Trabajo Práctico N°: 1 Crypto Monitor

Enunciado

Se desea construir una plataforma que permita monitorear los valores de criptomonedas, definir reglas de compra y venta que apliquen según las condiciones o variaciones del mercado.
El sistema debe permitir monitorear la variación del valor de una criptomoneda frente a otra.
Debe permitir crear reglas, que puedan combinarse y que al cumplirse dichas reglas se ejecuten acciones predefinidas de compra o venta.

Se desea contar con un rol administrador que pueda configurar reglas y acciones, y otro rol solo lectura que puede entrar y ver el estado de la billetera y las transacciones ejecutadas.

Se debe poder ver la billetera, es decir las cantidades de cada moneda
Se debe poder ver el historial de las transacciones
Sería bueno si se cuenta con algún tipo de notificaciones.
Las reglas se pueden compartir entre usuarios con permisos de lectura o lectura escritura.

Se contará luego con una api REST y websockets que brindan a modo de simulación los valores de mercado, como también así nos brinda una billetera con diferentes cantidad de criptomonedas para poder operar, por lo que habrá que integrarse a dichas APIs externas.
Dicha api REST contará con una private key o similar para poder identificarse y hashear los request a la misma.

Requerimientos:

- Autenticación:
    1) Permitir loguearse, con user/pass como así también con cuenta de google.

- Objetivos:
    1) Permitir configurar monedas para operar y monitorear valores.
    2) Permitir consultar promedios históricos de los últimos 2 días. Por ejemplo: 60 minutos, 8 horas, 1 día, etc.
    3) Permitir configurar criterios para definir el estado del mercado de cierta moneda frente a otra en base a la variación de la moneda en cierto tiempo. Los posibles estados son: *En Suba, En Baja, Estable* basado en los valores promedio diarios de los últimos X días.
    4) Permitir configurar criterios para definir el estado de la cartera para cierta moneda según el valor actual frente a un valor. Operable. No Operable. Es decir si la moneda está por arriba de cierto valor, se opera con las reglas, si esta por debajo de ese valor no quiero operar.
    5) Permitir definir estrategias, un conjunto de acciones a utilizar (agresiva, random, conservadora, etc) que se pueden configurar para ser utilizadas según el estado

del mercado. Por ejemplo si el mercado está ESTABLE, usar estrategias de mercado estable, si está en EN SUBA, usar las en suba,.

- Valores:
    1) Monitorear y guardar cada cambio de más de un 0.1% de las monedas seleccionadas (para evitar cambios insignificantes).
    2) Consultar el estado de la wallet y conocer sus montos.
    3) Permitir crear valores fijos con nombre.
    4) Permitir crear valores promedios, por ejemplo el valor promedio de la moneda X respecto a la Y de los últimos 15 minutos, 60 minutos etc.
    5) Permitir obtener el valor actual de una moneda, el valor que tuvo en cierta fecha, o el promedio en cierto periodo.

- Reglas:
    1) Se componen de condiciones en que se activan, y acciones a realizar una vez activadas.
    2) Son serializables a una estructura JSON, a definir por el cliente.
    3) Pueden hacer uso de cotizaciones históricas de las monedas, como valores numéricos o lista de los mismos.
    4) Pueden hacer uso de constantes y variables en tiempo de ejecución, consultándolas y (en caso de las variables) modificándolas.
    5) Permiten aplicar un conjunto de funciones predeterminadas para realizar cálculos aritméticos y comparaciones entre valores.
    6) Por el momento, las acciones a considerar son:
        a) BUY_MARKET
        b) SELL_MARKET
        c) SET_VARIABLE

       Donde las acciones MARKET operan inmediatamente, al mejor precio disponible. Esto puede ser otro precio diferente al de mercado, si la operación se realiza por partes.

Objetivos

- Entrega 1
    - Modelo De Dominio
    - Arquitectura Propuesta

- Entrega 2
    - Test de Integracion
    - Test Unitarios
- Entrega 3
    - Demo funcional, utilizando docker

Restricciones

- Trabajo Práctico grupal de 5 integrantes
- Implementar en Node con Typescript
- Utilizando GITLAB usando buenas prácticas
- Todas las decisiones tomadas deben estar justificadas.
- Se debe tener una amplia cobertura del código por tests.

