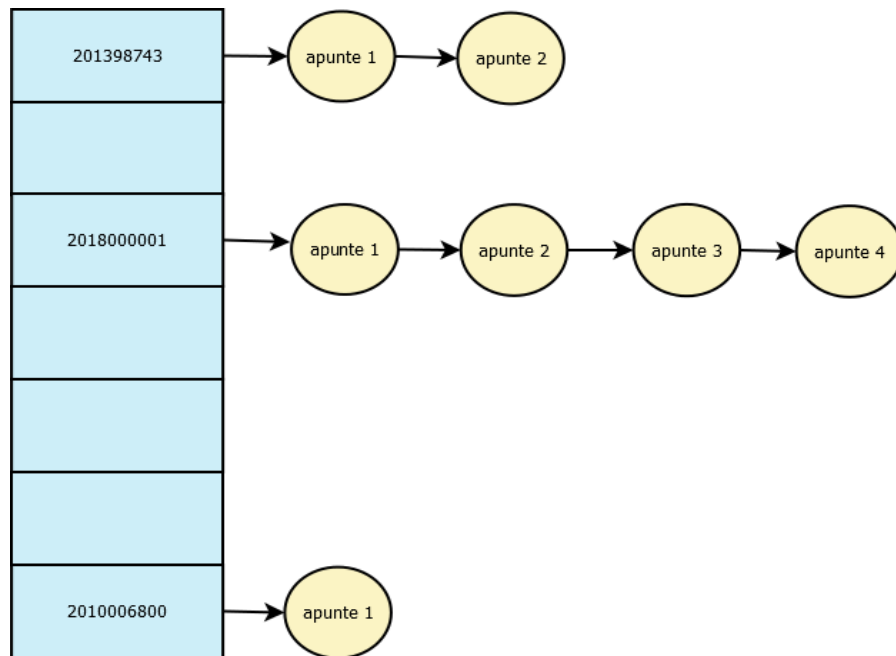


Apuntes - Tabla Hash

En esta fase se implementa una función de apuntes, donde cada estudiante, puede realizar sus apuntes, a diferencia de los recordatorios, estos no necesitan una fecha ni hora.

Estos apuntes se manejan mediante una tabla hash general, donde se guardaran los apuntes de todos los estudiantes. Para almacenar los apuntes en la tabla hash se utilizara el carnet como llave para la función hash.

La tabla hash **almacenará el estudiante que realiza el apunte en una posición, y cada estudiante tendrá una lista de apuntes.** por lo que las colisiones sólo se darán por estudiante y no, por apunte ya que si un estudiante ingresa más de un apunte se almacenará en la lista de ese estudiante.



Para calcular las claves, la función hash a utilizar debe ser el **método de división**. El tipo de hash será cerrado y el método para resolver colisiones debe ser **exploración cuadrática**.

El **tamaño inicial** de la tabla hash será de **7** y cuando se llegue a un **50% de uso**, se tendrá que hacer rehash de la tabla, el tamaño debe crecer al **número primo próximo**.

La información que tiene que tener los apuntes es la siguiente:

- **carnet de estudiante.**
- **Título**
- **Contenido**

Se **recomienda** manejar en los apuntes un id al almacenarlos en la lista que cada nodo estudiante de la tabla hash tendrá, para facilitar el uso de la misma.

Red de Estudio - Grafos

Este tipo de estructura permitirá manejar la red curricular que existe en la carrera de Ingeniería en Ciencias y Sistemas de la Universidad de San Carlos de Guatemala, **en donde los nodos del grafo serán los cursos de la carrera y las conexiones se tomarán por medio de los prerrequisitos que cada uno de los cursos contiene, de tal manera que el grafo resultante será la red curricular de Ingeniería en Ciencias y Sistemas.**

