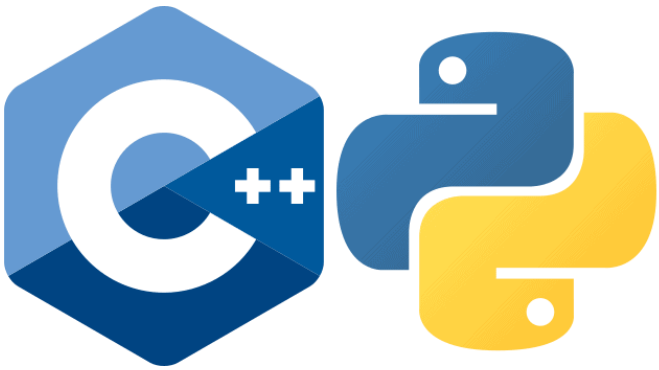
Laboratorio de Estructuras de Datos



PROYECTO EDD – FASE 3

MANUAL TECNICO

FECHA: 30/10/2022

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación, deseñada a través de código de estructura de datos con paradigma orientado a objetos. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

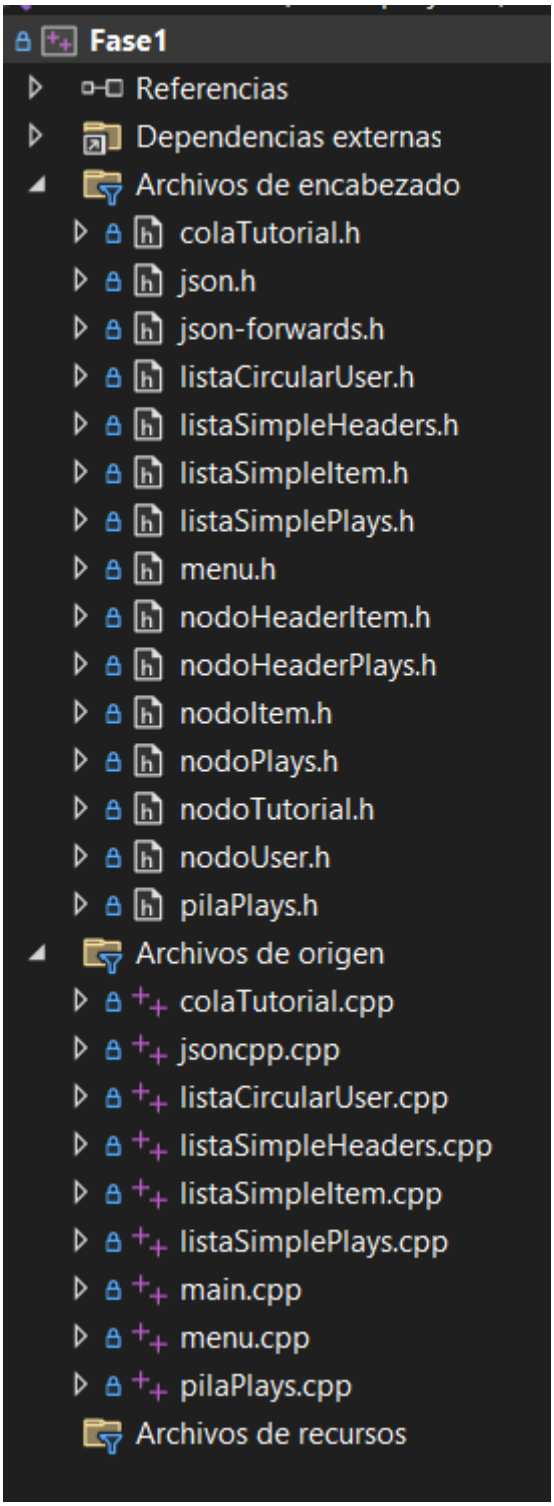
Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:



DESCRIPSION DE LOS METODOS Y CLASES:

Los métodos y clases utilizados para este programa fueron pensados y diseñados específicamente para el desarrollo e implementación de este, siguiendo rigurosamente los requerimientos de diseño inicialmente propuestos en el manual de requerimientos.

La clase main:

Contiene el método principal de nuestra aplicación, esta clase se encarga de inicializar los objetos que usaremos dentro del programa a lo largo de toda la ejecución. Ayudando así a la persistencia de datos en cualquier área del programa.

Clase Menu:

Tiene como función generar el entorno sobre el cual estará situado el usuario, este entorno se desplegará en consola permitiendo una interfaz gráfica primitiva pero intuitiva. Dentro de la clase podremos encontrar, el menú principal, el menú secundario, y el menú de reportes.

Clase listaCircularUser:

Esta clase tiene como función de crear una lista doblemente enlazada, permitiendo al usuario escoger por medio del menú los siguiente métodos:

Carga masiva de usuarios por medio de “archivo.json”:

```
void listaCircularUser::loadFile(string ruta) {

    nodoUser* nuevo;
    ifstream ifs(ruta);
    Json::Reader reader;
    Json::Value obj;
    reader.parse(ifs, obj);
    const Json::Value& dataUser = obj["usuarios"];

    for (int i = 0; i < dataUser.size(); i++) {

        nuevo = new nodoUser();

        string coins = dataUser[i]["monedas"].asString();
        string age = dataUser[i]["edad"].asString();

        nuevo->setNick(dataUser[i]["nick"].asString());
        nuevo->setPassword(dataUser[i]["password"].asString());
        nuevo->setCoins(stoi(coins));
        nuevo->setAge(stoi(age));

        addToEnd(nuevo);
    }
}
```

En este método se usa la librería Jsoncpp por medio de la cual se recorre el archivo de entrada y por medio de un ciclo for se almacenan los datos en la clase nodoUser que tiene los atributos del usuario a guardar.

Añadir al final:

```
void listaCircularUser::addToEnd(nodoUser* user) {

    nodoUser* nuevo = new nodoUser();
    nuevo = user;

    if (primero == NULL) {
        primero = nuevo;
        ultimo = nuevo;
        primero->siguiente = primero;
        primero->atras = ultimo;
    }
    else {
        ultimo->siguiente = nuevo;
        nuevo->atras = ultimo;
        nuevo->siguiente = primero;
        ultimo = nuevo;
        primero->atras = ultimo;
    }
    cout << "\n>Usuario Registrado!\n\n";
}
```

En este método se utiliza la lógica de la programación orientada a objetos y la implementación de estructuras de datos por medio de lo cual ingresamos nodos a una lista que es capaz de señalar los nodos entre si y de esta forma permitir un mejor maneo de memoria.

