****



**“Movie Cats”**

**YONATHAN ALEXANDER HERNÁNDEZ SATZ 201900619**

**GUATEMALA, 29 DICIEMBRE DEL 2022**

**Tabla de contenido**

**Introducción …………………………………………………………………………………………………………………… 3**

**Información destacada …………………………………………………………………………………………….…………. 3**

**Objetivos ………………………………………………………………………………………………………. 3**

**1.Requerimientos ………………………………………………………………………………………………………………. 4**

**2.Instalación de aplicaciones …………………………………………………………………………………..………. 5**

**2.1 Instalación de Visual Studio Code**

**3. Configuración de sistema ………………………………………………………………………………………………. 6**

**4. Estructura raíz ……………………………………………………………………………………………..…………..…… 7**

**4.1.1 App.js**

**4.1.2 Index.html**

**4.1.3 Archivos de recursos .json**

**5. Estructura ……………………………………………………………………………………………………………………... 9**

**5.1.2 Lista Simple**

**5.1.3 Lista de listas**

**5.1.4 Arbol AVL**

**5.1.5 Tabla Hash**

**5.1.6 Árbol binario**

**5.1.7 Blockchain**

**6. Funciones generales ………………………………………………………………………………..……………..…14**

**7. Consideraciones generales …………………………………………………………………………………….... 15**

**8. Flujo de análisis de un Json …………………………………………………………………………………..……… 15**

**Introducción**

El presente documento describe los aspectos técnicos informáticos del sistema de información. El documento introducirá al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

**Información destacada**

El manual técnico hace referencia a la información necesaria con el fin de orientar al personal en la concepción, planteamiento análisis programación e instalación del sistema. Es de notar que la redacción propia del manual técnico está orientada a personal con conocimientos en sistemas y tecnologías de información, conocimientos de programación media/avanzada en programación orientada a estructuras de datos y desarrollo en el entorno del lenguaje de programación **JAVASCRIPT.**

**Objetivos**

Instruir el uso adecuado de **Movie Cats**, para el acceso oportuno y adecuado en la instalación de este, mostrando los pasos a seguir en el proceso de instalación, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración y soporte de este.

1. **Requerimientos**

El sistema pude ser abierto en cualquier sistema operativo que cumpla con los siguientes requerimientos (se recomienda el uso de Windows) :

**Software**

* Microsoft Visual Studio (cualquiera de sus ediciones)

**Hardware**

* Equipo con al menos 2 GB RAM
* Equipo con al menos 4 GB disponible en el disco duro.

1. **Instalación de aplicaciones**

Para implementar de manera correcta **Movie Cats** en un sistema es necesario instalar ciertos programas, los cuales no necesitan de una conexión a internet de forma obligatoria.

* 1. **Instalación Visual Code**

Para su respectiva instalación nos dirigimos en el navegador a :

<https://code.visualstudio.com/>

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La página nos desplegara una vista intuitiva donde se procederá a presionar el botón **AZUL (**Download for Windows)

Icono

Descripción generada automáticamenteSe descargara un archivo con el nombre VSCodeUserSetup , el cual procederemos a abrir e instalar en nuestra disco local.

1. **Configuración del sistema**

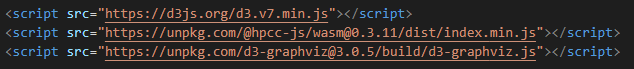
**Movie Cats** está desarrollado bajo JavaScript, esto permite estructurar el sistema de manera rápida lo que se facilita el mantenimiento a dicha solución, a continuación, se describe la estructura básica del sistema y se enfatiza en los archivos, librerías y directorios relevantes para su configuración y adaptación.

1. **Bibliotecas / Recursos**
   1. **D3**

**¿Qué es D3?**

D3 son las siglas de Data-Driven Documents. Es una biblioteca JavaScript de código abierto desarrollada por Mike Bostock para crear visualizaciones de datos interactivas personalizadas en el navegador web utilizando SVG, HTML y CSS.

D3 será utilizado para renderizar el código DOT generado en cada estructura para su posterior visualización. Para la implementación de D3 basta con agregar las siguientes líneas de código en el body del index.



1. **Estructura Raíz**

El proyecto “**Movie Cats**” tiene la siguiente estructura de directorios:

Texto

Descripción generada automáticamente

A continuación, se describirán los directorios y archivos más importantes:

* + 1. **App.js**

Incluye archivos que contienen declaraciones y definiciones de macro que se compartirán entre varios archivos de origen. Se llamarán a los archivos de origen haciendo referencia a su respectivo nombre.