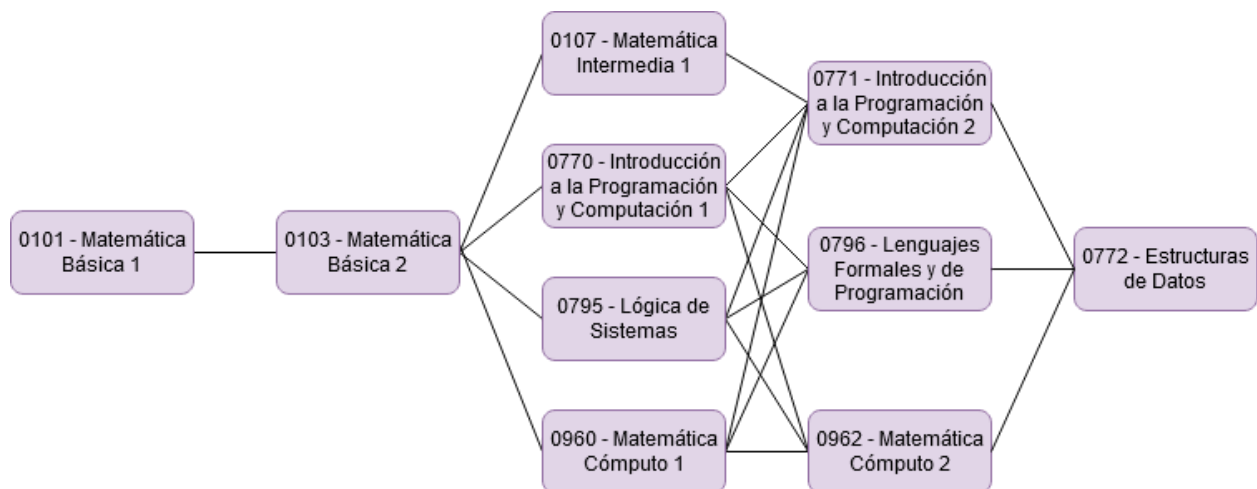
Luego de ya realizado el grafo completo anterior ya se tendrá almacenado en esta **estructura todos los cursos con todas sus relaciones**, siendo así que se pueden realizar consultas sobre dicha estructura, y **a continuación se describe el tipo de consulta que se podrá realizar**.

El **estudiante podrá consultar mediante el ingreso del código del curso deseado, todos los cursos necesarios que debe haber cursado o cursar para poder llevarlo** siendo así que se **generará un gráfico con todos los cursos previos que deben ser cursados para poder llevar el curso deseado**.

Este grafo será un grafo dirigido, es decir, **que los cursos apuntan a los posteriores teniendo las aristas ponderadas, con el número de créditos que el curso de origen da**.



La imagen que se adjunta a continuación permite como tal visualizar el gráfico solicitado anteriormente, mostrando la idea que se desea visualizar.



Cifrado

Para asegurar la información de los estudiantes, es necesario implementar buenas prácticas de seguridad, para ello es necesario cifrar información importante o información sensible de cada estudiante.

Al momento de realizar las acciones de: **cargar de información de estudiantes y registro de estudiantes**, es necesario encriptar **simétricamente** varios atributos sensibles de dicha información, de modo que al momento de alojar la información en la estructura árbol AVL, los datos importantes de cada estudiante se encuentren seguros con dicha encriptación.

Cifrado Simétrico



El cifrado simétrico es también conocido como **cifrado de clave secreta**; para hacer uso del cifrado simétrico es necesario configurar una clave **única para cifrar y descifrar los datos**. Por lo tanto **este tipo de cifrado será aplicado para cifrar información sensible de cada estudiante**, esto al momento de registrar un estudiante a la estructura de árbol AVL.

El proceso de cifrado **se realizará por medio de funciones que brindan distintas librerías, de las cuales se recomienda Fernet utilizando la librería Cryptography en Python**

Hay que tener en cuenta que al momento de cifrar la información, **el password del estudiante debe pasar por una función sha256 antes de ser cifrado, garantizando de esta manera la seguridad de la información.**

Información de Estudiante

El estudiante cuenta con la siguiente información:

- carnet
- DPI
- nombre
- carrera
- correo
- **password**
- créditos
- edad

Al momento de encriptar la información del estudiante, se recomienda encriptar los siguientes datos sensibles: **carnet, DPI, nombre, correo, password, edad**. Con dicha encriptación se asegura que los estudiantes cuenten con cierta seguridad sobre su información personal, esto debido a los posibles ciberataques que se han popularizado en la actualidad.