Método de Loguin o inicio de sesión:

```
void listaCircularUser::login(listaCircularUser listaUsers,listaSimpleItem listIte
    nodoUser* actual = new nodoUser();
    actual = primero;
    bool encontrado = false;
    string aux;
    if (primero == NULL) {
        cout << "No hay usuarios registrados!\n\n";
    }
    else {
        cout << " > Ingrese el nombre del usuario: ";
        cin >> aux;
        do {
            if (actual->getNick() == aux) {
                cout << ">Usuario Encontrado!\n\n";
                encontrado = true;

                string pass;
                cout << " > Ingrese su Password: ";
                cin >> pass;

                if (pass == actual->getPassword() && aux == actual->getNick()) {
                    menu secMenu;
                    secMenu.sec_menu(listaUsers,actual,listItems,queue,stack);

                }
                else {
                    cout << ">password incorrecto!\n";
                    system("pause");
                }
            }
            actual = actual->siguiente;
        } while (actual != primero && encontrado != true);

        if (!encontrado) {
            cout << "\n>usuario no Encontrado\n\n";
        }
```

En este método se busca al objeto usuario en la lista por medio de su nombre y al encontrarlo se le permite ingresar su contraseña si esta coincide con la almacenada en el objeto entonces se procede a enviarlo al menú secundario de lo contrario se rechaza la acción y se vuelve al menú principal.

Método de editar información:

```
cout << "\n\n1. Modificar Nick" << endl;
cout << "2. Modificar Password" << endl;
cout << "3. Modificar Edad" << endl;
cout << "4. Guardar y Salir" << endl;

cout << "\nIngrese una opcion: " << endl;
cin >> opcion;

switch (opcion) {
case '1':
    cout << "Ingresa el nuevo Nick: ";
    cin >> aux;
    actual->setNick(aux);
    system("pause");
    break;

case '2':
    cout << "Ingresa el nuevo Password: ";
    cin >> aux;
    actual->setPassword(aux);
    system("pause");
    break;

case '3':
    cout << "Ingresa tu nueva Edad: ";
    cin >> aux;
    actual->setAge(aux2);
    system("pause");
    break;
```

En este método se le pide al usuario que ingrese a una opción por medio de un switch case y de esta forma permitimos elegir qué información desea modificar, luego por medio de la programación orientada a objetos asignamos al nodo actual los atributos que el usuario digita, de esta forma logramos renovar los datos viejos con los actuales.

Método de eliminar un usuario:

```
// si si ha confirmado se procede a eliminar
if (confirm == true && primero != NULL) {
    do {
        if (actual->getNick() == aux && actual->getPassword() == aux2) {

            if (actual == primero) {
                primero = primero->siguiente;
                primero->atras = ultimo;
                ultimo->siguiente = primero;
            }
            else if (actual == ultimo) {
                ultimo = anterior;
                ultimo->siguiente = primero;
                primero->atras = ultimo;
            }
            else {
                anterior->siguiente = actual->siguiente;
                actual->siguiente->atras = anterior;
            }
            cout << "\n>Usuario Eliminado!\n\n";

        }
        anterior = actual;
        actual = actual->siguiente;
    } while (actual != primero && encontrado != true);

    encontrado = true;
    return true;
```

En este método lo que se busca es que un nodo desaparezca de la estructura por lo cual, al momento de eliminarlo, hacemos que tanto el nodo anterior como el que le sigue al nodo actual, dejen de apuntar a el, y asignamos nuevas direcciones para los nodos anterior y siguiente enlazando estos entre sí, de esta forma el nodo actual queda sin apuntadores que lo referencien por lo cual queda sin propiedad de una clase y es eliminado de la memoria.

listaSimpleHeaders y listaSimpleItems:

Estas 2 clases en conjunto tiene la función de generar una lista de listas, para lo cual la listaSimpleHeader toma el papel de la lista de Encabezados, y la listaSimpleItems toma el papel del listado de ítems que pertenecen a cada encabezado. Para esto su utilizaron 2 metodos, el método de Insertar y el método de BuscarPrincipal, los cuales se describen seguido:

Método de inserción:

```
void listaSimpleHeaders::Insertar(nodoItem* item, string categoria) {
    if (primero == NULL) {
        nodoHeaderItem* nuevo = new nodoHeaderItem();
        nuevo->categoria = categoria;
        nuevo->inList.addToEnd(item);
        primero = nuevo;
    }
    else {
        nodoHeaderItem* busqueda = BuscarPrincipal(primero, categoria);
        nodoHeaderItem* nuevo = new nodoHeaderItem();
        nuevo->setCat(categoria);
        nuevo->inList.addToEnd(item);
        if (busqueda == NULL) {
            nodoHeaderItem* auxActual = primero;
            while (auxActual != NULL) {
                if (auxActual->siguiente == NULL) {
                    auxActual->siguiente = nuevo;
                    break;
                }
                auxActual = auxActual->siguiente;
            }
        }
        else {
            busqueda->inList.addToEnd(item);
        }
    }
}
```

En este método lo que se realiza es una inserción de un nodo en una lista, este nodo se insertara en la lista que contenga el nombre de su categoría, para lo cual se ingresa el nodo y su categoría, las listas de cabeceras se crean con el nombre de la categoría que entra, si la categoría existe a dicha categoría se le añade una lista de objetos y se asigna ahí el nodo, si la categoría no existe se crea una nueva con el nombre de la categoría entrante y luego se añade una lista y se asigna a esa lista el nodo.

Meodo de búsqueda de cabeceras:

```
nodoHeaderItem* listaSimpleHeaders::BuscarPrincipal(nodoHeaderItem* primeroL, string categoria) {
    if (primeroL == NULL) {
        return primeroL;
    }
    else {
        nodoHeaderItem* auxActual = primeroL;
        while (auxActual != NULL) {
            if (auxActual->getCat() == categoria) {
                break;
            }
            auxActual = auxActual->siguiente;
        }
        return auxActual;
    }
}
```

Este método se encargar de recorrer la lista de cabeceras y verificar si existe algún nodo cabecero con el nombre de la categoría entrante, si existe retorna ese nodo, si no existe, crea un nodo con el nombre de la categoría entrante y luego retorna ese nodo.

