Manual de usuario

Este documento presenta el manual de usuario en el cual se explica detalladamente cómo ejecutar cada uno de los proyectos, una guía de uso de las aplicaciones mediante imágenes y finalmente una guía de cómo implementar nuevas estrategias mediante código Typescript.

En primer lugar, los dos proyectos se pueden encontrar en la plataforma de Github tal y como se ha comentado en la memoria del trabajo. Los enlaces son los siguientes:

- Aplicación web: https://github.com/SergioPujol/TrabajoFinGrado_CandlV-Web
- Aplicación de escritorio:https://github.com/SergioPujol/TrabajoFinGrado_CandlV-App

Por una parte, en el proyecto de la aplicación web se encuentran los diferentes servidores que van a ser ejecutados, que son: el servidor web, el servidor de la base de datos y el servidor de procesamiento.

Por otra parte, se encuentra también el proyecto de aplicación de escritorio, en el cual se hallan todos los directorios de los servidores que van a ser ejecutados, el servidor de base de datos de usuarios de forma externa y, los demás, localmente. Se localiza también un archivo index.js encargado de lanzar el launcher de escritorio y una carpeta de estrategias personalizadas con un ejemplo.

Ejecución de los proyectos

En este apartado se explican las diferentes instrucciones que se deben seguir para la ejecución de cada uno de los proyectos.

En primer lugar, es necesario instalar todo el software para la ejecución de ambas aplicaciones. Primeramente, se debe instalar MongoDB (se recomienda instalar MongoDB Compass para disponer de una interfaz sencilla de utilizar) para poder iniciar una base de datos no relacional de forma local. Después de ser instalado, hay que iniciar una nueva conexión con *mongodb://localhost:27017*. Una vez se haya establecido una conexión, se deben crear las siguientes bases de datos y colecciones:

candlv	**candlv_desktop**	**candlv_users_desktop**
bots	bots	users
charts	charts	
settings	settings	
trades	strategies	
users	trades	

La primera columna se utiliza para el proyecto de aplicación web, y por ello, se ha de crear una única base de datos denominada **candlv** con las colecciones bots, charts, settings, trades y users.

En las dos columnas restantes, se encuentran las dos bases de datos para la aplicación de escritorio. En primer lugar, se encuentra **candlv_desktop** junto a las colecciones bots, charts, settings, strategies y trades; y seguidamente, la base de datos **candlv_users_desktop** junto a la única colección users.

A continuación, una vez creadas las tres bases de datos en MongoDB, hay que añadir un usuario de prueba que permita conectarnos a las aplicaciones. Por ello, en las bases de datos de **candlv** y **candlv_users_desktop** y en la colección users hay que añadir un nuevo documento con un email y una key, como se indica en la siguiente imagen:

Una vez se han configurado todas las bases de datos mediante MongoDB, es necesario instalar los paquetes npm de los proyectos. Para ello, es necesario realizar lo siguiente:

- **Aplicación web:** en el directorio principal del proyecto, se debe realizar un *npm install*. Del mismo modo, se tiene que ejecutar ese comando en los directorios de las carpetas de Server_DB, Server_Process y Web.
- **Aplicación de escritorio:** de la misma forma, se debe ejecutar el comando *npm install* en el directorio principal y en los directorios de las carpetas Server_DB, Server_DB_Users, Server_Process y Web.

Ya realizadas todas las instalaciones de los paquetes y el software necesario, es momento de iniciar las aplicaciones.

Guía de uso de las aplicaciones

Durante este apartado, se presenta una guía de cómo utilizar las aplicaciones y sus diferentes funcionalidades. Para ello, el primer paso se centra en ejecutar los proyectos.

Para la aplicación web se debe ejecutar el comando "*npm run start*" en el directorio principal del proyecto, esto iniciará todos los procesos necesarios y se podrá acceder a la web desde la dirección http://localhost:3000/.

En cambio, para la aplicación de escritorio, se deben seguir dos pasos:

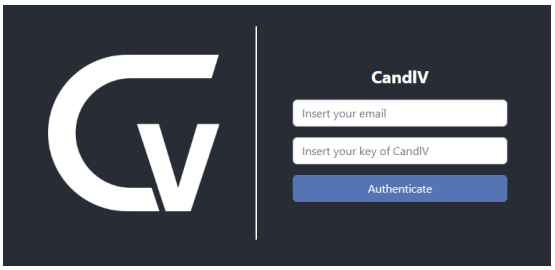
1. Ejecutar el comando "*npm run start*" en el directorio "Server_DB_Users" para iniciar el servidor de la base de datos de usuario externa.