Por lo tanto se debe utilizar una clave secreta (queda a discreción del estudiante cuál utilizar) para poder encriptar la información de los estudiantes y con dicha clave debe ser capaz de desencriptar la información, ya que esto se verá reflejado al momento de realizar el reporte de estudiantes, en donde se solicita mostrar la información encriptada y desencriptada.

Nota:

Se recomienda utilizar los **primeros 6 caracteres** de la encriptación, para ser mostrados en el reporte, esto debido a que la encriptación de la información es muy extensa, y al momento de mostrarlo como reporte es poco entendible. Hay que tener en cuenta que la muestra de los primeros 6 caracteres no afecta en nada a la encriptación original.

Árbol de Merkle

Debido a que necesitamos tener persistencia en los datos que almacenamos en smartclass, para que estos sean confiables, no solo para los estudiantes sino también para futuras revisiones del proyecto, necesitamos implementar una estructura de datos que pueda mantener nuestra 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 por cada operación que se realiza dentro de las estructuras, a continuación se describen para cuales debe crearse un **árbol de merkle**.

1. Estudiantes
2. Cursos
3. Apuntes

Generar una operación en las estructuras

Para poder insertar en los respectivos árboles, se debe realizar un proceso de asignación, crear un apunte, crear un curso o crear un estudiante.

En relación a esto, se debe simular el proceso de cada una de las anteriores, cuando el proceso sea generado de manera exitosa se debe generar un nodo con los datos que se consideren necesarios para generar un hash que sea único, esto para cada una de las estructuras implementadas.

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 operación realizada en cada uno de los casos, ahora bien para la asignación se deben de tener el **dpi del usuario** y la **fecha de la asignación realizada**.

Ejemplo de nodos hojas

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

$id = 39b923082a194166d8d92989116bd2ed93fd4ba3c68c345d0c6d9140538886bf$

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

id = sha256(“+39b923082a194166d8d92989116bd2ed93fd4ba3c68c345d0c6d9140538886bf
 , 4aa17d5fdf24761b54ad640691146656095ba38e5f92544c73e843db1a2f3875”)

id = 113b685b12dbfa76c04bec7759a24ba66dd6a72c7c6d89159149e56567f87fe4

Ejemplo de salida del árbol



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 bloque 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.

****Las operaciones pueden ser editadas, pero al ser editadas se debe realizar un reemplazo en el hash que tenía asociado al nodo.****



****Para poder volver a un estado correcto solamente un usuario administrador puede realizar este proceso, para esto se debe tener un botón en el módulo de administrador, con esto básicamente se recorrerán los bloques del blockchain (descritos en la siguientes sección) y luego recalcular los hashes asociados al merkle, haciendo que estos concuerden con los hashes de blockchain y se vuelva a un estado correcto de los árboles.****

******Nota:** 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.

Blockchain

En esta estructura de datos se van a guardar todos los procesos que se encuentran en todas las estructuras de datos que ya se hayan 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**

Almacenamiento de información en árboles de Merkle:

Inicialmente se estableció que se debe guardar la información de todas las operaciones realizadas dentro de la plataforma en los árboles de 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 árboles de merkle deben reiniciarse o limpiar todas las estructuras, esto para poder guardar la nueva información que se incluirá en el nuevo bloque.

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 vacíos.

Nuevo bloque:

Pasado el tiempo de configuración en la aplicación se genera un nuevo bloque que almacena las sumalizaciones de los árboles de merkle anteriormente descritos y si no es posible entonces todo tipo de operaciones tales como: realizar una asignación, agregar un apunte, agregar un

curso, agregar un estudiantes, etc., además cada bloque creado debe representar la siguiente estructura:

- **Índice:** el número correlativo del bloque, el bloque inicial tendrá el valor **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 las sumalizaciones de los árboles de Merkle ya creados con anterioridad, si no se encuentran los árboles de merkle entonces deben ser todos los datos que se utilizaron durante el periodo de creación de este bloque, creaciones, eliminaciones, actualizaciones, etc.
- **Nonce:** Es un número entero que se debe iterar de uno en uno hasta encontrar un hash válido, **por defecto** el número de ceros al inicio del hash debe de ser de **4 ceros (0000)** pero este valor debe poder cambiarse desde el panel de administrador.
- **PreviousHash:** Es el hash del bloque anterior, nos permite validar que la cadena del bloque no esté 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 debe hacer uso del algoritmo SHA256, utilizando las propiedades listadas anteriormente. Todos estos bloques deben ir como cadenas concatenadas sin espacios en blanco:

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

Para encontrar un hash válido para nuestro bloque, debemos tomar en cuenta lo mencionado en la prueba de trabajo.