Carga masiva de datos:

En este método se usa la librería Jsoncpp por medio de la cual se recorre el archivo de entrada y por medio de un ciclo for se almacenan los datos en la clase nodoItem que tiene los atributos del usuario a guardar.

```
nodoItem* nuevo;
ifstream ifs(ruta);
Json::Reader reader;
Json::Value obj;
reader.parse(ifs, obj);
const Json::Value& dataArticle = obj["articulos"];
for (int i = 0; i < dataArticle.size(); i++) {

    nuevo = new nodoItem();

    string price = dataArticle[i]["precio"].asString();

    nuevo->setId(dataArticle[i]["id"].asString());
    nuevo->setCategory(dataArticle[i]["categoria"].asString());
    nuevo->setPrice(stoi(price));
    nuevo->setName(dataArticle[i]["nombre"].asString());
    nuevo->setSrc(dataArticle[i]["src"].asString());

    Insertar(nuevo, dataArticle[i]["categoria"].asString());
}
```

listaSimplePlays y pilaPlays:

Estas 2 clases en conjunto tiene la función de generar una lista de pilas, para lo cual la listaSimplePlays toma el papel de la lista de Encabezados, y la pilaPlays toma el papel del listado de jugadas que pertenecen a cada encabezado. Para esto su utilizaron 2 metodos, el método de Insertar y el método de BuscarPrincipal, los cuales se describen seguido:

```
void listaSimplePlays::Insertar(nodoPlays* plays, string categoria) {
    if (primero == NULL) {
        nodoHeaderPlays* nuevo = new nodoHeaderPlays();
        nuevo->categoria = categoria;
        nuevo->inStack.addToEnd(plays);
        primero = nuevo;
    }
    else {
        nodoHeaderPlays* busqueda = BuscarPrincipal(primero, categoria);
        nodoHeaderPlays* nuevo = new nodoHeaderPlays();
        nuevo->setCat(categoria);
        nuevo->inStack.addToEnd(plays);
        if (busqueda == NULL) {
            nodoHeaderPlays* auxActual = primero;
            while (auxActual != NULL) {
                if (auxActual->siguiente == NULL) {
                    auxActual->siguiente = nuevo;
                    break;
                }
                auxActual = auxActual->siguiente;
            }
        }
        else {
            busqueda->inStack.addToEnd(plays);
        }
    }
}
```

En este método lo que se realiza es una inserción de un nodo en una lista, este nodo se insertara en la lista que contenga el nombre de su categoría, para lo cual se ingresa el nodo y su categoría, las listas de cabeceras se crean con el nombre de la categoría que entra, si la categoría existe a dicha categoría se le añade una lista de objetos y se asigna ahí el nodo, si la categoría no existe se crea una nueva con el nombre de la categoría entrante y luego se añade una lista y se asigna a esa lista el nodo.

Meodo de búsqueda de cabeceras:

```
nodoHeaderPlays* listaSimplePlays::BuscarPrincipal(nodoHeaderPlays* primeroL, string categoria) {
    if (primeroL == NULL) {
        return primeroL;
    }
    else {
        nodoHeaderPlays* auxActual = primeroL;
        while (auxActual != NULL) {
            if (auxActual->getCat() == categoria) {
                break;
            }
            auxActual = auxActual->siguiente;
        }
        return auxActual;
    }
}
```

Este método se encargar de recorrer la lista de cabeceras y verificar si existe algún nodo cabecero con el nombre de la categoría entrante, si existe retorna ese nodo, si no existe, crea un nodo con el nombre de la categoría entrante y luego retorna ese nodo.

El método jugar o doGameplays:

```
cout << "\n\n>Realizar Movimientos" << endl;
cout << "1. Ingresar Jugada" << endl;
cout << "2. Guardar y Salir" << endl;

cout << "\nIngrese una opcion: " << endl;
cin >> opcion;

coins = nuevoUser->getCoins();
int posX = 0;
int posY = 0;

switch (opcion) {

case '1':

    nuevo = new nodoPlays();
    cout << "Coordenada X: ";
    cin >> posX;
    cout << "Coordenada Y: ";
    cin >> posY;
    nuevo->setX(posX);
    nuevo->setY(posY);

    cout << ">Ingreso la coordenada: (" << posX << "," << posY << ")" << endl;

    Insertar(nuevo,nuevoUser->getNick());

    coins += 1;
    nuevoUser->setCoins(coins);

    cout << " > Tokens: +1" << endl;

    system("pause");
    break;
```

Este método de interacción con el usuario genera una pequeña interfaz donde el usuario puede ingresar coordenadas las cuales serán almacenadas en una pila y dicha pila en la categoría perteneciente al usuario que las esta realizando, en este método también se le añaden puntos al usuario lo cuales son traducidos en monedas las cuales pueden ser utilizadas posteriormente en la tienda.

ColaTutorial:

Esta clase se encarga de generar una cola por medio de la clase nodo tutorial el cual es el nodo que se aniade a la cola.

Método de inserción:

```
nodoTutorial* nuevo = new nodoTutorial();
nuevo = tuto;

if (primero == NULL) {
    primero = nuevo;
    primero->siguiente = NULL;
    ultimo = primero;
}
else {
    ultimo->siguiente = nuevo;
    nuevo->siguiente = NULL;
    ultimo = nuevo;
}

cout << "\n>Nodo Ingresado!\n\n";
```

En el método de inserción de una cola únicamente verificamos que este vacía y si lo esta ingresamos el nodo entrante como primero y como ultimo a la vez, de lo contrario el nodo que entra es el nodo siguiente del anterior y es a la vez el último, la estructura es similar a una lista enlazada simple.