2. Ejecutar el comando "*npm run start*" en el directorio principal del proyecto. Esto iniciará todos los servidores necesarios y lanzará el launcher de escritorio.

A continuación, se explican las diferentes funcionalidades de la aplicación de escritorio y cómo utilizarlas mediante imágenes. La aplicación web ofrece las mismas funcionalidades, aunque no permite implementar estrategias ni acceder a la página de simulación. Por lo tanto, en esta guía solo se utilizará la aplicación de escritorio.

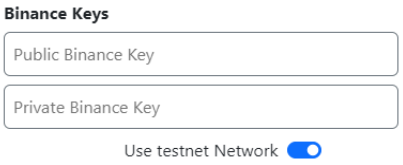
Autenticación

Primero de todo, hay que iniciar sesión mediante las credenciales, las cuales se han introducido anteriormente en la base de datos de MongoDB.

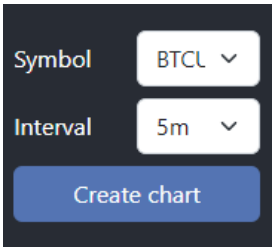


Trading

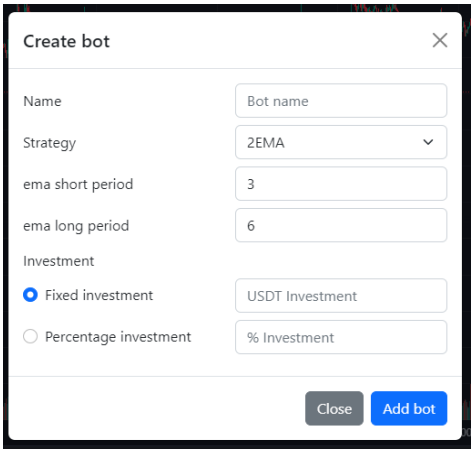
Una vez iniciada la sesión en la plataforma de trading, aparecerá una página vacía sin gráficas ni operaciones. El primer paso es añadir las credenciales de Binance, que deben haber sido solicitadas previamente en la plataforma, mediante el botón de "settings" en la parte superior derecha.



Para añadir gráficas, se debe seleccionar el símbolo y el intervalo deseados en la parte inferior derecha y hacer clic en el botón correspondiente. Se mostrará entonces una gráfica con las opciones seleccionadas. Es posible minimizar o eliminar la gráfica mediante los botones localizados en la parte superior izquierda.



Para crear un bot, se necesitará la siguiente información:

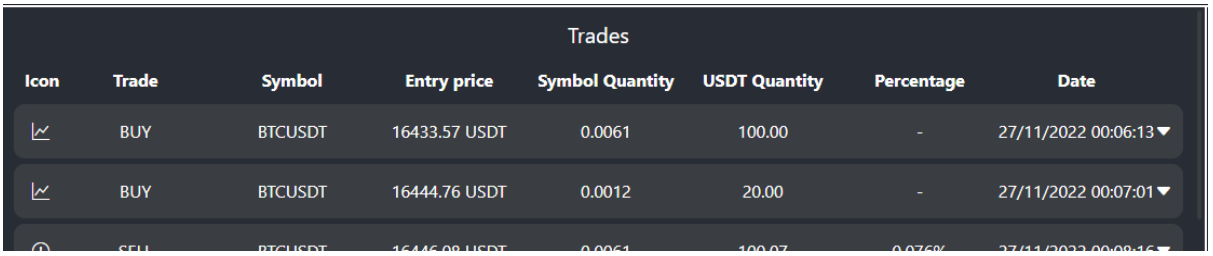


1. Nombre del bot (a elección del usuario)
2. Estrategia (con sus respectivas opciones de variables configurables)
3. Inversión en cada operación de compra (puede ser un valor fijo o un porcentaje del portafolio en USDT de Binance)

Una vez creado el bot, se puede ver su configuración haciendo clic en su contenedor. También se puede eliminar o pausar el bot en cualquier momento.

Cuando un bot está en ejecución, su situación se puede ver en el apartado "Operations", donde se pueden iniciar o detener las operaciones individualmente.

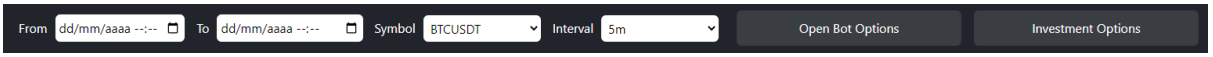
En el apartado "Trades" se pueden ver las operaciones realizadas, con información detallada sobre cada una. Al hacer clic en el contenedor de una operación, se mostrará la información del bot y la gráfica correspondiente.