Criterios de Corrección

- Cumplimiento de las restricciones
- Documentación entregada
- Diseño del modelo
- Diseño del código
- Test Unitarios

Se tendrán en cuenta también la completitud del tp, la correctitud, distribución de responsabilidades, aplicación y uso de criterios y principios de buen diseño, buen uso del repositorio y uso de buenas prácticas en gral.

Calendario

Jue 29/9	Presentación del TP
Jue 06/10	Entrega 1
Jue 27/10	Entrega 2
Jue 17/11	Entrega 3

Formato de serialización JSON de reglas

El usuario especifica una lista de reglas a aplicar, junto a un conjunto de variables que el usuario debería proveer para su evaluación.

```
{
	requiredVariables: ["LAST_TDD/USDT_SELL_PRICE"],
	rules: [
		...
	]
}
```

Los ejemplos a continuación muestran valores simples, en general las reglas pueden usar valores recursivamente mediante valores de tipo CALL.

Variables

Una variable es un valor asociado a un usuario, que puede ser usado para modificar el comportamiento de un conjunto de reglas.

Las variables se identifican por un string que les da nombre, y pueden contener un número, un booleano, o un string. Pueden ser definidas y modificadas por el usuario, mediante alguna API provista, o mediante la ejecución de acciones de tipo SET_VARIABLE.
La idea es que sirvan para cambiar los valores que utilizan las reglas.

Reglas

Cada regla tiene la siguiente forma:

```
{
	name: "Comprar 12 TDD/USDT siempre",
	condition: {
		type: "CONSTANT",
		value: true
	},
	action: [{
		type: "BUY_MARKET",
		symbol: "TDD/USDT",
		amount: {
			type: "CONSTANT",
			value: 12
		}
	}]
}
```

Donde condition debe evaluar a un valor booleano, y las acciones solo se ejecutan si el valor es verdadero.

Acciones

Las acciones tienen una de las siguientes formas:

```
{
	type: "BUY_MARKET",
	symbol: "TDD/USDT",
	amount: {
		type: "CONSTANT",
		value: 12
	}
}
{
	type: "SELL_MARKET",
	symbol: "TDD/USDT",
	amount: {
		type: "CONSTANT",
		value: 12
	}
}
{
	type: "SET_VARIABLE",
	name: "ValorMinimoTDD",
	value: {
		type: "CONSTANT",
		value: "12"
	}
}
```

Donde las acciones BUY_MARKET y SELL_MARKET deben recibir un valor numérico en su argumento "amount", y SET_VARIABLE puede recibir un valor de cualquier tipo en su argumento "value".

Valores

Los valores son de alguna de las siguientes formas:

```
{
	type: "CONSTANT",
	value: 1
}
```

```
{
	type: "CONSTANT",
	value: true
}
{
	type: "CONSTANT",
	value: "Hello world"
}
{
	type: "VARIABLE",
	name: "LAST_TDD/USDT_SELL_PRICE"
}
{
	type: "WALLET",
	symbol: "BTC"
}
{
	type: "CALL",
	name: "==",
	arguments: [
		{
			type: "CONSTANT",
			value: 1
		}
		{
			type: "CONSTANT",
			value: 2
		}
	]
}
```

Funciones

Las funciones a considerar son las siguientes:

- Comparación entre N valores de cualquier tipo: "==", "DISTINCT"
- Comparación entre N números: "<", "<=", ">", ">="
- Operaciones numéricas unarias: "NEGATE"
- Operaciones numéricas binarias: "-", "/"
- Operaciones numéricas de N valores: "+", "*", "MIN", "MAX", "AVERAGE", "STDDEV", "FIRST", "LAST"
- Operaciones booleanas unarias: "NOT"
- Operaciones booleanas de N valores: "AND", "OR"

De las cuales las operaciones sobre N valores:

- Todas requieren 1 argumento o más.
- "==" y "DISTINCT" indican si sus argumentos son todos iguales/todos diferentes.
- "<", "<=", ">", ">=" operan entre pares adyacentes.
    - Por ejemplo, aplicar "<" sobre [a, b, …, z] debería resultar en (a < b) && (b < c) && … && (y < z).
    - Aplicadas a un solo valor, son siempre verdaderas.
- "+", "*", "AND", "OR" operan entre todos sus argumentos.
    - Por ejemplo, aplicar "+" sobre [a, b, …, z] debería resultar en a + b + … + z.
- "MIN", "MAX", "FIRST", "LAST" son funciones de array.
- "AVERAGE" es la media aritmetica.
- "STDDEV" es la desviación estándar.