Método de carga masiva:

```
nodoTutorial* nuevo;
ifstream ifs(ruta);
Json::Reader reader;
Json::Value obj;
Json::Value attrib;
reader.parse(ifs, obj);
const Json::Value& dataTable = obj["tutorial"];

nuevo = new nodoTutorial();

string Alto = dataTable["alto"].asString();
string Ancho = dataTable["ancho"].asString();

cout << "(" << Alto << " <-> " << Ancho << ")" << endl;

nuevo->setAlto(stoi(Alto));
nuevo->setAncho(stoi(Ancho));

addToEnd(nuevo);

for (int i = 0; i < dataTable["movimientos"].size(); i++) {

    nuevo = new nodoTutorial();

    string posX = dataTable["movimientos"][i]["x"].asString();
    string posY = dataTable["movimientos"][i]["y"].asString();

    cout << "(" << posX << "," << posY << ")" << endl;

    nuevo->setX(stoi(posX));
    nuevo->setY(stoi(posY));

    addToEnd(nuevo);
}
```

En este método se usa la librería Jsoncpp por medio de la cual se recorre el archivo de entrada y por medio de un ciclo for se almacenan los datos en la clase nodoTutorial que tiene los atributos del Tutorial a guardar.

Método general de impresión en consola:

```
void listaCircularUser::showList() {
    nodoUser* actual = new nodoUser();
    actual = primero;

    cout << ">Listado de Usuarios: " << endl;

    if (primero != NULL) {
        do {
            cout << " Nombre: " << actual->getNick() << "\tPassword: " << actual->getPassword() << \
                "\tAge: " << actual->getAge() << "\tCoins: " << actual->getCoins() << "\n";
            actual = actual->siguiente;
        } while (actual != primero);
    }
    else {
        cout << "\n>La lista se Encuentra Vacia\n\n";
    }
}
```

Este es un método simple únicamente recorremos la lista de inicio a fin y como es circular se detiene cuando vuelve a comenzar, imprimiendo la información que contiene cada nodo.

Método general de ordenamiento de datos - burbuja:

En este método se aplicó la técnica del ordenamiento burbuja por lo cual se introdujeron 2 nodos, pivote y actual, por medio de los cuales se intercambió la información que contenga cada uno dependiendo si sus valores son o no mayores o menores, de este método se utilizaron 2 para poder generar el orden ascendente y descendente, algo importante para aclarar es que no se intercambiaron posiciones en memoria sino solamente los datos.

```
pivote = listaUser.primero;
while (pivote != listaUser.ultimo) {
    actual = pivote->siguiente;
    while (actual != listaUser.primero) {
        if (pivote->getAge() > actual->getAge()) {

            nick = pivote->getNick();
            password = pivote->getPassword();
            age = pivote->getAge();
            coins = pivote->getCoins();

            pivote->setNick(actual->getNick());
            pivote->setPassword(actual->getPassword());
            pivote->setAge(actual->getAge());
            pivote->setCoins(actual->getCoins());

            actual->setNick(nick);
            actual->setPassword(password);
            actual->setAge(age);
            actual->setCoins(coins);

        }
        actual = actual->siguiente;
    }
    pivote = pivote->siguiente;
}
```

Método general de impresión de grafos – Graphviz:

```
void listaCircularUser::doGraphics() {
    string dot = "";

    dot += "digraph G {\n";
    dot += +"label=\"Estructura: Lista Circular\";\n";
    dot += +"node [shape=box];\n";

    nodoUser* aux = primero;
        do {
            dot += "N" + (aux->getNick()) + "[label=\"" + (aux->getNick()) + "\"];\n";
            aux = aux->siguiente;
        } while (aux != primero);


        dot += "{rank=same;\n";
        aux = primero;
        do {
            dot += "N" + (aux->getNick());
            dot += "->";
            aux = aux->siguiente;

        } while (aux != primero);
```

```
    ultimo = primero;
    do {
        dot += "N" + (aux->getNick());
        dot += "->";
        aux = aux->atras;

    } while (aux != ultimo);
    dot += "N" + (aux->getNick());
    dot += "}\n";


dot += "}\n";

cout << dot;

ofstream file;
file.open("listaCiruclar.dot");
file << dot;
file.close();

system(("dot -Tpng listaCiruclar.dot -o  listaCiruclar.png"));

system(("listaCiruclar.png"));
```

En este método se recorren los datos de la lista, o de la pila y se añaden como texto a un archivo.dot el cual interpretara graphviz para genera una imagen en formato png. En la primera parte debemos recorrer los nodos principales o cabeceras si se tienen, de esta forma enlazamos cada uno de ellos con los apuntadores necesarios, esto lo hacemos por medio de {rank=same}, después de esto agregamos los nodos dependientes de las cabeceras si los hubiera con el método {rank=none}, después de esto únicamente queda a discreción añadir creatividad a la forma en la que se imprime.

La clase main.cpp:

Contiene el método principal de nuestra aplicación, esta clase se encarga de inicializar los objetos que usaremos dentro del programa a lo largo de toda la ejecución. Ayudando así a la persistencia de datos en cualquier área del programa.

Esta clase es la que se encarga de generar la conexión con el lenguaje de programación Python, en el cual se consumirá por medio de una interfaz grafica de usuario, las estructuras definidas en C++.

Librería implementada para la conexión con Python

Para realizar la conexión con el lenguaje Python se ultilizo la librería Crow, la es un marco C++ para crear servicios web HTTP o Websocket. Utiliza un enrutamiento similar a Python's Flask, lo que lo hace fácil de usar. También es extremadamente rápido, superando múltiples marcos C++ existentes, así como marcos que no son C++.

Puerto de Conexión Local:

La API estará alojada en el puerto local '5000': http://localhost:5000

```
app.port(5000).multithreaded().run();
```

Método de Estado de la Aplicación:

Por medio de un método “request” desde Python podemos ejecutar este enlace a la API de C++ la cual tendrá un response textual cuando el servidor este activo y un error con código 504 cuando el servidor este inactivo.

```
crow::SimpleApp app;
CROW_ROUTE(app, "/")
    ([] {
    return crow::response("API -> UP!");
        });
```

Método de inicio de sesión:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y retornar un estado, el cual dependerá si el usuario existe y si no existe no podrá logearse, también verificara si la contraseña ingresada pertenece al usuario.

```
CROW_ROUTE(app, "/login")
    ([&listaUsers](const crow::request& req)
        {
            auto x = crow::json::load(req.body);
            if (!x)
                return crow::response(400);
            string nick = x["nick"].s();
            string pass = x["password"].s();

            nodoUser* usuario = listaUsers.searchUsers(nick);

            if (usuario == nullptr) {
                return crow::response("Usuario no encontrado");
            }

            if (usuario->getPassword() == pass) {
                return crow::response(usuario->getNick());
            }

            return crow::response("incorrecta");

        });
```