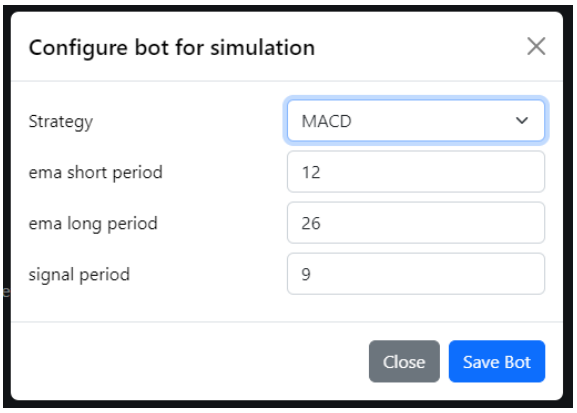


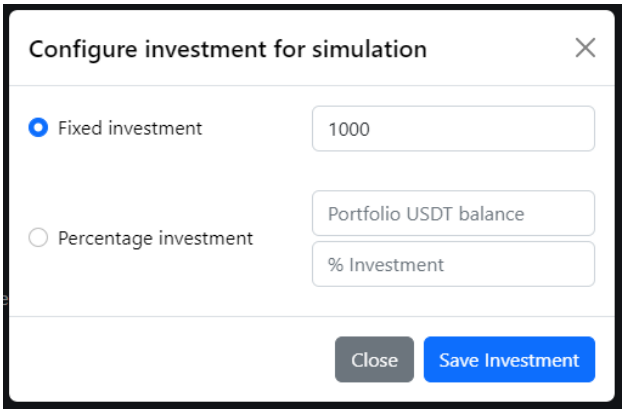
Simulation

Para realizar una simulación se deben seguir unos sencillos pasos:

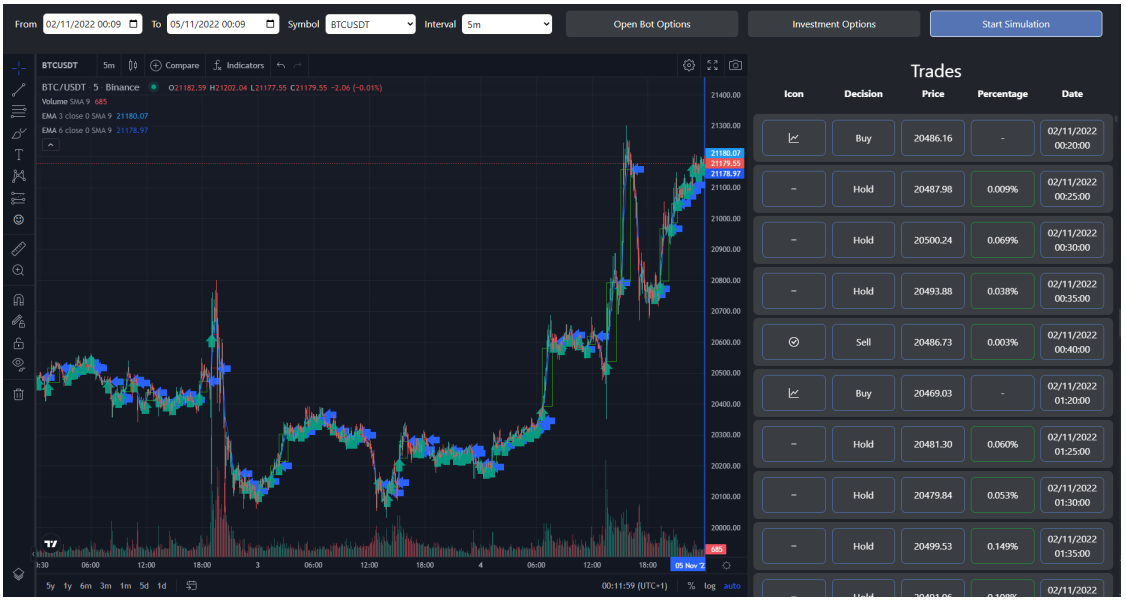
1. Seleccionar la fecha de inicio.
2. Seleccionar la fecha final.
3. Seleccionar el símbolo del cual se quiere hacer la simulación.
4. Seleccionar el intervalo de la gráfica que se va a analizar.
5. Rellenar la configuración del bot, de la misma manera que se ha realizado en el apartado anterior.
6. Seleccionar una inversión para la simulación.