* + 1. **Index.html**

Es la página por defecto dentro de los directorios del sitio web. En este caso el propio servidor web es el que se encarga de buscar el archivo index.html y mostrarlo al visitante (Contiene la interfaz de usuario)

.

* + 1. **Archivos de recursos. json**

Son archivos de prueba, que pueden ser cargados en el sistema para revisar el correcto funcionamiento de cada apartado del usuario/administrador.

**5. Estructura**

* 1. **Archivo de origen**

**5.1.1 Funcionamiento de las estructuras de datos**

**5.1.2 Lista Simple**

|  |  |
| --- | --- |
| **Class Node** | Inicializa un nodo con un constructor, parámetros (dpi,name,username,password,phone,admin)  posteriormente |
| **Class ListaSimple** | Contiene un constructor con una cabeza y un tamaño para el recorrido de la lista. |
| **push** | Inserta un nodo al frente de la lista |
| **append** | Inserta un nodo al final de la lista |
| **registrar** | Obtiene los valores ingresados en el apartado de registrarse para crear un nuevo nodo con esa información. |
| **deleteNodo** | Elimina un nodo seleccionado. |
| **getCount** | Obtiene el tamaño de la lista simple. |
| **buscar** | Busca un nodo en especifico, pasando el valor como parámetro |
| **printList** | Imprime en consola el recorrido de la lista simple |
| **graficarLista** | Obtiene el recorrido de la lista y lo genera en un .dot |
| **mostrarUsuarios** | Obtiene los usuarios de la lista simple |
| **enviarUsuarios** | Envia la información hacia una lista para su posterior uso |

**5.1.3 Lista de listas**

|  |  |
| --- | --- |
| **Class Nodo** | Contiene un constructor con un dato, un siguiente , anterior y una lista simple |
| **Class Lista** | Contiene un constructor con un contador para saber el numero de elementos y un primero haciendo referencia a la cabeza de la segunda lista. |
| **add** | Crea una lista normal . |
| **add2** | Con parámetros nombre,lista : ingresa una nueva a una ya creada. |
| **repetido** | Verifica si en la lista normal existe un nodo repetido. |
| **mostrar** | Imprime en consola el recorrido de las listas |
| **buscar** | Dado un índice busca la información en la lista. |
| **graficarListaDeListas** | Obtiene el recorrido de la lista y lo genera en un .dot |
| **mostrarArtistas** | Obtiene los usuarios de la lista de listas |
| **enviarArtistas** | Envia la información hacia una lista de artistas para su posterior uso |
| **ordenar2** | Ordena la lista de manera ascendente usando el método Quicksort. |

**5.1.4 Arbol Avl**

|  |  |
| --- | --- |
| **Nodo** | Inicializa un nodo con un constructor, parámetro “d”. Los atributos del constructor serán dato, factor de equilibrio, hijoDerecho, hijoIzquierdo |
| **TextGraphviz** | Escribirá las líneas correspondientes al apartado de enlaces de nodo. |
| **obtenerRaiz** | Retorna la raíz del árbol |
| **Buscar** | Busca un nodo en el árbol |
| **ObtenerFE** | Retorna el factor de equilibrio |
| **rotacionIzquierda** | Realiza el calculo para rotar un nodo a la izquierda |
| **rotacionDerecha** | Realiza el cálculo para rotar un nodo a la derecha |
| **rotacionDobleIzquierda** | Realiza el cálculo para rotar un nodo dos veces a la izquierda |
| **rotacionDobleDerecha** | Realiza el cálculo para rotar un nodo dos veces a la derecha |
| **insertAVL** | Ingresa en un nodo un valor |
| **obtenerGraphviz** | Construye los encabezados del texto en graphviz del árbol además de añadirle los enlaces |
| **cargaMasivaPeliculas** | Recorre el archivo Json y los ingresa al árbol |

**5.1.5 Tabla Hash**

|  |  |
| --- | --- |
| **Class nodoHash** | Inicializa un nodo con un constructor, con parámetro dato y atributos, siguiente , anterior, y una lista simple |
| **add** | Método para crear la lista normal |
| **add2** | Método para ingresar un valor dentro de una lista |
| **repetido** | Método para verificar si hay algún dato repetido |
| **mostrar** | Recorre la pila y lo muestra en consola. |
| **buscar** | Busca información dentro de la lista |
| **graficarTabla** | Crea el texto que luego será renderizado en graphviz |
| **insertar** | Ingresa un dato en una posición especifica en la tabla |