Método de Registro:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y crear u nuevo nodo para generar un nuevo usuario y asignarle los atributos de nodo usuario.

```
CROW_ROUTE(app, "/sign_up")
    .methods("POST"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);

        nodoUser* nuevo;

        string nick = x["nick"].s();
        string pass = x["password"].s();
        string age = x["edad"].s();

        if (listaUsers.searchExist(nick) != true) {
            nuevo = new nodoUser();

            nuevo->setNick(nick);
            nuevo->setPassword(pass);
            nuevo->setAge(stoi(age));

            listaUsers.addToEnd(nuevo);
            return crow::response("registrado");
        }

        return crow::response("repetido");

    });
```

Método de Editar información:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y edita por medio del método PUT la información actual de usuario.

```
CROW_ROUTE(app, "/edit")
    .methods("PUT"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);
        string name = x["name"].s();
        string nick = x["nick"].s();
        string pass = x["password"].s();
        string age = x["edad"].s();

        nodoUser* usuario = listaUsers.searchUsers(name);

        if (nick != "") {
            usuario->setNick(nick);
        }
        if(age != "") {
            usuario->setAge(stoi(age));
        }
        if (pass != "") {
            usuario->setPassword(pass);
        }

        return crow::response("datos actualizados");

    });
```

Método de Eliminar Usuario:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y edita por medio del método DELETE la información actual de usuario y elimina el nodo para eliminar la información del usuario.

```
CROW_ROUTE(app, "/delete")
    .methods("DELETE"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);
        string nick = x["nick"].s();
        string pass = x["password"].s();

        nodoUser* usuario = listaUsers.searchUsers(nick);

        if (listaUsers.deleteUser(usuario->getNick(), usuario->getPassword()) == true) {
            return crow::response("Usuario Eliminado!");
        }

        return crow::response("Eliminacion imcompleta!");

    });
```

Método de Apilar Jugadas:

Este método se encarga de realizar un request hacia la pila de jugadas y por medio de la conexión con la lista de usuarios y la pila se guarda el nombre del usuario y las jugadas realizadas en forma de pila.

```
CROW_ROUTE(app, "/up_plays")
    .methods("POST"_method)([&listaUsers,&newStack](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);

        string nick = x["nick"].s();
        string posx = x["x"].s();
        string posy = x["y"].s();

        nodoUser* usuario = listaUsers.searchUsers(nick);

        if (usuario != NULL) {

            newStack->doGamePlays(usuario, stoi(posx), stoi(posy));
            return crow::response("Juagada aniadida");

        }

        return crow::response("Usuario no encontrado");

    });
```

Método de Actualizar Monedas:

Este método se encarga de realizar un request hacia la lista circular de usuarios y actualiza el atributo monedas, del usurario por medio del método

PUT y de esta forma interactúa la interfaz grafica con la base de datos en C++.

```
CROW_ROUTE(app, "/up_coins")
    .methods("PUT"_method)([&listaUsers](const crow::request& req)
        {
            auto x = crow::json::load(req.body);
            if (!x)
                return crow::response(400);
            string nick = x["nick"].s();
            string coins = x["monedas"].s();

            nodoUser* usuario = listaUsers.searchUsers(nick);

            if (coins != "") {
                usuario->setCoins(stoi(coins));
            }

            return crow::response("Monedas actualizadas");

        });
```

Método de Carga Masiva:

Este método se encarga de realizar un request hacia la las lista circular de usuarios, hacia la cola de movimientos, hacia la lista de artículos y hacia la lista de listas de Artículos, y llama el método de carga masiva de cada uno para poder almacenar en memoria lo obtenido del archivo en formato Json.

```
CROW_ROUTE(app, "/masiva")
    .methods("POST"_method)([&listaUsers, &structList, &listItem, &queue](const crow::request& req)
        {
            auto x = crow::json::load(req.body);
            if (!x)
                return crow::response(400);

            string ruta = x["ruta"].s();

            listaUsers.loadFile(ruta);
            structList.loadFile(ruta);
            listItem.loadFile(ruta);
            queue.loadFile(ruta);

            return crow::response("Informacion Cargada!");
        });
```

Método de reporte de usuarios:

Este método se encarga de realizar un request hacia la lista circular de usuarios, y llama a los métodos de ordenamiento y retorna la lista ordenada

en forma ascendente y descendente, para su posterior visualización en formato HTML.

```
CROW_ROUTE(app, "/usersUp")
    ([&listaUsers]()
        {
            std::vector<crow::json::wvalue> temp = listaUsers.bubbleSortUP();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });


CROW_ROUTE(app, "/usersDw")
    ([&listaUsers]()
        {
            std::vector<crow::json::wvalue> temp = listaUsers.bubbleSortDW();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });
```

Método de reporte de Artículos:

Este método se encarga de realizar un request hacia la lista circular de Items, y llama a los métodos de ordenamiento y retorna la lista ordenada en forma ascendente y descendente, para su posterior visualización en formato HTML.

```
CROW_ROUTE(app, "/itemsUp")
    ([&listItem]()
        {
            std::vector<crow::json::wvalue> temp = listItem.bubbleSortUP();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });


CROW_ROUTE(app, "/itemsDw")
    ([&listItem]()
        {
            std::vector<crow::json::wvalue> temp = listItem.bubbleSortDW();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });
```

La clase Main.py

Esta es la clase principal en el lenguaje de programación Python, y se encargara del manejo de la interfaz grafica así como de la matriz dispersa

para que el usuario pueda visualizar las operaciones que realiza dentro de C++ por medio de la interfaz de la librería TKinter de Python.

La variable global base_url:

Esta variable representa el puerto local en el cual esta alojado nuestra API en C++, esta es la URL por medio de la cual interactuaremos en todo momento con nuestra base de datos y con las estructuras ya definidas.

```
base_url = "http://localhost:5000"
```

Matriz Dispersa:

Para la implementación de la estructura de la matriz dispersa se implementaron distintas estructuras de datos, tales como un nodo y una lista doblemente enlazada para las cabeceras y también un nodo interno. Tales métodos se describen a continuación como parte de la estructura principal de la matriz dispersa.