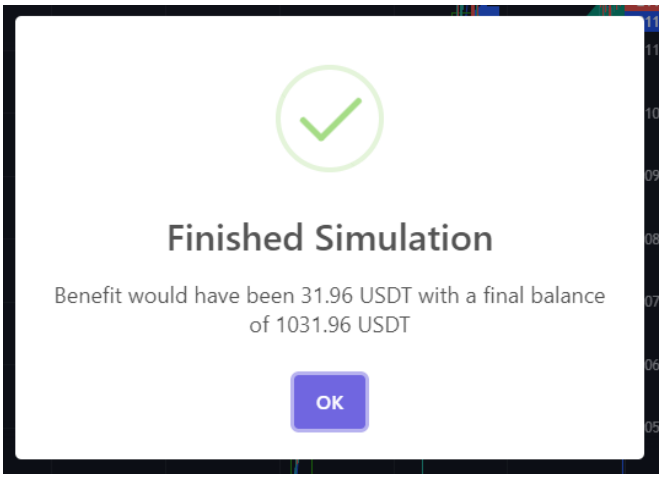




Una vez completado el análisis, se obtendrán los datos de la simulación realizada. Los resultados se mostrarán en una lista de operaciones (trades) indicando la información de cada una de ellas. Además, se podrán visualizar las operaciones en la gráfica, marcando el momento de entrada/compra y el momento de salida/venta con un rectángulo de color verde o rojo dependiendo de si el resultado es positivo o negativo, respectivamente.



Por último, aparecerá una ventana emergente indicando el beneficio que se hubiese obtenido si se hubiese iniciado un bot con la configuración establecida.



Implementación de estrategias

La implementación de estrategias mediante scripts permite hacer uso de ellas en la aplicación para la toma de decisiones automática. Cabe destacar, que para poder crear nuevas estrategias totalmente personalizadas se requiere un nivel mínimo de habilidades de programación.

Para crear una clase de estrategia se utiliza el lenguaje Typescript y se deben seguir unas normas específicas para que esta pueda funcionar.

Import de clases

Primero de todo, para poder empezar a crear el script de la estrategia es necesario importar los módulos de clases, los cuales incluyen variables utilizadas en la clase abstracta Strategies, y modelos de bot y decisiones. Estos se importan como se indica a continuación:

```
// Classes

import { Strategy } from "../Classes/Strategy";
import { Strategies } from '../Classes/Strategies';
import { Candle } from "../Classes/Candle";

// Models

import { BotModel } from "../Models/bot";
import { DecisionType } from "../Models/decision";
```

Clase

Para definir la estrategia que se va a desarrollar, es necesario crear una clase con su nombre y la cual extiende la clase abstracta principal de **Strategies** (importada anteriormente). También es necesario realizar un **export** de la clase en cuestión. Esto se realiza como se indica en la siguiente imagen. Habría que cambiar el nombre de CustomModelClass por el cual se decida nombrar a la estrategia.

```
export class CustomModelClass extends Strategies {
```

Dentro de esta clase es necesario definir un constructor, el cual tendrá como parámetros el objeto bot y la clase strategy, necesaria para el código interno. Con el bot será posible obtener su configuración y utilizar su información al respecto.

```
constructor(_bot: BotModel, _strategyClass: Strategy) {
    super(_bot, _strategyClass)
```

El BotModel es el siguiente:

```
export interface BotModel {
    client: Client | false; // not necessary of simulation

    botId: string;
    chartId: string;
    symbol: string;
    interval: string;
    strategy: string;
    botOptions: any;

    investment: { investmentType: string, quantity: string };
    simulationPeriod?: {from:string, to:string}; // only necessa
    isStrategyCustom?: boolean;
}
```

Para utilizar valores del bot en cuestión, se accede mediante ***this.bot***. Se utiliza principalmente para acceder a las botOptions de este.

La clase abstracta Strategies exige una serie de métodos que se han de definir, los cuales serán ejecutados cuando esta clase vaya a ser utilizada.

Flow method

El primero de ellos consiste en el flow, método al que se llamará cada vez que haya una nueva operación. Es el momento adecuado para actualizar los indicadores (como se podrá observar en el ejemplo al final del apartado). Este método siempre ha de llamar a ***this.updateSignal()*** al final de la función.