Prueba de trabajo:

Es un sistema con el fin de dificultar el proceso de generación de hashes válidos, para evitar comportamientos indeseados como ataques o spam. Para esto se debe iterar el campo Nonce hasta encontrar un hash válido para el bloque.

Para que el hash cumpla con la condición de que contenga una cantidad de ceros al inicio, por defecto el número de ceros al inicio del hash debe de ser de 4 ceros (0000) pero este valor debe poder cambiarse desde el panel de administrador.

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 en el cual guardará la información necesaria para que al ser leída, la aplicación siga teniendo los mismos datos que tenía al momento de cerrarse o ser interrumpida.

Estos bloques no son legibles para personas que tienen la intención de vulnerar el sistema o manipular los datos.

Configuración de bloques:

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:

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

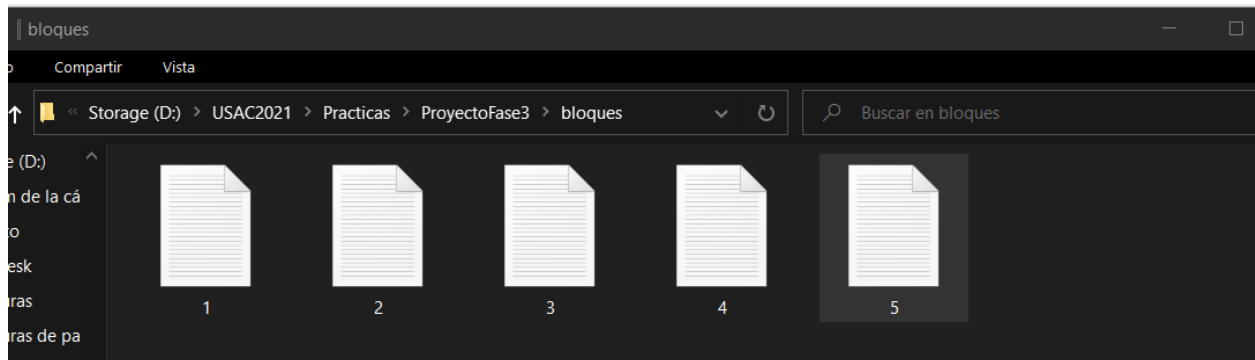
Cierre de la aplicación:

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

Notas:

1. Los bloques se deben guardar en una carpeta llamada “bloques”, de no existir esta carpeta debe de crearse.
2. El nombre de los bloque debe ser manejado por correlativos para evitar que se reemplace los mismos y haya pérdida de información o inconsistencia.
3. Si no hay nuevas transacciones en la estructura, se generará un nuevo bloque pero en el apartado de DATA no tendrá información por lo que se encontrará vacío.
4. Solo el usuario administrador debe tener acceso a esta interfaz.

Ejemplo de carpeta de bloques:



Consideraciones

1. El lenguaje a utilizar será Python.
2. Puerto por defecto **Python: 3000 y Javascript: 7000.**
3. La carpeta de la generación de los reportes debe ser creada en el escritorio, y debe llamarse **Reportes_F3.**
4. Las estructuras serán realizadas por el estudiante.
5. IDE a utilizar es libre (Recomendaciones: **Visual Studio Code, PyCharm, Sublime Text**).
6. La entrega será por medio de la plataforma UEDI.
7. El estudiante debe tener un repositorio privado en github con el nombre **EDD_SmartClass_#Carné** y agregar a su tutor como colaborador al repositorio del proyecto (Cada tutor les hará llegar su estudiante).
8. Será calificado del último commit realizado antes de la fecha y hora de entrega.
9. Las copias totales o parciales **serán penalizadas con nota de 0 puntos y reportadas** ante la escuela de ciencias y sistemas.
10. **Se deberán graficar todas las estructuras de la fase con graphviz.**
11. **Fecha de entrega: 06 de noviembre de 2021 antes de las 23:59 horas.**