Nodo Encabezado:

Este nodo se encarga de Ocupar las posiciones en memoria de las cabeceras de la matriz disperasa.

```
class Nodo_Encabezado():
    def __init__(self, id):
        self.id: int = int(id)
        self.siguiente = None
        self.anterior = None

        self.acceso = None
```

Lista Encabezado:

Este nodo se encarga de asignar las posiciones de los nodos encabezado para que se ordenen y se generen según el tamaño y el requerimiento de espacio de la matriz dispersa.

```python
class Lista_Encabezado():
    def __init__(self, tipo):
        self.primero: Nodo_Encabezado = None
        self.ultimo: Nodo_Encabezado = None
        self.tipo = tipo
        self.size = 0

    def insertar_nodoEncabezado(self, nuevo):
        self.size += 1
        if self.primero == None:
            self.primero = nuevo
            self.ultimo = nuevo
        else:
            if nuevo.id < self.primero.id:
                nuevo.siguiente = self.primero
                self.primero.anterior = nuevo
                self.primero = nuevo
            elif nuevo.id > self.ultimo.id:
                self.ultimo.siguiente = nuevo
                nuevo.anterior = self.ultimo
                self.ultimo = nuevo
            else:
                tmp: Nodo_Encabezado = self.primero
                while tmp != None:
                    if nuevo.id < tmp.id:
                        nuevo.siguiente = tmp
                        nuevo.anterior = tmp.anterior
                        tmp.anterior.siguiente = nuevo
                        tmp.anterior = nuevo
                        break
                    elif nuevo.id > tmp.id:
                        tmp = tmp.siguiente
                    else:
                        break
```

Método obtener Encabezado:

Este método facilita la consulta de Nodo Encabezado, si existe el Nodo lo retorna, y si no existe se crea uno nuevo, este es esencial para la creación correcta de la matriz dispersa

```python
def getEncabezado(self, id) -> Nodo_Encabezado:
    tmp = self.primero
    while tmp != None:
        if id == tmp.id:
            return tmp
        tmp = tmp.siguiente
    return None
```

Nodo Interno:

Este método se encarga de obtener la asignación de un valor en memoria de un nodo interno de la matriz dispersa, estos nodos son los que se encargaran de sostener el tipo de dato almacena y las principales características de este y por medio del cual implementaremos la logia del tablero y la ubicación y características de los barcos.

```
class Nodo_Interno():
    def __init__(self, x, y, caracter, color):
        self.caracter = caracter
        self.coordenadaX = int(x)
        self.coordenadaY = int(y)
        self.color = color

        self.arriba = None
        self.abajo = None
        self.derecha = None
        self.izquierda = None
```

Matriz Dispersa:

Esta es la clase principal de la matriz dispersa y se encarga de manejar toda la lógica de recorrido y asignación de nodos cabecera, listas cabeceras y nodos internos de tal forma que se genere correctamente la matriz dispersa y todas las características que esta conlleva. La matriz dispersa es capa de administrar la memoria dinámicamente de forma visual, de modo que tanto los nodos cabeceros como los nodos internos se crea únicamente a medida que son solicitados y esos son asignados en su posición únicamente cuando la posición solicitada no existe.

```
class MatrizDispersa():
    def __init__(self, capa):
        self.capa = capa
        self.filas = Lista_Encabezado('fila')
        self.columnas = Lista_Encabezado('columna')
```

Método de Tamaño:

Este método se encarga de definir el tamaño que tendrá la matriz dispersa por medio de las listas de Cabeceras debido a que según se creen los nodos cabeceros de crearan también los espacios en memoria que definen los límites de la matriz.

```
def size(self,size):
    global matrix_size
    matrix_size = size

    for i in range(1,matrix_size+1):
        nodo_X = self.filas.getEncabezado(i)
        nodo_Y = self.columnas.getEncabezado(i)

        if nodo_X == None:
            nodo_X = Nodo_Encabezado(i)
            self.filas.insertar_nodoEncabezado(nodo_X)

        if nodo_Y == None:
            nodo_Y = Nodo_Encabezado(i)
            self.columnas.insertar_nodoEncabezado(nodo_Y)
```

Método de Inserción:

Este método se encarga de definir la inserción del nodo interno y se almacena como nodo interno de ambas listas cabeceras, las cuales son filas y columnas.

```
def insert(self, pos_x, pos_y, caracter, color):
    nuevo = Nodo_Interno(pos_x, pos_y, caracter, color)
    nodo_X = self.filas.getEncabezado(pos_x)
    nodo_Y = self.columnas.getEncabezado(pos_y)

    if nodo_X == None:
        nodo_X = Nodo_Encabezado(pos_x)
        self.filas.insertar_nodoEncabezado(nodo_X)

    if nodo_Y == None:
        nodo_Y = Nodo_Encabezado(pos_y)
        self.columnas.insertar_nodoEncabezado(nodo_Y)
```

Este método deriva en la inserción por fila y la inserción por columna las cuales se puede verificar como sigue:

```
if nodo_X.acceso == None:
    nodo_X.acceso = nuevo
else:
    if nuevo.coordenadaY < nodo_X.acceso.coordenadaY:
        nuevo.derecha = nodo_X.acceso
        nodo_X.acceso.izquierda = nuevo
        nodo_X.acceso = nuevo
    else:
        tmp : Nodo_Interno = nodo_X.acceso
        while tmp != None:
            if nuevo.coordenadaY < tmp.coordenadaY:
                nuevo.derecha = tmp
                nuevo.izquierda = tmp.izquierda
                tmp.izquierda.derecha = nuevo
                tmp.izquierda = nuevo
                break;
            elif nuevo.coordenadaX == tmp.coordenadaX and nuevo.coordenadaY == tmp.coordenadaY:
                break;
            else:
                if tmp.derecha == None:
                    tmp.derecha = nuevo
                    nuevo.izquierda = tmp
                    break;
                else:
                    tmp = tmp.derecha
```

```
if nodo_Y.acceso == None:
    nodo_Y.acceso = nuevo
else:
    if nuevo.coordenadaX < nodo_Y.acceso.coordenadaX:
        nuevo.abajo = nodo_Y.acceso
        nodo_Y.acceso.arriba = nuevo
        nodo_Y.acceso = nuevo
    else:
        tmp2 : Nodo_Interno = nodo_Y.acceso
        while tmp2 != None:
            if nuevo.coordenadaX < tmp2.coordenadaX:
                nuevo.abajo = tmp2
                nuevo.arriba = tmp2.arriba
                tmp2.arriba.abajo = nuevo
                tmp2.arriba = nuevo
                break;
            elif nuevo.coordenadaX == tmp2.coordenadaX and nuevo.coordenadaY == tmp2.coordenadaY:
                break;
            else:
                if tmp2.abajo == None:
                    tmp2.abajo = nuevo
                    nuevo.arriba = tmp2
                    break
                else:
                    tmp2 = tmp2.abajo
```