Como parámetro, se obtendrá la lista de velas de los últimos valores de la gráfica de la criptomoneda que se esté utilizando en el bot. La definición es la siguiente.

```
async flow(candles: Candle[]) {
    this.updateSignal()
}
```

UpdateSignal method

Es el método para actualizar la señal y decidir si comprar, vender o mantener. Para definir esta variable, hay que realizar lo siguiente:

```
this.signal = DecisionType.Buy
```

Esta señal puede ser: ***DecisionType.Buy***, ***DecisionType.Sell*** o ***DecisionType.Hold***

Durante esta función es importante hacer uso de los estados del bot, el cual indicará si actualmente se ha comprado (“InLong”) o si no hay ningún proceso en marcha (“None”).

Al final de este método es necesario llamar a ***this.decideAct()***, el cual está definido en la clase abstracta Strategies, encargado de procesar la decisión.

A continuación se mostrará un ejemplo de cómo se debe implementar el método updateSignal:

```
    updateSignal() {

        const candleClosePrice = parseFloat(this.pricedateObject!.actualPrice);
        const topLimit = this.customStrategyObject!.topLimit;
        const bottomLimit = this.customStrategyObject!.bottomLimit;

        if(this.state == 'None') {
            if(candleClosePrice < bottomLimit) this.signal = DecisionType.Buy
            else if(candleClosePrice > topLimit) this.signal = DecisionType.Sell
            else this.signal = DecisionType.Hold
        }
        else if(this.state == 'InLong' && candleClosePrice > topLimit) this.signal = DecisionType.Sell // Abort Long
        else this.signal = DecisionType.Hold

        this.decideAct()
    }
}
```

Importe de la clase desde la aplicación de escritorio

Desde el launcher de escritorio es necesario importar el archivo y definir el objeto principal de la clase. Este objeto va a tener variables constantes que serán definidas desde la aplicación y servirán a la hora de utilizar la estrategia personalizada.

En el contenedor en el cual aparece el texto “Strategy object” es necesario definir un objeto de la estrategia, este consiste en un JSON con un objeto en su interior. Este objeto tendrá el nombre de la estrategia (el cual ha de coincidir con el nombre de la clase del archivo que se va a importar), y en su interior, se definirán las variables de la estrategia, customizables desde la aplicación y utilizadas en la clase. Un ejemplo de JSON es el siguiente: **"Bollinger": { "period": 20, "times": 2 }}.**

Al clicar sobre seleccionar archivo, se permitirá abrir el explorador de archivos para seleccionar aquel que se desee importar. Y finalmente, se debe clicar en Import Strategy para su almacenamiento en la base de datos.

Ejemplo de estrategia personalizada

Para poder entender con más facilidad cómo implementar una estrategia personalizada, se presenta el siguiente ejemplo con una estrategia muy sencilla. La estrategia a implementar consiste en unos límites, donde se definirá un límite máximo y uno mínimo.

Cuando el precio esté por debajo del límite mínimo, indicará una señal de compra. En cambio, cuando el precio supere el número de límite máximo, será señal de venta.

Entonces, el objeto a declarar en la aplicación es el siguiente: **{ "CustomModelClass": { "topLimit": 1650, "bottomLimit": 1525 }}**.

Y el código del archivo Typescript de la estrategia es::

```
// Classes

import { Strategy } from "../Classes/Strategy";
import { Strategies } from '../Classes/Strategies';
import { Candle } from "../Classes/Candle";

// Models

import { BotModel } from "../Models/bot";
import { DecisionType } from "../Models/decision";


export class CustomModelClass extends Strategies {

    private customStrategyObject: { topLimit: number, bottomLimit: number } | undefined; // EXAMPLE
```

```
    constructor(_bot: BotModel, _strategyClass: Strategy) {
        super(_bot, _strategyClass)

        this.customStrategyObject = {
            topLimit: parseFloat(this.bot.botOptions.topLimit),
            bottomLimit: parseFloat(this.bot.botOptions.bottomLimit),
        }
    }
```

```
    async flow(candles: Candle[]) {

        this.pricedateObject = {
            actualPrice: candles[candles.length-1].getClose(),
            actualDate: candles[candles.length-1].getOpenTime()
        }

        this.updateSignal()
    }

    updateSignal() {

        const candleClosePrice = parseFloat(this.pricedateObject!.actualPrice);
        const topLimit = this.customStrategyObject!.topLimit;
        const bottomLimit = this.customStrategyObject!.bottomLimit;

        if(this.state == 'None') {
            if(candleClosePrice < bottomLimit) this.signal = DecisionType.Buy
            else if(candleClosePrice > topLimit) this.signal = DecisionType.Sell
            else this.signal = DecisionType.Hold
        }
        else if(this.state == 'InLong' && candleClosePrice > topLimit) this.signal = DecisionType.Sell // Abort Long
        else this.signal = DecisionType.Hold

        this.decideAct()
    }
}
```