**5.1.7 Árbol**

|  |  |
| --- | --- |
| **Class nodoArbol** | Inicializa un nodo con un constructor, parámetros datos,izquierda,derecha |
| **Class Arbol** | Contiene un constructor con una raiz |
| **agregar\_recursivo** | Añade un elemento recorriendo de manera recursiva el árbol. |
| **inorden\_recursivo** | Recorre el árbol para colocar los elementos de manera , inorder. |
| **buscar** | Busca un nodo en el árbol. |
| **agregar** | Agrega un elemento a el recorrido recursivo. |
| **buscar2** | Busca un nodo en especifico en el árbol. |
| **graficarArbol** | Obtiene el recorrido del árbol y lo genera en un .dot |

**5.1.8 Blockchain**

|  |  |
| --- | --- |
| **Class Block** | Contiene un constructor con parámetros index,data,previus hash que servirán para inicializar el bloque Genesis de la cadena. |
| **createHash** | Crea un hash incluyendo el index , la fecha , el previus hash y el nonce. |
| **mine** | Crea un nuevo hash con una nueva dificultad |
| **Class Blockchain** | Contiene un constructor con parámetros: el bloque genesis y la dificultad (00). |
| **createFirtsBlock** | Crea el bloque genesis de la cadena. |
| **getLastBlock** | Retorna el ultimo bloque de la cadena. |
| **addBlock** | Agrega un nuevo bloque a la cadena. |
| **isValid** | Valida si los hashes de los bloques coinciden. |

**6. Funciones generales**

|  |  |
| --- | --- |
| **login** | Valida las credenciales de un usuario. |
| **logout** | Cierra la sesión devolviendo el estado de la web a div iniciales. |
| **CargaMasivaUsuarios** | Carga al sistema usuarios por medio de un archivo json. |
| **CargaMasivaPeliculas** | Carga al sistema peliculas por medio de un archivo json. |
| **CargaMasivaActores** | Carga al sistema actores por medio de un archivo json. |
| **CargaMasivaCetegorias** | Carga al sistema categorias por medio de un archivo json. |
| **cambiarValoracion** | Agrega a la playlist la opción seleccionada en el select. |
| **crearComentario** | Crea un comentario con los valores ingresados en el collapse de peliculas |
| **crearTabla** | Crea la tabla de comentarios |
| **mostrarTabla** | Muestra la tabla de comentarios por pelicula |
| **showDivIniciales** | Oculta los divs de usuario y administrador |
| **showDivAdministrador** | Oculta las funciones que no sean de administrador |
| **showDivUsuario** | Oculta las funciones que no sean de usuario |
| **limpiarInicioDeSesion** | Limpia el form del login |
| **limpiarFormularios** | Limpia formularios después de su uso. |
| **showDivIniciales** | Muestra los divs iniciales (ingreso a la web). |
| **capturarPeliculas** | Obtiene el contenido del div de películas . |
| **capturarClientes** | Obtiene el contenido del div de clientes. |
| **capturarActores** | Obtiene el contenido del div de actores. |
| **capturarCategorias** | Obtiene el contenido del div de actores. |
| **dowload\_image** | Obtiene el contenido del canvas de películas y lo convierte en PNG para su respectiva descarga. |
| **dowload\_image2** | Obtiene el contenido del canvas de clientes y lo convierte en PNG para su respectiva descarga. |
| **dowload\_image3** | Obtiene el contenido del canvas de actores y lo convierte en PNG para su respectiva descarga. |
| **dowload\_image** | Obtiene el contenido del canvas de categorias y lo convierte en PNG para su respectiva descarga. |

1. **Consideraciones de clases, directorios y archivos:**

Todos los archivos y directorios no mencionados son parte importante para el funcionamiento del sistema, no se hacen referencia en este documento debido a que solo se enfatizan los archivos que el usuario puede en un dado caso modificar, con conocimiento previo de lo que se hace.

* No se puede analizar un archivo sin antes cargarlo al sistema.
* Es importante fijarse en el formato de los archivos de prueba. json del sistema, ya que ese formato será el aceptado en el sistema.
* El sistema no cuenta con persistencia de datos. El recargar la página hará que pierda su información.
* Las estructuras de datos se comparten entre todos los usuarios.

1. **Flujo del análisis de Json**

|  |
| --- |
| **Nombre: Análisis de un archivo.** |
| **Actor(es):** Administrador |
| **Descripción:** Se analiza un archivo. json cargado al sistema en busca de diferente información encapsulada por partes |
| **Flujo Normal:**   1. El actor ingresa a la web y elige la opción de carga. 2. Se carga la información y se añade a sus respectivas estructuras. 3. Se analiza la información para ser procesada. 4. Se genera en la opción de reportes su visualización. |
| **Flujo alternativo:**   1. Error en la lectura del archivo |
| **Post condición:**   1. Archivos gráficos/información solicitada. |