Método de Inserción automática de barcos:

Este conjunto de métodos se encarga de cuantificar la cantidad de barcos necesarios según el tamaño, este mismo método se repite en cada tipo de barco, ya sea la porta avión, el submarino, el destructor y el buque. El método Genera un numero aleatorio en cada coordenada, de tal forma que se consigue una coordenada de X y Y aleatoria desde la cual se empieza a evaluar las posibles posiciones donde se podrá situar la unidad.

```
def add_protaviones(self):

    global matrix_size, rec_pa

    row = random.randint(1,matrix_size)
    col = random.randint(1,matrix_size)

    pivote = self.get_By_Pos(row,col)

    tmp: Box =  self.boxes_head


    #derecha
    tmpr1 = self.get_By_Pos(row,col+1)
    tmpr2 = self.get_By_Pos(row,col+2)
    tmpr3 = self.get_By_Pos(row,col+3)

    #iquierda
    tmpl1 = self.get_By_Pos(row,col-1)
    tmpl2 = self.get_By_Pos(row,col-2)
    tmpl3 = self.get_By_Pos(row,col-3)

    #arriba
    tmpu1 = self.get_By_Pos(row+1,col)
    tmpu2 = self.get_By_Pos(row+2,col)
    tmpu3 = self.get_By_Pos(row+3,col)

    #abajo
    tmpd1 = self.get_By_Pos(row-1,col)
    tmpd2 = self.get_By_Pos(row-2,col)
    tmpd3 = self.get_By_Pos(row-3,col)
```

Este método valida que las posiciones aproximadas de todas las 4 direcciones principales no sean nulas saliendo dela matriz y que al estar dentro de la matriz no contengan ningún color.

```
while tmp != None:
    if (tmp.posX == pivote.posX and tmp.posY == pivote.posY and tmp.color == withe):
        op = random.randint(1,4)

        if(op == 1):
            if(tmpr1 != None and tmpr2 != None and tmpr3 != None):
                if(tmpr1.color == withe and tmpr2.color == withe and tmpr3.color == withe):
                    tmp.color = Maroon
                    tmpr1.color = Maroon
                    tmpr2.color = Maroon
                    tmpr3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        elif(op == 2):
            if(tmpl1 != None and tmpl2 != None and tmpl3 != None):
                if(tmpl1.color == withe and tmpl2.color == withe and tmpl3.color == withe):
                    tmp.color = Maroon
                    tmpl1.color = Maroon
                    tmpl2.color = Maroon
                    tmpl3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        elif(op == 3):
            if(tmpu1 != None and tmpu2 != None and tmpu3 != None):
                if(tmpu1.color == withe and tmpu2.color == withe and tmpu3.color == withe):
                    tmp.color = Maroon
                    tmpu1.color = Maroon
                    tmpu2.color = Maroon
                    tmpu3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        elif(op == 4):
            if(tmpd1 != None and tmpd2 != None and tmpd3 != None):
                if(tmpd1.color == withe and tmpd2.color == withe and tmpd3.color == withe):
                    tmp.color = Maroon
                    tmpd1.color = Maroon
                    tmpd2.color = Maroon
                    tmpd3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        else:
            self.add_protaviones()

    tmp = tmp.next
```

Es un método recursivo que por lo que al no encontrar una posición desde la coordenada aleatoria se llama a si mismo para generar una nueva coordenada e intentar nuevamente la inserción.

Tabla Hash:

Esta es la clase principal de la tabla hash y se encarga de crear el objeto tabla hash a partir del nodo llamado Nodo Hash, y su funcionamiento principal se basa en el almacenamiento y la búsqueda de objetos por medio de sus identificadores únicos los cuales son creados por la función hash:

```
def Hash(self, id_user, p_name):
    aux = 0
    for caracter in str(p_name):
        aux += ord(caracter)
    prekey = id_user+int(aux)
    key = (prekey % self.size_M)
    return key
```

Esta función mezcla el numero del ID del usuario y el nombre del objeto que desea adquirir para poder crear una llave única dentro del rango del tamaño de la tabla hash, en dado caso que la llave ya exista, produce una colisión, en este caso se utiliza el método de resolución de colisión por doble dispersión:

```
def double_dispersion(self, prekey):
    key = ((prekey % 3)+1)*self.colide % self.size_M
    return key
```

El cual genera una nueva llave a partir de la llave anterior siempre dentro del rango del tamaño de la tabla hash.

En otro caso cuando la tabla hash supera o alcanza el porcentaje de ocupación establecido en el diseño:

```
def porcentaje_ocupacion(self):
    aux = round(self.ocupados*100/self.size_M)
    return aux
```

Se utiliza una técnica llamada Re-Hash la cual consiste en buscar el siguiente número primo del tamaño de la tabla actual:

```python
def get_next_prime(self, primo_actual):
    cont = primo_actual
    cont += 1
    for n in range(2, cont):
        if cont % n == 0:
            cont += 1
    return cont
```

y este se le asigna a la tabla nueva, y como siguiente paso se copian los datos de la tabla vieja a la nueva quedando una tabla hash con mayor espacio y con los mismos datos en los mismos punteros.

```python
    def re_Hash(self):

        aux = self.get_next_prime(self.size_M)

        for i in range(self.size_M,aux+1):
            nodo_X = self.Indice.getEncabezado(i)
            if nodo_X == None:
                nodo_X = Nodo_Simple(i)
                self.Indice.insertar_nodoSimple(nodo_X)

        self.size_M = aux
```