Todas las funciones reciben un array de argumentos, donde cada elemento del array es un valor. Excepcionalmente, puede especificarse el array de argumentos completo de un llamado a función numérica de N valores con los precios históricos de una moneda, de la siguiente forma:

```
// La cotización de TDD en USDT se mantuvo constante entre dos horas atrás, y una hora atrás
{
      type: "CALL"
      name: "=="
      arguments: {
            type: "DATA",
            symbol: "TDD/USDT",
            from: 7200,
            until: 3600,
            default: [{
                  type: "CONSTANT"
                  value: 2000
            },{
                  type: "CONSTANT"
                  value: 2001
            }]
      }
}
```

En caso que no haya valores en el historial solicitado, existe una clave opcional "default", que indica el array de números a usar por defecto. El mismo se puede expresar como un array de valores numéricos, o como otro uso de "DATA".

Restricciones

- Los valores que pueden asignarse en constantes o variables son números, booleanos, o strings. La presencia de cualquier otro tipo de dato es un error.
- Las reglas se evalúan en el orden especificado, evaluando cada condición antes de su acción asociada, y las acciones de una regla antes de evaluar las condiciones de la siguiente.
- En ciertos casos excepcionales, se detiene la evaluación de las reglas:
    - Lectura de una variable a la que no se asignó valor.
    - Un valor "DATA" no tiene historial de valores y no tiene valor por defecto.
    - Error al ejecutar una acción de compra/venta.
    - Errores aritméticos (división por 0, calcular desviaciones estándar de un conjunto vacío)
    - Errores de tipo (sumar strings, multiplicar por booleanos, etc)
    - Errores de número de argumentos (dividir un solo valor)

Ejemplos

```
// Si el precio BTC/USDT cae bajo un nivel determinado por una variable,
// vender todo el BTC disponible
{
      requiredVariables: [
            "LIMIT_VALUE_BTC/USDT"
      ],
      rules: [{
            name: "Escape",
            condition: {
                  type: "CALL",
                  name: "<",
                  arguments: [{
                        type: "CALL",
                        name: "LAST",
                        arguments: {
                              type: "DATA",
                              symbol: "BTC/USDT",
                              since: 3600,
                              until: 0,
                              default: {
                                    type: "VARIABLE",
                                    name: "LIMIT_VALUE_BTC/USDT"
                              }
                        }
                  },{
                        type: "VARIABLE",
                        name: "LIMIT_VALUE_BTC/USDT"
                  }]
            },
            action: [{
                  type: "SELL_MARKET",
                  symbol: "BTC/USDT",
                  amount: {
                        type: "WALLET",
                        symbol: "BTC"
                  }
            }]
      }]
}
```

```
// Si el precio de BTC/USDT aumenta más de 15% del valor de la última venta,
// vender 0.1 BTC
{
    requiredVariables: [
       "LAST_SELL_VALUE_BTC/USDT"
    ],
    rules: [{
       name: "Vender si sube 15%",
       condition: {
             type: "CALL",
             name: ">",
             arguments: [
                   {
                         type: "CALL",
                         name: "*",
                         arguments: [
                               {
                                     type: "CONSTANT",
                                     value: 1.15
                               },
                               {
                                     type: "VARIABLE",
                                     name: "LAST_SELL_VALUE_BTC/USDT"
                               }
                         ]
                   },
                   {
                         type: "CALL",
                         name: "LAST",
                         arguments: {
                               type: "DATA",
                               symbol: "BTC/USDT",
                               from: 3600,
                               until: 0
                         }
                   }
             ]
       },
       action: [{
             type: "SELL_MARKET",
             symbol: "BTC/USDT",
             amount: {
                   type: "CONSTANT",
```

```
                value: 0.1
            }
        },
        {
            type: "SET_VARIABLE",
            name: "LAST_SELL_VALUE_BTC/USDT",
            value: {
                type: "CALL",
                name: "LAST",
                arguments: {
                    type: "DATA",
                    symbol: "BTC/USDT",
                    from: 3600,
                    until: 0
                }
            }
        }]
    }]
}
```