Por último podemos observar el método de impresión de grafico de tabla hash a través de graphviz.

```python
def do_Graphics(self,type_player):
    contenido = '''digraph G{node[shape=box, fontname="Arial", fillcolor="white", style=filled] edge[style = "bold"] node[label = "''' + str(0) +'''" pos = "-1,1!"]raiz;'''

    pivote = self.Indice.primero
    posx = 0
    while pivote != None:
        contenido += '\n\tnode[label = "{}" pos="-1,-{}!" shape=box]x{};'.format(pivote.id, posx, pivote.id)
        pivote = pivote.siguiente
        posx += 1

    pivote = self.Indice.primero
    while pivote.siguiente != None:
        contenido += '\n\tx{}->x{};'.format(pivote.id, pivote.siguiente.id)
        pivote = pivote.siguiente

    contenido += '\n\traiz->x{};'.format(self.Indice.primero.id)

    pivote = self.Indice.primero
    posx = 0
    while pivote != None:
        pivote_celda : Nodo_Hash = pivote.acceso
        while pivote_celda != None:
            if pivote_celda.datos != None:
                contenido += '\n\tnode[label="Id: {}\nNombre: {}" pos="1,-{}!" shape=box]i{}_1;'.format(pivote_celda.datos.id, pivote_celda.datos.nombre,posx, pivote_celda.coordenadaX)
            pivote_celda = pivote_celda.derecha

        pivote_celda = pivote.acceso
        while pivote_celda != None:
            if pivote_celda.derecha != None:
                contenido += '\n\ti{}_1->i{}_1;'.format(pivote_celda.coordenadaX, pivote_celda.derecha.coordenadaX)
            pivote_celda = pivote_celda.derecha

        pivote_celda = pivote.acceso
        if(pivote_celda != None):
            contenido += '\n\tx{}->i{}_1;'.format(pivote.id, pivote.acceso.coordenadaX)

        pivote = pivote.siguiente
        posx += 1

    contenido += '\n}'

    dot = "Hash_{}.txt".format(type_player)
    with open(dot, 'w') as grafo:
        grafo.write(contenido)
    result = "Hash_{}.pdf".format(type_player)
    os.system("neato -Tpdf " + dot + " -o " + result)
    webbrowser.open(result)
```

Blockchain:

La clase block se encarga de recibir los atributos que se guardaran posteriormente en cada nodo del blockchain:

```
class Block:
    def __init__(self, lista: L_Simple, timestamp:str, nonce, previoushash, hash, index) -> None:
        self.timestamp = timestamp
        self.nonce = nonce
        self.previoushash = previoushash
        self.hash = hash
        self.index = index
        self.next = None
        self.lista = lista
```

La clase blockchain es la que se encarga de recopilar cada uno de los nodo Block y convertirse en un listado de blockes:

```
class Blockchain:
    def __init__(self) -> None:
        self.genesis = None
        self.actual = None
        self.prefix = "0000"
```

El método get_index se encarga de devolver el índice del nodo actual, por medio del cual se generará el orden de los blockes:

```
def get_index(self):
    return self.actual.index + 1 if self.actual else 0
```

El prefijo designa la cantidad de ceros que están al inicio de la encriptación, y puede ser modificado por medio del uso del los método getter y setter:

```
def set_prefix(self, prefix):
    self.prefix = prefix

def get_prefix(self):
    return self.prefix
```

La prueba de trabajose utiliza para eliminar los intermediarios reemplazando a los mismos por la propia comunidad que deben aprobar cada transacción a través del consenso de esta. Además de que elimina la posibilidad de transacciones maliciosas al requerir la aprobación del total del conjunto de mineros para formar el nuevo bloque dentro del blockchain:

```
def proof_of_work(self, timestamp, previoushash):
    nonce = 0
    Ubicado = ""
    while True:
            nonce += 1
            StringHash = str(self.get_index())+timestamp+previoushash+str(nonce)
            if(str(sha256(StringHash.encode('utf-8')).hexdigest()).find(self.get_prefix()) == 0):
                    Ubicado = sha256(StringHash.encode('utf-8')).hexdigest()

                    break;
    return Ubicado,nonce
```

La insercion de bloques se encarga de recopilar toda la información generada en las funciones anteriores y genera un nuevo objeto, con todos los atributos y las claves de encriptación:

```
def insertBlock(self, lista):
    timestamp = datetime.now().strftime("%d-%m-%Y::%H:%M:%S")
    previoushash = self.get_previoushash()
    hashh,nonce = self.proof_of_work(timestamp,previoushash)
    new = Block(lista, timestamp,nonce,previoushash,hashh, self.get_index())

    if self.genesis == None:
        self.genesis = new
        self.actual = self.genesis
    else:
        tmp = self.actual
        while tmp.next is not None:
            tmp = tmp.next
        self.actual.next = new
        self.actual = new
```

El método do_graphics se encarga de recorrer la lista de blockes creados en hash y extrae sus datos para formar un formato de tablas con la librería graphviz:

-

```
def do_graphics(self):
    contenido = '''digraph G{ node [shape=record, fontname="Arial"]'''
    aux = self.genesis
    while aux is not None:
        contenido += '''\nB_{} ['''.format(aux.index)
        contenido += '''shape=plain label=<<table border="0" cellpadding="0">'''
        contenido += '''<tr> <td> <table border="0" cellborder="0" cellspacing="10" >'''
        contenido += '''<tr> <td align="left">INDEX:        {}</td> </tr>'''.format(aux.index)
        contenido += '''<tr> <td align="left">TIME STAMP: {}</td> </tr>'''.format(aux.timestamp)
        contenido += '''<tr> <td align="left">NONCE:        {}</td> </tr>'''.format(aux.nonce)
        contenido += '''<tr> <td align="left">PREV_HASH:  {}  </td> </tr>'''.format(aux.previoushash)
        contenido += '''<tr> <td align="left">NEW_HASH:   {}  </td> </tr>'''.format(aux.hash)
        contenido += '''<tr> <td align="left">Shopping Information: \n </td> </tr>'''
        tmp = aux.lista.primero
        while tmp != None:
            contenido += '<tr> <td align="left">Item_ID: {} , Item_Name: {} , Item_Price: {}</td> </tr>'.format(tmp.id, tmp.nombre, tmp.precio)
            tmp = tmp.siguiente
        contenido += '''</table> </td> </tr>'''
        contenido += ''' </table>> ]'''
        aux = aux.next
    aux2:Block = self.genesis
    while aux2.next is not None:
        contenido += "B_{} -> B_{}; \n".format(aux2.index,aux2.next.index)
        aux2 = aux2.next
    contenido += '\n}'

    dot = "Blockchain.dot"
    with open(dot, 'w') as grafo:
        grafo.write(contenido)
    result = "Blockchain.png"
    os.system("dot -Tpng " + dot + " -o " + result)
    webbrowser.open(result)
```